PUBLIC TRANSPORT OPTIMIZATION

INTROUDCTION:

Public transport is an essential part of modern cities, offering a convenient and environmentally friendly way for people to move around. However, managing and optimizing public transportation systems can be quite challenging. This is where the Internet of Things (IoT) comes into play. IoT technology is transforming the way we design, operate, and improve public transportation. In this context, IoT involves connecting various devices and sensors to the internet to collect and share data, allowing public transport systems to become smarter, more efficient, and more user-friendly.

GATHERING DATA:

Gathering data for public transport optimization using IoT involves collecting various types of data from different sources, including sensors, devices, and systems. The collected data can be used for real-time monitoring, analysis, and decision-making to improve the efficiency and quality of public transportation.

- **Vehicle Sensors:**

  GPS/Location Data: Collect real-time location data from GPS sensors installed in vehicles to track their positions and movements.

  Speed and Acceleration: Measure vehicle speed and acceleration to optimize routes and reduce fuel consumption.

- **CCTV and Surveillance Systems:**

  Implement surveillance systems for security, incident detection, and video analysis to ensure passenger safety.

- **Infrastructure Sensors:**

  Railroad Track Sensors: Install sensors on tracks to detect defects and wear, ensuring safe and efficient rail transport.

  Traffic Lights and Control Systems: Use sensors to manage traffic signals to give priority to public transport vehicles.

  Energy Consumption: Monitor energy usage in stations and vehicles to optimize energy efficiency.

- **Traffic and Environmental Data:**

  Traffic Flow: Utilize traffic sensors and cameras to monitor road conditions and traffic congestion.

  Weather Data: Gather real-time weather data to anticipate weather-related disruptions and optimize schedules.

Air Quality: Monitor air quality and emissions to assess environmental impact.

- **Passenger Data:**

  Ticketing and Payment Systems: Collect data on ticket sales, payments, and fare collection to optimize pricing and revenue.

  Passenger Flow: Analyze how passengers move within stations and vehicles to improve station layout and boarding processes.

PREPROCESSING DATASET:

preprocessing a dataset for public transport optimization using IoT typically involves several steps. These steps are essential to ensure that the data is in a suitable format and quality for analysis and modeling.

- **Data Collection:**
  Collect data from IoT sensors, transit infrastructure, and other relevant sources. Ensure that the data is well-documented, and you understand the meaning of each data field.Ensure that the dataset is in a structured format, such as CSV, JSON, or a database, for ease of processing.

- **Data Cleaning:**
  Address missing values: Identify and handle missing data points. Depending on the dataset, you can choose to fill in missing data, remove rows or columns with missing data, or use imputation techniques.
  Outlier detection and handling: Identify and deal with outliers that might skew analysis. You can choose to remove outliers or use appropriate statistical techniques for handling them.

- **Data Transformation:**
  Convert data types: Ensure that data types are appropriate for analysis. For instance, convert date and time data to datetime objects for time series analysis.
  Normalize or scale data: Normalize numerical features to have a standard scale for machine learning models.

- **Data Integration:**
  Merge or join datasets if you have data from multiple sources that need to be combined for analysis.

- **Data Visualization:**
  Visualize the dataset to gain an initial understanding of its characteristics. Use libraries like Matplotlib, Seaborn, or Plotly to create various plots, histograms, and time series graphs.

- **Data Splitting:**
  If you plan to use machine learning, split the dataset into training, validation, and test sets to assess model performance.

- **Data Preprocessing:**
  Encode categorical variables: Convert categorical variables into a numerical format using techniques like one-hot encoding or label encoding.
  Time series data handling: If your dataset includes time series data, consider time-based splitting and lag features for time series forecasting.
  Feature scaling: Apply feature scaling techniques such as Min-Max scaling or Standardization to ensure that features have a consistent scale.

Program:

```
import pandas as pd

# Load the dataset
df = pd.read_csv('public_transport_data.csv')

# Data cleaning and preprocessing steps
#  remove missing data
df = df.dropna()

# Convert date and time columns to datetime objects
df['timestamp'] = pd.to_datetime(df['timestamp'])

# Feature scaling
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df['scaled_feature'] = scaler.fit_transform(df[['feature']])

# Encoding categorical variables (if applicable)
df = pd.get_dummies(df, columns=['categorical_column'])

# Data splitting for machine learning
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df[['features']], df['target'],
test_size=0.2)

# Store the preprocessed dataset
df.to_csv('preprocessed_public_transport_data.csv', index=False)
```

PYTHON SCRIPT:

Python script for real-time location and ridership data transmission to a transportation information platform requires careful planning and integration with IoT sensors and the platform's APIs. Below is a simplified example that demonstrates how you might send location and ridership data using Python and the MQTT (Message Queuing Telemetry Transport) protocol to an IoT platform.

Before implementing this script, you need to set up the IoT sensors and connect them to your platform of choice (e.g., AWS IoT, Google Cloud IoT, etc.). Ensure you have the necessary credentials and access to the platform.

Program:

```
import paho.mqtt.client as mqtt

import json

import random

import time


# Replace with your IoT platform details

MQTT_BROKER = "your_broker_address"

MQTT_PORT = 1883

MQTT_TOPIC = "your_topic"

MQTT_USERNAME = "your_username"

MQTT_PASSWORD = "your_password"


def on_connect(client, userdata, flags, rc):

    print(f"Connected with result code {rc}")

    client.subscribe(MQTT_TOPIC)


def on_publish(client, userdata, mid):

    print("Data published")


def simulate_data():

    location_data = {

        "latitude": random.uniform(0, 90),
```

```
        "longitude": random.uniform(0, 180)
    }
    ridership_data = {
        "passenger_count": random.randint(1, 100)
    }
    return location_data, ridership_data

def main():
    client = mqtt.Client()
    client.on_connect = on_connect
    client.on_publish = on_publish
    client.username_pw_set(MQTT_USERNAME, MQTT_PASSWORD)

    try:
        client.connect(MQTT_BROKER, MQTT_PORT, 60)
    except ConnectionRefusedError:
        print("Connection to the MQTT broker failed. Check your broker settings.")
        return

    while True:
        location_data, ridership_data = simulate_data()
        message = json.dumps({"location": location_data, "ridership": ridership_data})
        client.publish(MQTT_TOPIC, message)
        print(f"Data sent: {message}")
        time.sleep(5)  # Simulate data transmission every 5 seconds

if __name__ == "__main__":
    main()
```