

# Mini Project

**Title: Detecting Car's Inbuilt Bluetooth using Python and Raspberry PI**

## ***Team Members***

<b><i>Name</i></b>	<b><i>Student ID</i></b>	<b><i>Supervisor</i></b>
<i>Vishal</i>	<i>2022UCE0034</i>	<i>Dr. Ankit Dubey</i>
<i>Ashutosh Viswakarma</i>	<i>2022UCS0083</i>	
<i>Lucky</i>	<i>2022UCE0016</i>	

# Contents

1.	Problem Statement .....	3
2.	Approach.....	3
3.	How to Setup the Project in Raspberry PI and Automated setup.....	4
4.	Code Overview .....	5
5.	Data Format & Database.....	6
6.	Data Processing.....	8
7.	Results.....	9
8.	Challenges.....	10
9.	Future Scope .....	10
10.	Application in real-world scenarios: .....	11

## 1. Problem Statement

Everywhere there are many devices which have integrated Bluetooth and Wi-Fi. People have Bluetooth devices in their smart watches, Mobile phones and Car. By detecting these device's Bluetooth target, a major problem of traffic volume prediction and Also due to uniqueness of Bluetooth Mac Id location tracking of these devices also be done.

The task at hand involves developing a codebase capable of detecting these Bluetooth devices, particularly those integrated into cars. This detection aims to leverage the presence of such devices for various applications, with a primary focus on urban road intersections. The objective is to integrate this code onto a Raspberry Pi platform, thereby enabling the collection of data from diverse urban intersections. However, the challenge extends beyond mere data collection; it also encompasses the management of incoming data streams and the subsequent processing required to derive meaningful insights. By tackling this multifaceted challenge, the project aims to facilitate advancements in traffic analysis, volume prediction, and location tracking, thus contributing to the optimization of urban transportation systems.

## 2. Approach

- **Code and Device**

- **Utilization of Advanced Bluetooth Libraries:**

- We've integrated [BleakScanner](#) for Bluetooth Low Energy (BLE) devices and [PyBluez](#) for classic Bluetooth devices. This ensures comprehensive coverage and compatibility across various types of Bluetooth-enabled devices.

- **Integration of NodeMCU for Wi-Fi Scanning:**

- In addition to Bluetooth detection, we've incorporated [NodeMCU](#) devices for Wi-Fi scanning. This comprehensive approach enables us to capture a broader spectrum of device interactions within urban environments, enhancing the depth and accuracy of our data collection.

- **Raspberry Pi Model 4 for Data Collection:**

- Leveraging the processing power and connectivity features of the [Raspberry Pi Model 4](#), we've established a robust platform for data collection. This choice ensures scalability and efficiency, crucial for handling large volumes of data in real-time urban environments.

- **Data Collection:**

- **Comprehensive Data Collection:**

- Our data collection process captures [MAC addresses](#) of detected devices along with corresponding timestamps in [Unix format](#). This rich dataset serves as the foundation for subsequent analysis and insights generation.

- **Data Storage:**

- **Diverse Storage Options:**

- Two storage options are provided: MongoDB and local storage ([CSV and JSON files](#)). **For MongoDB storage**, the project interacts with a MongoDB database to store device data. This facilitates real-time data processing and analysis. **For local storage**, CSV and JSON files are created to store device data. This allows for offline data storage without internet for long time (Storage Limit of Memory card)

