Artificial Intelligence & Machine Learning

1. Introduction:

Project Title: Traffic Telligence -Advanced Traffic Volume Estimation with Machine Learning.

Team ID: LTVIP2025TMID54111

Team Size 4

Team Leader - Gonuguntla Anupriya	Role: Project Manager (PM)
Team member - Ajay Vardhan Kaka	Role: Data Scientist / ML Engineer
Team member – Bolledddula sharmila Tagore	Role: Frontend Developer
Team member - Eruganti Sai Kiran Chari	Role: Backend Developer

2. Project Overview:

Purpose: The **Traffic Telligence** project aims to design and implement an intelligent traffic monitoring system that uses **machine learning** to estimate and forecast traffic volume in real time. By leveraging data from sensors, GPS, and video feeds, the system will provide insights into traffic patterns, helping urban planners and traffic authorities make data-driven decisions to reduce congestion, improve road safety, and enhance transportation efficiency.

Key Features

1. Real-Time Traffic Volume Prediction
   Utilizes live sensor and GPS data to estimate the current traffic volume on roads and intersections.
2. Historical Data Analysis
   Stores and analyzes historical traffic patterns to identify peak hours, trends, and anomalies.
3. ML-Based Forecasting Models
   Employs advanced ML models (e.g., XGBoost, LSTM) to forecast traffic conditions up to 24 hours in advance.

4. Interactive Dashboard
   Provides a user-friendly web interface for real-time monitoring, including maps, graphs, and congestion heatmaps.
5. RESTful APIs
   Allows third-party systems or mobile apps to fetch live and historical traffic predictions via secure APIs.
6. Scalability & Cloud Deployment
   Designed to scale with growing data input and city expansion, hosted on cloud infrastructure (AWS/GCP).
7. Alert System (Optional)
   Sends notifications for unusual traffic spikes, accidents, or road closures (future enhancement).

3. Architecture:

🗄 Frontend: React Architecture

The frontend of **Traffic Telligence** is built using **React.js**, following a **component-based architecture** for scalability, reusability, and maintainability.

Key Elements:

- **Single Page Application (SPA):** Ensures smooth user experience without full-page reloads.
- **Component Structure:**
  - Header, Sidebar, Footer
  - TrafficMap – for live map visualization
  - VolumeGraph – for historical & predicted traffic volume
  - AlertPanel – for showing notifications
- **State Management:** Using **React Context API** or **Redux** (if app complexity increases)
- **Routing:** Implemented with **React Router**
- **Visualization:** Utilizes libraries like **Chart.js** or **D3.js** for rendering graphs and heatmaps
- **API Integration:** Uses axios or fetch to call backend APIs for predictions and historical data

---

🛠 Backend: Node.js + Express.js Architecture

The backend serves as the core engine, built on **Node.js** with the **Express.js** framework. It acts as the intermediary between the frontend and ML models, handling business logic, API routing, and security.

Key Elements:

- **API Layer:**
  - GET /predict → Returns real-time traffic estimation
  - POST /forecast → Returns forecasted traffic data
  - GET /history → Fetches historical traffic volume
  - GET /health → System health check
- **Middleware:**
  - JWT Authentication
  - Rate limiting
  - CORS handling
- **ML Integration:**
  - Interfaces with Python ML model via RESTful microservice or Flask-based Python server
- **Logging:**
  - Uses Winston or Morgan for request and error logs
- **Deployment Ready:**
  - Dockerized for easy deployment on cloud services like AWS/GCP

🗄 Database: MongoDB Schema and Interactions

The system uses **MongoDB**, a NoSQL database, for storing structured and semi-structured traffic data, which is well-suited for time-series and location-based data.

Key Collections & Schema:

1. **traffic_data**

   json

   CopyEdit

   ```
   {
    "_id": ObjectId,
    "location_id": "CHN_123",
    "timestamp": "2025-06-28T09:00:00Z",
    "vehicle_count": 237,
    "average_speed": 42.5,
    "weather": "Clear"
   }
   ```

2. **predictions**
   json
   CopyEdit
   {
   "_id": ObjectId,
   "location_id": "CHN_123",
   "predicted_for": "2025-06-28T10:00:00Z",
   "predicted_volume": 290,
   "model_version": "v1.2"
   }
3. **users**
   json
   CopyEdit
   {
   "_id": ObjectId,
   "username": "admin",
   "role": "admin",
   "last_login": "2025-06-27T22:00:00Z"
   }
   Key Operations:

