**Phase 4: Development Part 2**

# **Performing,different activities and model training in (IOT) traffic management system**

Working of iot traffic management system:

IoT sensors collect data on the number of vehicles, congestion, weather conditions, and more. All this data is processed and displayed on a centralised control panel. Traffic organisations and municipalities can monitor traffic conditions on the interactive control panel.

Program:

import time

import random

from datetime import datetime

import pandas as pd

from sklearn.ensemble import RandomForestRegressor

# Simulate real-time traffic data

def generate\_traffic\_data():

timestamp = datetime.now()

traffic\_count = random.randint(50, 200)

return timestamp, traffic\_count

# Sample historical traffic data

historical\_data = {

'timestamp': pd.date\_range(start='2023-11-05 00:00:00', end='2023-11-05 07:59:59', freq='15min'),

'traffic\_count': [random.randint(50, 200) for \_ in range(32)],

}

historical\_df = pd.DataFrame(historical\_data)

# Train a machine learning model for traffic prediction

def train\_traffic\_model(df):

X = df[['hour', 'day\_of\_week', 'is\_weekend']]

y = df['traffic\_count']

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

model.fit(X, y)

return model

# Function to predict traffic using the trained model

def predict\_traffic(model, current\_time):

hour = current\_time.hour

day\_of\_week = current\_time.dayofweek

is\_weekend = 1 if current\_time.weekday() >= 5 else 0

input\_data = pd.DataFrame({'hour': [hour], 'day\_of\_week': [day\_of\_week], 'is\_weekend': [is\_weekend]})

predicted\_traffic = model.predict(input\_data)

return int(predicted\_traffic[0])

# Main loop for real-time traffic management

def main():

historical\_df = perform\_feature\_engineering(historical\_df)

traffic\_model = train\_traffic\_model(historical\_df)

while True:

current\_time, current\_traffic\_count = generate\_traffic\_data()

# Feature engineering for real-time data

current\_data = pd.DataFrame({'timestamp': [current\_time], 'traffic\_count': [current\_traffic\_count]})

current\_data = perform\_feature\_engineering(current\_data)

# Predict traffic using the trained model

predicted\_traffic = predict\_traffic(traffic\_model, current\_time)

# Perform traffic management actions based on predicted traffic

if current\_traffic\_count > 1.2 \* predicted\_traffic:

print(f"High traffic detected at {current\_time}. Implementing traffic management strategies.")

# Implement traffic management actions here, such as adjusting traffic signals, rerouting, etc.

time.sleep(300) # Simulate data being received every 5 minutes

if \_\_name\_\_ == "\_\_main\_\_":

main()

**different activites of  iot TMS:**

* [1. Smart traffic signals](https://www.hologram.io/blog/7-ways-iot-can-improve-traffic-management/#Smart%20traffic%20signals)
* [2. Emergency assistance via IoT technology](https://www.hologram.io/blog/7-ways-iot-can-improve-traffic-management/#Emergency%20assistance%20via%20IoT%20technology)
* [3. Optimized commutes with apps such as Waze](https://www.hologram.io/blog/7-ways-iot-can-improve-traffic-management/#Optimized%20commutes%20with%20apps%20such%20as%20Waze)
* [4. Smart parking technology](https://www.hologram.io/blog/7-ways-iot-can-improve-traffic-management/#Smart%20parking%20technology)
* [5. Safer truck driving and fleet management](https://www.hologram.io/blog/7-ways-iot-can-improve-traffic-management/#Safer%20truck%20driving%20and%20fleet%20management)
* [6. Predictive vehicle maintenance](https://www.hologram.io/blog/7-ways-iot-can-improve-traffic-management/#Predictive%20vehicle%20maintenance)
* [7. Enhanced tolls and ticketing](https://www.hologram.io/blog/7-ways-iot-can-improve-traffic-management/#Enhanced%20tolls%20and%20ticketing)

### 1. Smart traffic signals

Smart traffic signals might look like your typical stoplight, but they use an array of sensors to monitor traffic in real-time. Often, the goal is to reduce the amount of time cars spend idle. Using IoT technology, the various signals communicate with each other and adapt to changing traffic conditions in real-time. Not only does this mean less time in traffic jams, but it cuts carbon emissions that cars release into the atmosphere. Carnegie Mellon University is part of a pilot program in Pittsburgh that has been testing [this type of technology](https://www.cmu.edu/homepage/computing/2012/fall/smart-traffic-signals.shtml), and the initial data is very promising. The pilot run showed 40% reductions in vehicle wait time, 26% in travel time, and 21% in projected vehicle emissions.

### 2. Emergency assistance via IoT technology

Among people aged 1-54 in the United States, road crashes are the leading [cause of death](https://www.asirt.org/safe-travel/road-safety-facts/#:~:text=More%20than%2038%2C000%20people%20die,for%20people%20aged%201%2D54.). Getting into cars and driving to work or school is, statistically, a dangerous activity that many people have to engage in every day. IoT technology could make it a little safer. Potentially, this could look like [real-time accident detection and notification](http://ijece.iaescore.com/index.php/IJECE/article/view/16079), reducing the critical time an injured person sits unattended. In New Orleans, [the city has deployed IoT solutions](https://devblogs.microsoft.com/azuregov/how-government-is-using-iot-to-provide-better-services-to-citizens/) across different emergency response personnel, such as the fire department, police officers, and EMTs. Using IoT, 911 dispatchers can streamline communications to these responders, allowing them to harness data and make quicker, more informed decisions.

