Public Transportation Optimization

Project Overview:

The goal of this project is to enhance the efficiency and user experience of public transport systems using IoT devices for real-time data collection and a Python script for data analysis and optimization.

Components:

1. IoT Devices:

- GPS Trackers: Installed on public transport vehicles for real-time location tracking.
- Passenger Counters: Installed at entry/exit points to monitor occupancy levels.

2. Data Transmission:

IoT Gateway: Acts as a bridge between IoT devices and cloud services, facilitating data transfer securely.

3. Cloud Platform:

Cloud Service (e.g., AWS, Azure): Hosts the data and provides necessary computing power for analysis.

4. Python Script:

Data Analysis: Processes and analyzes real-time data to optimize routes, schedules, and occupancy levels.

Project Steps:

1. Hardware Setup:

- Install GPS trackers on public transport vehicles.

- Deploy passenger counters at entry/exit points of vehicles.

2. IoT Device Integration:

- Connect IoT devices to the IoT gateway for data transmission.

- Ensure secure communication protocols (e.g., MQTT, HTTPS) are implemented.

3. Data Collection:

Collect real-time data from GPS trackers (location, speed) and passenger counters (occupancy levels).

Store data in the cloud platform.

4. Data Processing:

Develop a Python script to process the collected data.

Filter and clean data to remove noise and anomalies.

5. Route Optimization:

Analyze GPS data to identify optimal routes based on traffic conditions, congestion, and passenger demand.

6. Schedule Optimization:

Analyze historical data to create dynamic schedules that adapt to changing demand patterns.

7. Occupancy Management:

Use passenger counter data to monitor occupancy levels and provide real-time updates to passengers.

8. Alerts and Notifications:

Implement alerts for critical events (e.g., route deviations, overcrowding) to inform both operators and passengers.

9. User Interface:

Develop a user interface (web or mobile app) to display real-time information to passengers (e.g., estimated arrival times, occupancy levels).

10. Testing and Validation:

Conduct thorough testing to ensure the system operates reliably under various conditions.

11. Deployment and Integration:

Deploy the optimized system on a pilot route for initial testing and user feedback.

12. Scaling and Expansion:

Once validated, scale the system to cover additional routes and vehicles.

Expected Outcomes:

- Improved public transport efficiency and reliability.
- Enhanced passenger experience with real-time updates.
- Reduced environmental impact through optimized routes.

Conclusion:

This Public Transport Optimization project aims to revolutionize urban mobility through the integration of IoT devices and data-driven decision-making. By deploying GPS trackers and passenger counters, coupled with a

Python-based optimization script, we anticipate a significant improvement in the efficiency and user satisfaction of public transportation services.

```
# Import necessary libraries
import pandas as pd
import geopy.distance
import datetime
import time

# Load real-time data from IoT devices (GPS trackers and passenger counters)
# Sample data: vehicle_id, timestamp, latitude, longitude, speed, passenger_count
data = pd.read_csv('realtime_data.csv')

# Define key variables
MAX_OCCUPANCY = 50  # Maximum passenger capacity of a vehicle
MIN_SPEED = 5       # Minimum speed to consider the vehicle moving

# Data Preprocessing
# Filter out stationary vehicles and incomplete data
data = data[data['speed'] > MIN_SPEED]

# Calculate distance between consecutive GPS data points
data['distance'] = data.apply(lambda row: geopy.distance.distance((row['latitude'], row['longitude']), (row['latitude'].shift(), row['longitude'].shift())).m, axis=1)

# Route Optimization
# Find the best route based on traffic conditions and demand
# (You can use various algorithms like Dijkstra's, A*, etc. for this)
optimized_route = your_route_optimization_algorithm(data)

# Schedule Optimization
# Analyze historical data to create dynamic schedules
# Adjust departure times based on demand patterns
# You can use machine learning models for prediction
optimized_schedule = your_schedule_optimization_algorithm(data)

# Occupancy Management
```

```
# Monitor and manage vehicle occupancy
while True:
    # Real-time occupancy updates from passenger counters
    passenger_counts = get_passenger_counts()
    # Check if occupancy exceeds the maximum capacity
    for vehicle_id, count in passenger_counts.items():
        if count > MAX_OCCUPANCY:
            alert_overcrowding(vehicle_id)

    time.sleep(60)  # Check occupancy every minute

# Real-time alerts and notifications
def alert_overcrowding(vehicle_id):
    current_time = datetime.datetime.now()
    print(f"ALERT: Vehicle {vehicle_id} is overcrowded at {current_time}")

# Function to get real-time passenger counts from IoT devices
def get_passenger_counts():
    # Implement logic to collect passenger counts from passenger counters
    passenger_counts = {}  # Dictionary with vehicle_id as keys and passenger
count as values
    return passenger_counts

# Save the optimized route and schedule data for use in the user interface
optimized_route.to_csv('optimized_route.csv', index=False)
optimized_schedule.to_csv('optimized_schedule.csv', index=False)
```