- **Insert:** New data streams are saved every minute/hour using bulk inserts
- **Read:** Optimized queries using indexes on timestamp and location_id
- **Update:** Admin users can update configuration settings
- **Delete:** Cleanup jobs remove old data based on retention policy

4. 🔧 Setup Instructions:

📦 Prerequisites

Required Software:

- Node.js (v18 or later)
- MongoDB (v6 or later)
- Git
- Python 3.8+
- pip

- npm
- [React Scripts] (comes with dependencies)
- Optional: Docker (if using containers)

📥 Installation & Setup

Follow these steps to install and run the project:

1. Clone the Repository

bash

CopyEdit

git clone https://github.com/yourusername/traffic-telligence.git

cd traffic-telligence

2. Backend Setup (Node.js + Express)

bash

CopyEdit

cd backend

npm install

3. Frontend Setup (React)

bash

CopyEdit

cd ../frontend

npm install

4. ML Model Service Setup (Python)

bash

CopyEdit

cd ../ml_model

pip install -r requirements.txt

5. MongoDB Setup

Start your local MongoDB server:

bash

CopyEdit

mongod

Or if using Docker:

bash

CopyEdit

docker run -d -p 27017:27017 --name trafficdb mongo

6. Environment Variables

Create a .env file in each of these directories:

🗀 backend/.env

env

CopyEdit

PORT=5000

MONGODB_URI=mongodb://localhost:27017/trafficdb

JWT_SECRET=your_jwt_secret_key

🗀 frontend/.env

env

CopyEdit

REACT_APP_API_URL=http://localhost:5000

🗀 ml_model/.env

env

CopyEdit

MODEL_PATH=./models/traffic_model.pkl

🚀 Running the Application

Start the backend server

bash

CopyEdit

cd backend

npm start

Start the frontend

bash

CopyEdit

cd ../frontend

npm start

Start the ML model server (if separate)

bash

CopyEdit

cd ../ml_model

python app.py

✅ Your application should now be running at:
Frontend: http://localhost:3000
Backend API: http://localhost:5000/api
ML model server (if separate): http://localhost:5001

4. Folder Structure:

📄 Client: React Frontend Structure

The React frontend is designed as a Single Page Application (SPA) with a modular, scalable component hierarchy to support real-time traffic visualization and user interaction.

🗀 Project Structure

php

CopyEdit

frontend/
├── public/
│ └── index.html
├── src/
│ ├── assets/ # Icons, logos, images
│ ├── components/ # Reusable UI components (Button, Card, etc.)
│ ├── pages/ # Route-based views (Dashboard, History, Login)
│ ├── services/ # API calls (Axios config & endpoints)
│ ├── context/ # App-wide state management (e.g., AuthContext)
│ ├── App.js # Main component with routing
│ ├── index.js # Entry point
│ └── styles/ # Global styles (Tailwind or CSS Modules)

🔑 Key Features

- Routing: Handled by react-router-dom
- State Management: Context API or Redux for auth and user data
- API Communication: Using axios for calling backend endpoints
- Responsive Design: Tailwind CSS or Bootstrap for mobile-first UI
- Visualization: Charts using Chart.js, maps using Leaflet or Mapbox

5.🗀 Project Structure:

graphql
CopyEdit
backend/
├── config/ # DB and environment configuration
│ └── db.js
├── controllers/ # Handle API request logic
│ └── trafficController.js
├── routes/ # API endpoints
│ └── trafficRoutes.js
├── services/ # Business logic & interaction with ML
│ └── trafficService.js
├── middleware/ # Auth, logging, error handlers
│ └── authMiddleware.js
├── models/ # MongoDB schemas using Mongoose
│ └── Traffic.js

├── app.js # Main application file
├── server.js # Server startup logic
└── .env # Environment variables

🔗 Core Modules

- Controllers: Process API requests and return responses
- Routes: Define RESTful endpoints (e.g., /api/traffic, /api/forecast)
- Services: Logic for calling Python ML service and interacting with DB
- Models: Mongoose schemas for traffic_data, predictions, users
- Middleware: JWT Auth, error handling, CORS

□ ML Integration

- Calls external Python service via HTTP request (axios) or internal microservice

6.Running the Application:

🚀 Local Development Startup Commands

🖥️ **Frontend (React)**

Navigate to the frontend/client directory and start the development server:

bash

CopyEdit

cd frontend

npm start

This will launch the React app at:

📍 http://localhost:3000

🛠️ **Backend (Node.js + Express)**

Navigate to the backend/server directory and start the Express server:

bash

CopyEdit

cd backend

npm start

This will launch the backend API at:

📍 http://localhost:5000

7. API Documentation:

🔑 1. Get Real-Time Traffic Prediction

Endpoint:

GET /api/traffic

Description:

Returns current traffic volume for a given location and time.