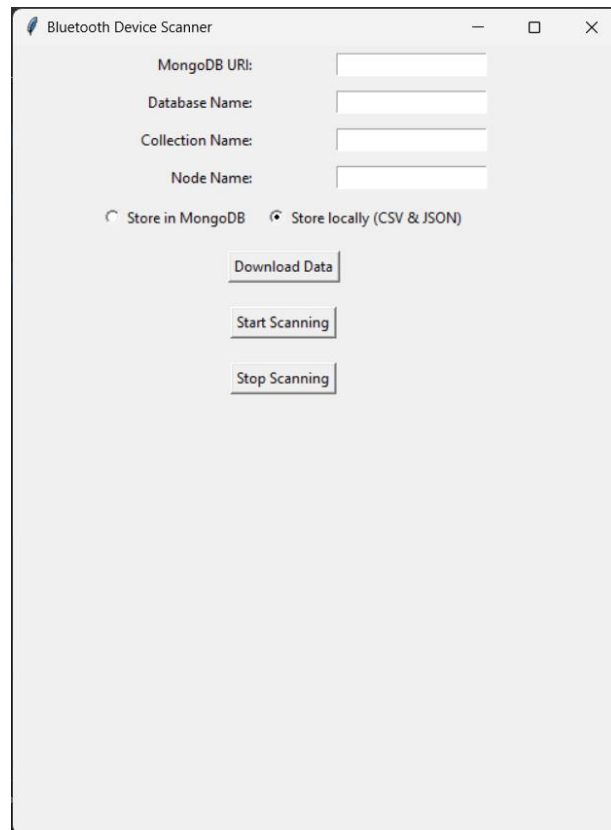
- **Application Development:**

- **User-Centric Interface Development:**

- We've developed a user-friendly interface using Tkinter, empowering users to configure settings such as MongoDB URI, database name, and collection name effortlessly. This intuitive interface enables seamless interaction and customization according to user preferences.

- **Enhanced Control and Flexibility:**

- Users have the autonomy to start and stop the scanning process, choose between MongoDB and local storage options, and download collected data for further analysis. This flexibility ensures adaptability to diverse use cases and operational requirements.



*Figure 1 Application Interface for Bluetooth and Wi-Fi node's data collection*

- **Data Processing:**

- **Data Streamlined Data Processing:**

Our data processing pipeline involves robust filtering mechanisms to extract relevant data points. By focusing on devices detected between specified locations within defined timeframes, we ensure the accuracy and relevance of processed data for subsequent analysis.

- **Timezone Conversion for Improved Interpretation:**

We've implemented timezone conversion to convert Unix timestamps to Indian Standard Time (IST), enhancing the interpretability and contextual relevance of the processed data. This conversion facilitates seamless integration with local timeframes, aiding in traffic analysis, volume prediction, and location tracking.

### 3. How to Setup the Project in Raspberry PI and Automated setup

- Update and upgrade system packages.

`sudo apt update`

`sudo apt upgrade -y`

- Install Necessary Packages:

`sudo apt install -y python3 git python3-pip`

- Install necessary packages like Python, Git, and virtual environment.

- **Set Git Configuration:**

`git config --global user.Name "YourUserName"`

`git config --global user.Email YourEmail@example.com`

- **Clone the Project Repository:**

`git clone https://github.com/Radio-active-Boys/MiniProject.git`

- Download external site packages and move them to the Raspberry Pi Python directory.

Download the site packages from link

<https://drive.google.com/file/d/16De0qRHSMxURdS5kZN9AfhdkIS1x-hE4/view?usp=sharing>

and move the package into raspberry pi python directory as a sudo

- Install Visual C on Linux if required.