### 3. Optimized commutes with apps such as Waze

Every driver on the road with Waze, or a similar app, essentially acts as an IoT sensor. [Waze collects and analyzes data](https://support.google.com/waze/answer/6078702?hl=en#:~:text=Waze%20collects%20data%20for%20every,to%20suggest%20the%20optimal%20route.) for every road someone drives on. With the help of a powerful algorithm, it’s able to make suggestions, determine optimal routes, give advance notice of accidents or traffic jams, and even suggest the best time to leave. [In a pilot test in Europe](https://www.enterpriseitnews.com.my/waze-iot-enabler-and-social-engineer/), Waze reduced driving time by 19%. Like smart traffic lights, less time on the road means less pollution, and it’s generally more convenient for everyone.

### 4. Smart parking technology

One of the more frustrating aspects of city driving is finding a place to park. It seems there are never any on-street parking spaces available. The second one driver leaves, the next pulls in. And lack of available parking isn’t just an issue that comes up from time to time; drivers spend about [17 hours a year](https://www.usatoday.com/story/money/2017/07/12/parking-pain-causes-financial-and-personal-strain/467637001/) driving in circles, just looking for a place to cut the engine. Smart parking technology is already tackling this problem with [apps and smart meters](https://www.nytimes.com/2017/11/30/business/car-parking-apps.html) that can alert drivers when parking spots are available, allow them to reserve the spot, give directions to the spot, and give the driver the option to pay for the space directly from their phone.

### 5. Safer truck driving and fleet management

From the trucks that deliver packages to the trucks local governments use — like fire trucks, or school buses — fleet management is important for any entity that requires vehicles to operate. These automobiles must get from one place to the next safely, efficiently, and in the most cost-effective way possible. IoT fleet management solutions can do all of that. IoT sensors can monitor important safety elements on the vehicles themselves, like tire pressure, and alert drivers to any issues. IoT devices can also follow the cars in real-time, redirecting drivers to the most efficient route or allowing the owners to track the vehicle from a computer.

6. Predictive vehicle maintenance

Imagine driving around without a fuel gauge on your car. Instead of knowing when it was time for a fill-up, you just had to take a guess and hope for the best. Sounds pretty inefficient, not to mention dangerous, right?

In many ways, that’s how a lot of other parts in our cars work. Many times, you don’t know there’s an issue with a car until something has actually happened. Although more technologically advanced, predictive vehicle maintenance in IoT works a lot like a gas gauge. Sensors monitor various aspects of a vehicle — like oil or fuel consumption, temperature, light, or vibrations — and inform owners when there is an issue. This way, the driver can get it fixed before the situation worsens, and they also won’t spend unnecessary money or time checking a vehicle that doesn’t actually need attention.

### 7. Enhanced tolls and ticketing

While tickets and tolls may not be drivers’ favorite parts about being on the road, they’re a common element of most major cities. While IoT can’t make a ticket or toll go away, it can make it a little easier for the driver to pay and be on their way. With a modern car, IoT devices and sensors can automatically pay these bills and debit the owner’s bank account.

This technology already exists to some extent; products like [E-Z pass](https://www.e-zpassny.com/en/home/index.shtml) let drivers go through tolls and pay automatically without stopping. However, government bodies are already thinking up ways to take this a step further. [A pilot program in California](https://dot.ca.gov/programs/road-charge)has been studying how IoT could be used to replace the gas tax the state currently uses to fund road repairs and construction. In this system, drivers would owe money based on how much they drive. IoT sensors would track this data and automatically charge the user based on this information.

Model training for TMS

To create a model for training an IoT traffic management system, we'll use a regression model to predict traffic based on various features. In this example, I'll use Python and the scikit-learn library to create a simple Random Forest Regressor. Keep in mind that in a real-world scenario, you might need more sophisticated models and additional features.

**Program**

import pandas as pd

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

from sklearn.preprocessing import OneHotEncoder

# Sample historical traffic data

historical\_data = {

'timestamp': pd.date\_range(start='2023-11-05 00:00:00', end='2023-11-05 07:59:59', freq='15min'),

'traffic\_count': [random.randint(50, 200) for \_ in range(32)],

'hour': [t.hour for t in pd.date\_range(start='2023-11-05 00:00:00', end='2023-11-05 07:59:59', freq='15min')],

'day\_of\_week': [t.dayofweek for t in pd.date\_range(start='2023-11-05 00:00:00', end='2023-11-05 07:59:59', freq='15min')],

'is\_weekend': [1 if t.weekday() >= 5 else 0 for t in pd.date\_range(start='2023-11-05 00:00:00', end='2023-11-05 07:59:59', freq='15min')],

}

historical\_df = pd.DataFrame(historical\_data)

# Feature engineering function

def perform\_feature\_engineering(df):

encoder = OneHotEncoder(sparse=False, drop='first')

encoded\_features = encoder.fit\_transform(df[['hour', 'day\_of\_week']])

encoded\_df = pd.DataFrame(encoded\_features, columns=encoder.get\_feature\_names\_out(['hour', 'day\_of\_week']))

df = pd.concat([df, encoded\_df], axis=1)

df.drop(['hour', 'day\_of\_week'], axis=1, inplace=True)

return df

# Feature engineering for historical data

historical\_df = perform\_feature\_engineering(historical\_df)

# Split the data into training and testing sets

X = historical\_df.drop(['timestamp', 'traffic\_count'], axis=1)

y = historical\_df['traffic\_count']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train a Random Forest Regressor model

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

# Evaluate the model on the test set

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

print(f'Mean Squared Error on the test set: {mse}')