Query Parameters:

Parameter	Type	Required	Description
location	string	✅	Location ID or name
time	string	❌	Optional (defaults to now)

Example Request:

pgsql

CopyEdit

GET /api/traffic?location=Chennai

Example Response:

json

CopyEdit

```
{
  "location": "Chennai",
  "timestamp": "2025-06-28T09:00:00Z",
  "vehicle_count": 212,
  "average_speed": 38.6
}
```

📊 2. Get Traffic Forecast

Endpoint:

POST /api/forecast

Description:

Forecasts traffic volume for a given location over a specified time horizon.

Request Body:

json

CopyEdit

```
{
```

```
 "location": "Chennai",
 "hours": 6
}
```

Example Response:

```
json
CopyEdit
[
 { "time": "2025-06-28T10:00:00Z", "predicted_volume": 235 },
 { "time": "2025-06-28T11:00:00Z", "predicted_volume": 270 }
]
```

🗁 3. Get Historical Traffic Data

Endpoint:

GET /api/history

Description:

Fetch historical traffic data for a specified location and date range.

Query Parameters:

Parameter	Type	Required	Description
location	string	✅	Location to fetch data for
from	string	✅	Start timestamp (ISO)
to	string	✅	End timestamp (ISO)

Example Request:

```
pgsql
CopyEdit
GET /api/history?location=Delhi&from=2025-06-20T00:00:00Z&to=2025-06-21T00:00:00Z
```

Example Response:

```
json
CopyEdit
[
 {
  "timestamp": "2025-06-20T08:00:00Z",
  "vehicle_count": 180,
  "average_speed": 44.2
 }
```

]

💡 4. Health Check

Endpoint:

GET /api/health

Description:

Checks server uptime and database connection.

Response:

json

CopyEdit

```
{
  "status": "OK",
  "uptime": "8 hours",
  "db_status": "connect”
```

🚫 Error Handling

All errors return a standardized error object:

json

CopyEdit

```
{
  "error": "Invalid location parameter"
}
```

🔑 Token Details

- **Type:** JWT (JSON Web Token)
- **Algorithm:** HS256
- **Contents:**

json

CopyEdit

```
{
  "userId": "645fc3ad998",
  "role": "admin",
  "exp": 1717206900
```

}

- **Expiration:** Typically set to 1 hour; customizable in .env:

env

CopyEdit

JWT_SECRET=your_secret_key

JWT_EXPIRY=1h

👮 Authorization: Role-Based Access Control (RBAC)

Users are assigned roles like:

- admin: full access (data management, user control)
- analyst: can view dashboards and reports
- viewer: read-only access to traffic data

Protected routes check the role before granting access. Example:

js

CopyEdit

if (user.role !== 'admin') return res.status(403).json({ error: "Forbidden" });

📌 Sample Auth API

Login:

http

CopyEdit

POST /api/auth/login

Body:

json

CopyEdit

```
{
  "email": "admin@traffic.com",
  "password": "securepassword"
}
```