<https://www.geeksforgeeks.org/how-to-install-visual-c-on-linux/amp/>

- Navigate to the project directory

`cd MiniProject`

- Install required Python packages using pip install.  
**`pip install dnspython Flask pymongo asyncio bleach python3-tk pyserial python-dotenv`**
- Set MongoDB URI environment variable.  
Navigate to cd ScannerAPI\Backend\.env  
**`MONGODB_URI=mongodb+srv://your_username:your_password@cluster1.Opamvyh.mongodb.net/your_database_name?retryWrites=true&w=majority`**  
Make sure to replace placeholders like your\_username, your\_password, and your\_database\_name with your actual MongoDB credentials. Also, adjust paths and commands as necessary based on your project structure and requirements.
- **To automate the setup process** outlined in the provided instructions, create a **shell script** (.sh file) containing all the necessary commands.  
To download .sh file of project click link below (CTRL +)  
<https://drive.google.com/file/d/1JiWKXB7yBPKBILsMpn2lms3EE3Yv7Oko/view?usp=sharing>  
Save the above script as **setup.sh** or any other prefer name. Then, make it executable using the following command:  
**`chmod +x setup.sh`**  
Finally, execute the script by running or clicking on setup.sh:  
**`./setup.sh`**  
This script will automate the entire setup process on your Raspberry Pi, including updating packages, installing necessary software, configuring Git, cloning the project repository, downloading and moving external site packages, installing Visual Studio Code & C++, installing required Python packages, setting MongoDB URI environment variable, and running the application.

#### 4. Code Overview

- **Data Collection and Storage:**
  - **Libraries Used:**
    - asyncio, datetime, threading, tkinter: For asynchronous tasks, date/time handling, multithreading, and GUI development, respectively.
    - bleak: For Bluetooth Low Energy scanning.
    - pybluez: For classic Bluetooth scanning.
    - serial: For serial communication with NodeMCU devices.
    - pymongo: For MongoDB interaction.
    - dotenv: For loading environment variables from a .env file.

```
import os
import asyncio
from datetime import datetime
import tkinter as tk
from bleak import BleakScanner
import bluetooth
import threading
from pymongo import MongoClient
from dotenv import load_dotenv
import csv
import json
from tkinter import filedialog
from serial.tools.list_ports import comports
import time
import serial
import re

from flask import Flask, jsonify
from pymongo import MongoClient
from dotenv import load_dotenv
# Load environment variables from .env file
load_dotenv()
```

*Figure 2 Python Library Used*

- **Classes:**

- **MongoDBStorage:**

- Handles interactions with MongoDB for storing device data.

- Uses pymongo library to connect to a MongoDB database and store/retrieve data.

- Implements a method discover\_devices() for continuous scanning and storing devices in MongoDB.

- Data collected includes MAC addresses of detected devices and corresponding timestamps in Unix format.

- **LocalStorage:**

- Manages local storage of data in CSV and JSON formats.

- Offers methods for storing data locally and choosing directory locations.

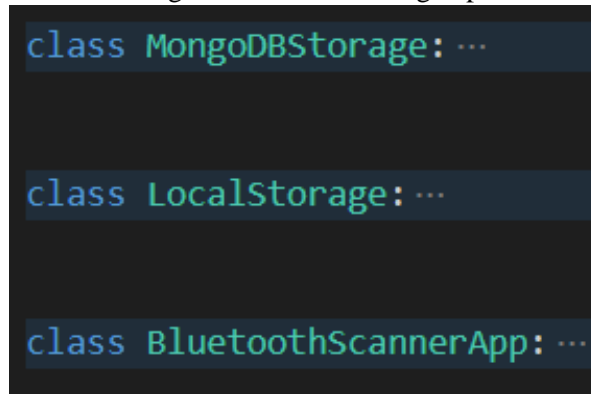
- Uses CSV and JSON libraries for writing data to respective file formats.

- **BluetoothScannerApp:**

- Provides a GUI interface using Tkinter for user interaction.

- Allows users to configure storage options, start/stop scanning, and download collected data.

- Handles MongoDB and local storage operations based on user selections.



*Figure 3 Different Classes in Code*

- **Data Processing:**

- **Functions:**

- convert\_unix\_to\_realtime(data):

- Converts Unix timestamps in the provided data to Indian Standard Time (IST).

- Uses datetime and timezone libraries to handle timezone conversion.

```
def convert_unix_to_realtime(data):
    ist = timezone(timedelta(hours=5, minutes=30)) # Indian Standard Time (IST)
    for item in data:
        for key in item:
            if isinstance(item[key], list):
                item[key] = [datetime.fromtimestamp(timestamp, ist).strftime('%Y-%m-%d %H:%M:%S') for timestamp in item[key]]
    return data
```

*Figure 4 Function to convert Unix Timestamp to IST*

- **Main Function:**

- main():

- Loads data from an input file, converts timestamps, and writes processed data to an output file.

## 5. Data Format & Database

Here the data is non-relational for such data JSON format is best suited. So the data format we choose is JSON type where we have docs { } and each docs its various field " "

```
{ "MAC_add": " D0:49:7C:4E:F0:A2",
  "Node-1": [1712388064, Unixtime],
  "Node-2": [1712388515, Unixtime],
```

```

    },
    {
      "MAC_add": "D0:49:7C:4E:F0:A2",
      "chais": [
        1712388064,
        1712388092,
        1712388123,
        1712388149,
        1712388176
      ],
      "neon": [
        1712388515,
        1712388541,
        1712388562,
        1712388587,
        1712388691,
        1712388748,
        1712388786,
        1712388812
      ]
    }
  ],
}

```

Figure 5 Data Format

For storing non-relational such data, we use MongoDB database. Where we can easily handle data handling and to get all data results in desire format (CSV and JSON)

- Database Operations in MongoDB
  - Querying Data with Multiple Fields: To query data where multiple fields exist, you can use the \$exists operator within a \$and condition.

```
Data.find({"$and": [{"Node-1": {"$exists": True}}, {"Node-2": {"$exists": True}}]})
```

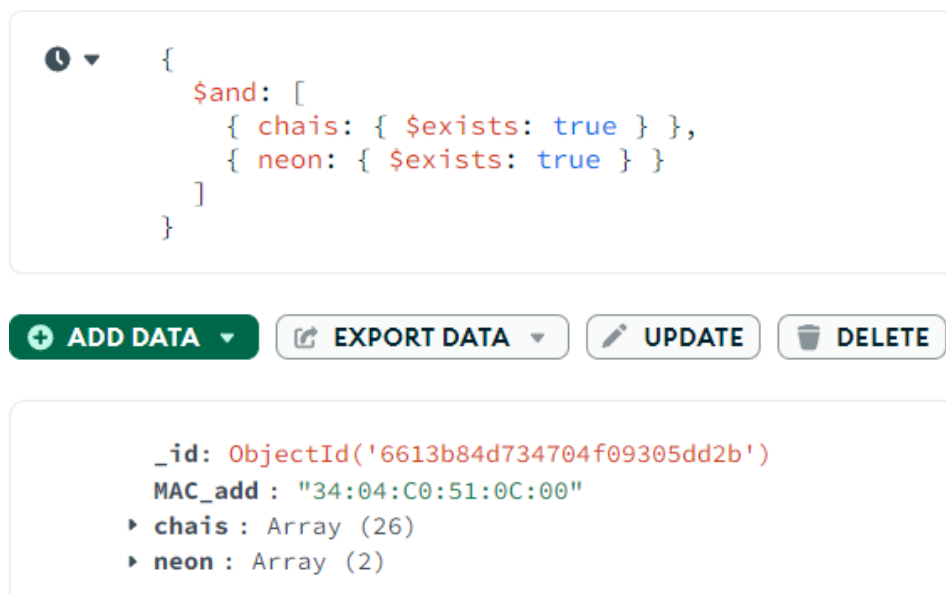
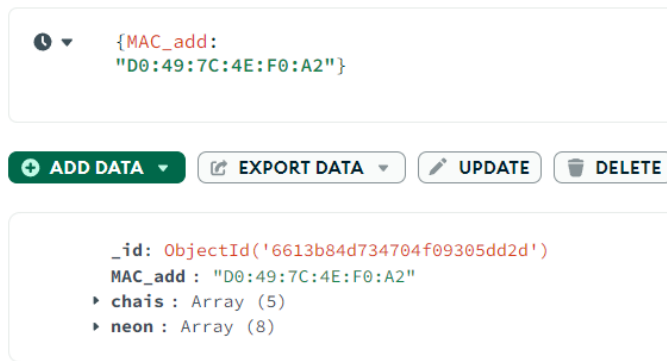


Figure 6 Command to filter devices at between nodes

This allows you to find those Devices where both Node-1 and Node-2 exist.