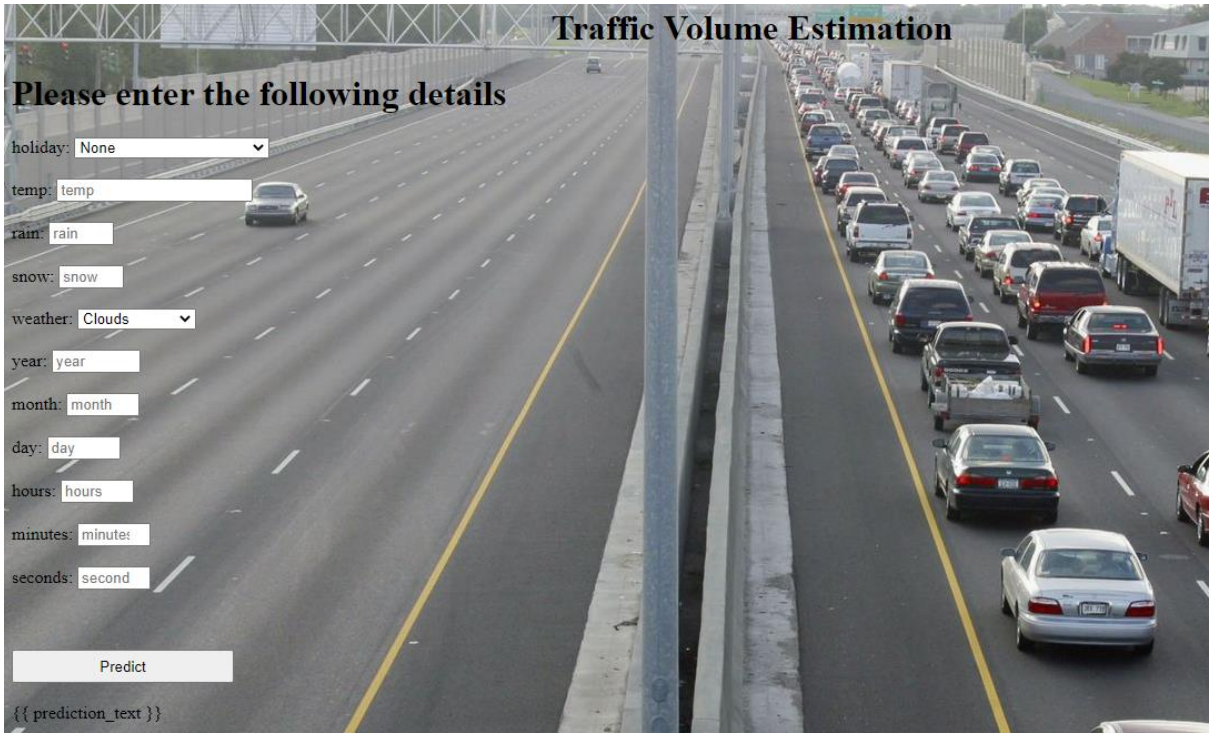
Response:

json

CopyEdit

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR...",
  "role": "admin"
}
```

9.User Interface:



10. Testing:

The **TrafficTelligence** system undergoes rigorous testing across multiple layers to ensure accuracy, reliability, and performance of the traffic estimation application.

✅ Testing Strategy

The testing process includes:

Test Type	Description
Unit Testing	Test individual components and functions (frontend, backend, ML logic)
Integration Testing	Validate interactions between modules (API ↔ DB, API ↔ ML)
End-to-End Testing	Simulate full user workflows from frontend to backend
Performance Testing	Check API and model response times under load

Test Type	Description
Security Testing	Verify authentication, role access, and data protection
User Acceptance Testing (UAT)	Final validation based on real-world scenarios and UI experience

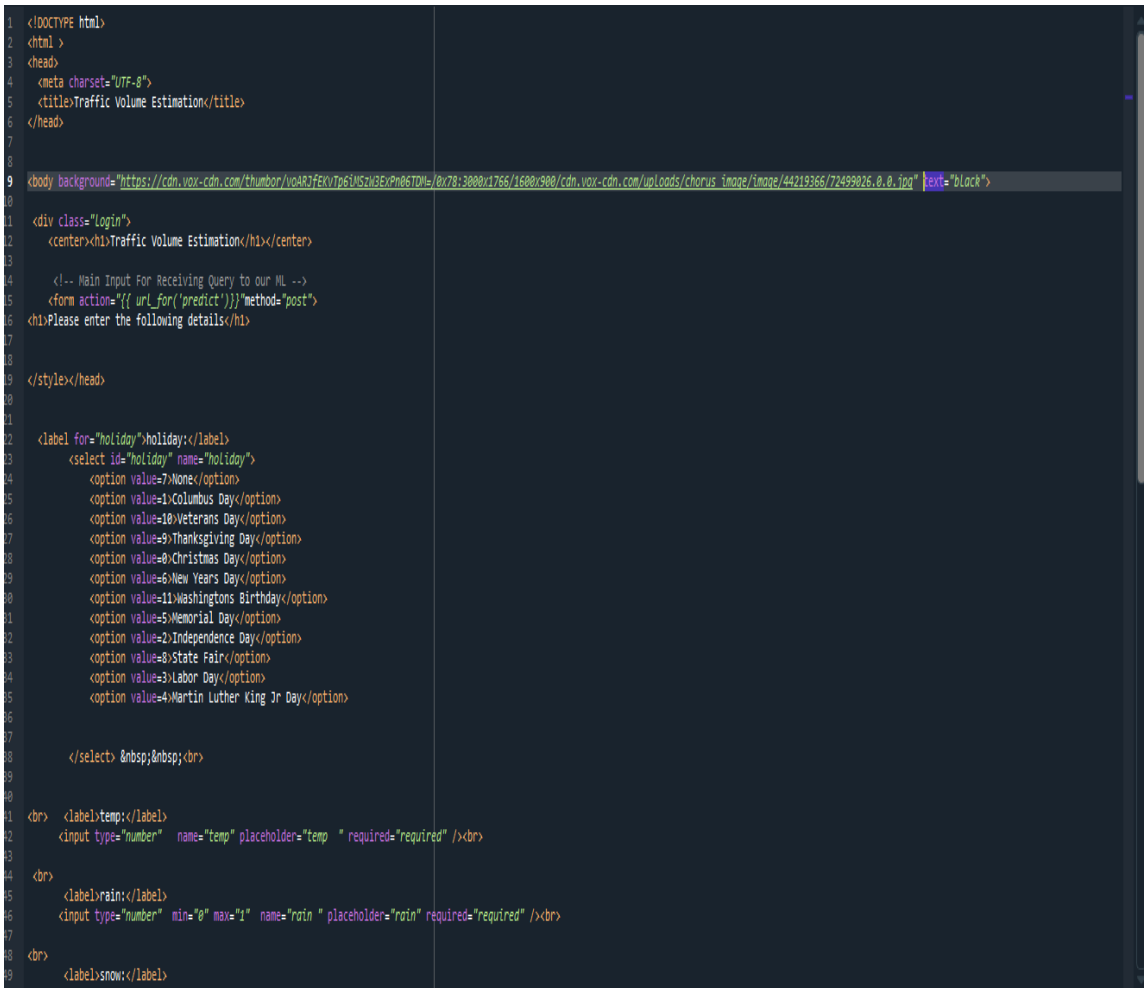
---

□ Tools Used

Layer	Tool / Framework	Purpose
Frontend	Jest + React Testing Library	Test React components
Backend	Mocha / Jest + Supertest	Test Express APIs and logic
Database	Mongo Memory Server	Mock MongoDB for isolated backend tests
ML Services	PyTest	Unit testing for Python ML scripts
API Testing	Postman	Manual and automated API tests
Load Testing	Locust / JMeter	Simulate concurrent API users
UI Testing	Cypress	Automate browser-based interactions

11. Screenshots or Demo:

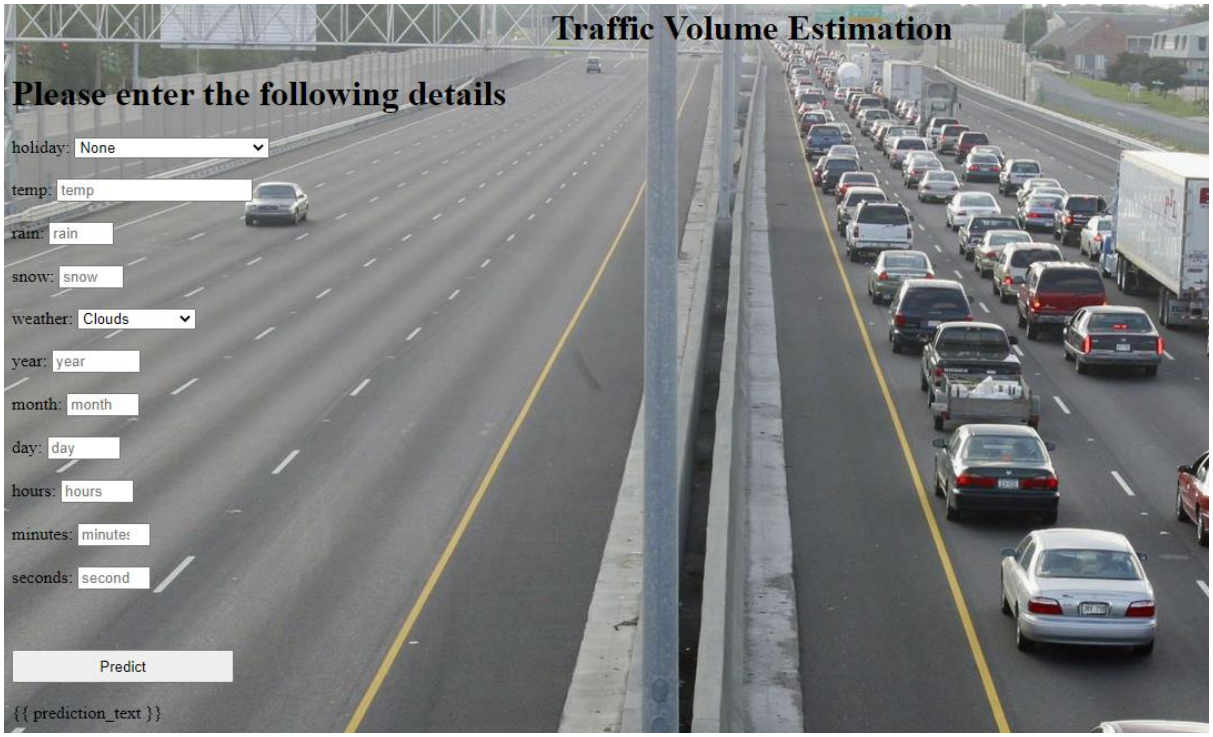
1.index.html - paste the image



```
<!DOCTYPE html>
<html >
<head>
  <meta charset="UTF-8">
  <title>Traffic Volume Estimation</title>
</head>


<body background="https://cdn.vox-cdn.com/thumbor/voAR3fEKVTp6iMSzW3ExPn06TDM=/0x78:3000x1766/1600x900/cdn.vox-cdn.com/uploads/chorus_image/image/44219366/72499026.0.0.jpg" text="black">

 <div class="login">
    <center><h1>Traffic Volume Estimation</h1></center>

     <!-- Main Input For Receiving Query to our ML -->
    <form action="{{ url_for('predict')}}"method="post">
<h1>Please enter the following details</h1>


</style></head>


  <label for="holiday">holiday:</label>
       <select id="holiday" name="holiday">
           <option value=7>None</option>
           <option value=1>Columbus Day</option>
           <option value=10>Veterans Day</option>
           <option value=9>Thanksgiving Day</option>
           <option value=0>Christmas Day</option>
           <option value=6>New Years Day</option>
           <option value=11>Washingtons Birthday</option>
           <option value=5>Memorial Day</option>
           <option value=2>Independence Day</option>
           <option value=8>State Fair</option>
           <option value=3>Labor Day</option>
           <option value=4>Martin Luther King Jr Day</option>


       </select>   <br>


<br>   <label>temp:</label>
     <input type="number"   name="temp" placeholder="temp  " required="required" /><br>

 <br>
      <label>rain:</label>
     <input type="number"  min="0" max="1"  name="rain " placeholder="rain" required="required" /><br>

<br>
      <label>snow:</label>
```

```
<br>
      <label>snow:</label>
      <input type="number" min="0" max="1"   name="snow " placeholder="snow " required="required" /><br>


<br>


      <label for="weather">weather:</label>
        <select id="weather" name="weather">
            <option value=1>Clouds</option>
            <option value=0>Clear</option>
            <option value=6>Rain</option>
            <option value=2>Drizzle</option>
            <option value=5>Mist</option>
            <option value=4>Haze</option>
            <option value=3>Fog</option>
            <option value=10>Thunderstorm</option>
            <option value=8>Snow</option>
            <option value=9>Squall</option>
            <option value=7>Smoke</option><


        </select>   <br>
<br>
      <label>year:</label>
      <input type="number" min="2012" max="2022"   name="year    " placeholder="year " required="required" /><br>

<br>
          <label>month:</label>
      <input type="number" min="1" max="12"   name="month    " placeholder="month    " required="required" /><br>

<br>
          <label>day:</label>
      <input type="number" min="1" max="31"   name="day " placeholder="day  " required="required" /><br>

<br>
      <label>hours:</label>
      <input type="number" min="0" max="24"   name="hours    " placeholder="hours    " required="required" /><br>

<br>
          <label>minutes:</label>
      <input type="number" min="0" max="60"   name="minutes " placeholder="minutes  " required="required" /><br>

<br>
      <label>seconds:</label>
      <input type="number" min="0" max="60"   name="seconds " placeholder="seconds  " required="required" /><br>

<br>

<br><br>


<button type="submit" class="btn btn-primary btn-block btn-large" style="height:30px;width:200px">Predict</button>