- Query Data: Use find() method with optional query parameters.

```
Data.find("MAC_add":"D0:49:7C:4E:F0:A2")
```



*Figure 7 To find Mac Address from data base*

This allow you a target Mac Address from the data

- Insert Data: Use insert\_one() or insert\_many() methods.

```
Data.insert_one({"MAC_add": "00:18:CE:2F:FD:77", "Node-1": [1712388064, Unixtime]})
```

- Update Data: Use update\_one() or update\_many() methods.

```
Data.update_one({"MAC_add": "00:18:CE:2F:FD:77"}, {"$set": {"Node-2": [1712388515, Unixtime]}})
```

- Delete Data: Use delete\_one() or delete\_many() methods.

```
Data.delete_one({"MAC_add": "00:18:CE:2F:FD:77"})
```

## 6. Data Processing

- **Raw Data:**

The raw data comprises MAC IDs (Media Access Control identifiers) of devices detected at specific locations along with Unix timestamps. These timestamps represent the time at which the devices were detected.

- **Filter Data:**

The filtration process involves selecting data points where devices are detected between two specified locations.

For instance, consider a scenario where Bluetooth and Wi-Fi devices are scanned at different nodes or intersections along urban roads. The filter identifies devices detected between two nodes within a given timeframe.

The filter ensures that only relevant data points, representing device movements between specific locations, are retained for further analysis.

```
{
  "MAC_add": "00:18:CE:2F:FD:77",
  "chais": [
    1712388064,
    1712388092,
    1712388123,
    1712388149,
    1712388176,
    1712391748
  ],
  "neon": [
    1712388515,
    1712388541,
    1712388562,
    1712388587,
    1712388691,
    1712388786,
    1712388812
  ]
},
```

*Figure 8 Filter Data Devices between both node*

- **Processed Data:**

Once the filter identifies the relevant data points, the processed data is generated.



Processed data retains MAC IDs and corresponding timestamps but adjusts the timestamps to Indian Standard Time (IST) for better comprehension and analysis. This conversion to IST facilitates easier interpretation of the data in the local time zone, aiding in traffic analysis, volume prediction, and location tracking.

```
MAC_add : "76:70:D5:A5:39:00"
▼ Node1 : Array (5)
  0: "2024-04-25 15:27:12"
  1: "2024-04-25 15:30:06"
  2: "2024-04-25 15:30:24"
  3: "2024-04-25 15:31:36"
  4: "2024-04-25 15:45:02"
▼ Node2 : Array (2)
  0: "2024-04-25 15:43:42"
  1: "2024-04-25 15:44:08"
```

Figure 9 Processed Data in IST

• **Data File Formats:**

Both raw and processed data are available in JSON and CSV formats, offering flexibility in data storage, retrieval, and analysis. JSON (JavaScript Object Notation) provides a structured format for storing data in key-value pairs, facilitating easy parsing and manipulation. CSV (Comma-Separated Values) format offers tabular data representation, suitable for spreadsheet software and statistical analysis tools.

• **Querying Data:**

MongoDB, a non-relational database, is utilized for storing and querying data. Queries are constructed to retrieve data based on specific criteria, such as the existence of MAC IDs at different nodes or intersections. MongoDB's query language enables complex operations, such as finding devices detected at multiple locations within a defined timeframe.

**7. Results**

The project yielded insightful results pertaining to urban traffic dynamics and Bluetooth device detection. Notably, on April 5, 2024, the code successfully identified a moving car with a MAC address of D0:49:7C:4E:F0:A2, traveling at a speed of 40 kilometers per hour. Analysis of raw data indicated a total of 58 scans, with 36 instances of the car being detected. Furthermore, on April 6, 2024, data collection between two distinct locations, Chai Sutta Bar and NEON The Bamboo Restaurant, within a one-hour timeframe, revealed 320 devices detected at the former and 520 at the latter. Interestingly, between these locations, 98 devices were identified, offering valuable insights into traffic patterns and movement between specific urban nodes. Similarly, on April 26, 2024, data collected between two sides of a bridge over a half-hour period illustrated 167 devices detected at the IIT Jammu side, 164 at the Grillin side, and 11 devices between both locations. These findings underscore the project's efficacy in capturing and analyzing real-time traffic volume and movement, laying the groundwork for informed urban planning and traffic management strategies.

**Day: 5 April, 2024**

Moving Car detected by code at speed of 40 KMs **MAC: D0:49:7C:4E:F0:A2** ( [Raw Data](#) )

Mac ID	Total Number of Scan	Car Detected
D0:49:7C:4E:F0:A2	58 ( Car Goes Back and For and in between one-two scan of Bluetooth Car cross Device only one time )	36 ( <a href="#">Processed Data</a> )

**Day: 6 April, 2024**

We collect data between two locations [Chai Sutta bar](#) and [NEON The Bamboo Restaurant](#) at a distance difference of 600m for one hour. ( [Raw Data](#) )

Detected Devices at <a href="#">Chai Sutta bar</a> ( 1hour )	Detected Devices at <a href="#">NEON The Bamboo Restaurant</a> ( 1hour )	Detected Devices between both location ( 1hour )
320	520	98
<a href="#">Link to Processed Data Indian Time</a>	<a href="#">Link to Processed Data in Indian Time</a>	<a href="#">Link to Processed Data in Indian Time</a>

**Day: 26 April, 2024**

We collect data between two sides bridge at a distance difference of 300m for half hour. ( [Raw Data](#) )

Detected Devices at IIT Jammu Side ( 0.5hour )	Detected Devices at Grillin Side ( 0.5hour )	Detected Devices between both location ( 0.5hour )
167	164	11
<a href="#">Link to Processed Data Indian Time</a>	<a href="#">Link to Processed Data in Indian Time</a>	<a href="#">Link to Processed Data in Indian Time</a>

## 8. Challenges

- Latency in Bluetooth Device Detection:** Despite leveraging advanced libraries and Raspberry Pi's capabilities, there remains a challenge in reducing the latency for detecting Bluetooth devices. Improving the efficiency of the scanning process through optimization techniques and potentially exploring alternative hardware configurations could mitigate this issue. Implementing asynchronous scanning mechanisms or parallel processing could expedite device detection, enhancing the real-time nature of the system.
- Enhancing Penetration Rate:** While the project aims to capture a diverse range of Bluetooth-enabled devices, including those integrated into cars, there's a need to address the low penetration rate, particularly in dense urban environments. Exploring strategies to improve signal strength or employing additional scanning nodes strategically positioned across intersections could enhance coverage and increase the detection rate. Collaborating with urban infrastructure stakeholders to integrate detection mechanisms into existing infrastructure could also bolster the penetration rate.
- Overcoming Device Limitations:** The project encounters limitations inherent to the hardware and software used, impacting the scalability and robustness of the system. Some devices like I-Phone have security features which don't allow to scan these devices. To address this, continuous refinement and optimization of the codebase and hardware configurations are essential. Additionally, exploring advancements in Bluetooth technology and considering alternative detection methods, such as radio frequency identification (RFID) or computer vision, could provide more comprehensive and reliable data collection capabilities.