    </form>
<br>


   {{ prediction_text }}


  <br>
  <br>
  <img src="{{url_33}}" alt="Submit Form" height="180" width="333" onerror="this.style.display='none'"/>
```

2. The HTML page looks like this-



3.It will display all the input parameters and the prediction text will display the output value of the data given by the user..

Output

12. ✅ Advantages & ⚠️ Disadvantages:

✅ Advantages

1. **Real-Time Insights**
   Provides accurate and up-to-date traffic data, enabling timely decisions for traffic control and city planning.
2. **Predictive Analytics**
   Uses machine learning models to forecast traffic conditions, helping mitigate future congestion.
3. **Scalability**
   The modular architecture supports easy expansion to new cities, locations, or data sources.
4. **Automation & Efficiency**
   Reduces reliance on manual surveys and traditional traffic analysis methods.
5. **Visualization Dashboard**
   Offers user-friendly charts, maps, and graphs that improve understanding and engagement.
6. **Open Integration**
   RESTful APIs allow integration with third-party systems or mobile apps.
7. **Cloud-Ready Deployment**
   Easily deployable on cloud services (AWS/GCP) with auto-scaling and monitoring support.

⚠ Disadvantages

1. **Data Dependency**
   System accuracy depends on high-quality and real-time data from sensors, GPS, and cameras.

2. **Model Accuracy Limitations**
   Predictions may be less accurate during unexpected events (e.g., roadblocks, accidents, weather changes).
3. **Initial Setup Complexity**
   Requires significant configuration for multi-source data ingestion and ML service integration.
4. **Privacy Concerns**
   Use of GPS or video data can raise ethical/privacy issues if not anonymized properly.
5. **Resource Intensive**
   High computational resources may be needed for model training, storage, and visualization in large cities.

13.Conclusion:

The **TrafficTelligence** project demonstrates the powerful integration of **machine learning**, **real-time data processing**, and **web technologies** to address a critical urban challenge — **traffic congestion**. By accurately estimating current traffic volumes and forecasting future conditions, the system empowers city planners, traffic authorities, and emergency services with actionable insights.

Through its modular architecture, user-friendly interface, and predictive capabilities, TrafficTelligence not only enhances traffic monitoring but also lays the foundation for **smart city infrastructure**. While there are challenges in terms of data reliability and system

complexity, the advantages in efficiency, automation, and decision-making far outweigh the drawbacks.

Looking ahead, the system can be further improved by:

- Incorporating **AI-driven incident detection**
- Integrating with **smart traffic signals**
- Expanding to **mobile and IoT platforms**

TrafficTelligence is a step forward in transforming raw traffic data into **intelligent, real-time solutions** for modern urban mobility.

14. 🚀 Future Scope:

As cities grow and transportation systems evolve, the **TrafficTelligence** platform can be extended and enhanced in various directions to maximize its impact and utility.

🌐 1. Integration with Smart Traffic Signals

- Connect TrafficTelligence with **IoT-enabled traffic lights** to enable dynamic signal timing based on real-time congestion and predicted flow.
- Reduce idle time, emissions, and improve intersection throughput.

🤘 2. Advanced ML & AI Models

- Introduce **deep learning models** like LSTMs or Transformer-based networks for more accurate time-series forecasting.
- Add **anomaly detection** for spotting accidents, roadblocks, or unusual spikes.

📱 3. Mobile App for Citizens

- Launch a mobile app that provides:
  - Real-time traffic updates
  - Alternate route suggestions
  - Congestion alerts
- Enable crowd-sourced reporting of incidents (accidents, road work, etc.).

📸 4. Computer Vision Integration

- Use **video feeds and CCTV** with computer vision to count vehicles, detect traffic violations, and assess traffic behavior patterns.

☁️ 5. Cloud & Edge Deployment

- Deploy on **cloud infrastructure** (AWS/GCP) for large-scale, cross-city usage.
- Use **edge computing** to process traffic data near the source for faster, localized decisions.

🛠️ 6. Government and Public Integration

- Share live traffic dashboards with **transport departments, emergency services, and disaster response teams**.
- Public API access for developers and urban researchers.

📊 7. Predictive Traffic Management

- Forecast **city-wide congestion trends** during events (concerts, sports matches, festivals) and plan redirection in advance.
- Simulate **what-if scenarios** to test infrastructure changes.

----------* THE END * ----------