## 9. Future Scope

With the foundation laid by the current project, there are numerous avenues for further enhancement and expansion, incorporating improvements based on the challenges encountered and emerging opportunities. Here's a comprehensive outline of the future scope, enriched with refinements and new directions:

- Correlation Study between Traffic Volume and Number of Detection Devices:**  
Leveraging the wealth of data collected from Bluetooth and Wi-Fi devices, future iterations of the project can delve deeper into understanding the correlation between traffic volume and the presence of detection devices. By employing advanced data analytics techniques, including machine learning algorithms, we can uncover nuanced patterns and relationships. This deeper insight can inform urban planners and transportation authorities, facilitating more informed decision-making regarding traffic management and infrastructure development.
- Location Tracing and Movement Analysis:**  
Building upon the foundation of device detection, the project can evolve to incorporate sophisticated location tracing capabilities. By tracking the movement patterns of detected devices over time, we can gain valuable insights into traffic flows, congestion hotspots, and commuting behaviors. This granular understanding can

enable proactive measures to optimize traffic routes, enhance road safety, and alleviate congestion in urban areas. Additionally, by integrating GPS data or leveraging advanced triangulation techniques, we can enhance the accuracy and reliability of location tracking.

- **Development of Live Traffic Volume Analyzer Based on Distributed Nodes:**

To achieve real-time monitoring and analysis of traffic volume, a distributed network of detection nodes can be deployed across urban environments. Each node, equipped with Bluetooth and Wi-Fi scanning capabilities, collaborates to create a comprehensive picture of traffic dynamics in real-time. By harnessing the power of edge computing and cloud technologies, data from distributed nodes can be aggregated, processed, and analyzed in real-time. This enables the development of live traffic volume analyzers, dashboards, and predictive models, empowering stakeholders with actionable insights for dynamic traffic management and decision support.

## **10. Application in real-world scenarios:**

- **Traffic Management and Optimization:**

By leveraging the data collected from Bluetooth and Wi-Fi devices, urban planners and transportation authorities can gain a comprehensive understanding of traffic dynamics. This includes real-time monitoring of traffic volume, identification of congestion hotspots, and prediction of traffic patterns. Armed with this information, authorities can implement targeted interventions such as adjusting signal timings, rerouting traffic, or optimizing public transportation routes to alleviate congestion and improve overall traffic flow.

- **Route Planning and Navigation:**

Mobile navigation applications can integrate real-time traffic data to provide users with optimized routes based on current traffic conditions. By analyzing historical traffic patterns and live data from Bluetooth and Wi-Fi devices, these applications can offer personalized route recommendations that minimize travel time and avoid congested areas. This enhances the efficiency of commuting and reduces fuel consumption and emissions associated with traffic congestion.

- **Urban Infrastructure Development:**

Data collected from Bluetooth and Wi-Fi devices can inform the design and planning of urban infrastructure projects. By analyzing traffic patterns and movement trends, city planners can identify areas with high traffic volume and prioritize infrastructure upgrades such as road expansions, intersection improvements, or the implementation of dedicated lanes for public transportation and bicycles. This proactive approach to infrastructure development can enhance road safety, reduce commute times, and improve overall urban livability.

- **Emergency Response and Disaster Management:**

During emergencies or natural disasters, real-time traffic data can be invaluable for emergency response teams and disaster management agencies. By monitoring traffic flow and identifying congestion points, authorities can optimize evacuation routes, deploy resources effectively, and ensure timely assistance to affected areas. Additionally, location tracking capabilities can aid in locating and rescuing individuals in distress, enhancing the efficiency and effectiveness of emergency response efforts.

- **Smart City Initiatives:**

The project aligns with broader smart city initiatives aimed at harnessing technology to improve urban livability and sustainability. By deploying IoT sensors and data analytics platforms, cities can create dynamic traffic management systems that respond to real-time conditions. This includes adaptive traffic signal control, dynamic pricing for parking and tolls, and the promotion of alternative modes of transportation such as public transit, cycling, and ride-sharing. By fostering a holistic approach to urban mobility, smart cities can reduce traffic congestion, lower emissions, and enhance quality of life for residents.