TrafficTelligence: Advanced Traffic Volume Estimation with Machine Learning

Project Documentation format

1. Introduction

• Project Title: [TrafficTelligence]

• Team Members:

Team members	role
Swetha Rani	Testing
Mounika	Model testing, Backend
Thasyfa	Model Training,Backend
Keerthi Priya	Analysizing
Vaishnavi	Fronend development

2. Project Overview

✓ **Purpose:** The main purpose of **TrafficTelligence: Advanced Traffic Volume Estimation** is to build a smart system that can **predict traffic volume** based on real-world inputs like weather conditions and time data. By using **machine learning models** and integrating them into a **web application**, the system allows users to estimate traffic congestion in advance — helping in **better planning, navigation, and traffic management**.

✓ **Features:**

- **Real-time traffic volume prediction** using weather inputs
- **Machine Learning model integration** (Random Forest)
- **User-friendly web interface** built with HTML & CSS
- **Flask backend** for processing inputs and returning predictions
- **Complete ML pipeline**: data preprocessing, training, evaluation, deployment
- **Organized project structure** for easy development and testing

3. Architecture

• **Frontend:** The frontend of the TrafficTelligence application is designed using **React**, a powerful JavaScript library for building interactive user interfaces. It follows a **component-based architecture**, ensuring modularity and reusability.

◈ 1. Component Structure

- **App.js** – Main container that renders the entire UI
- **InputForm.js** – Handles user inputs for temperature, rain, snow, and cloud data
- **ResultCard.js** – Displays the predicted traffic volume
- **Header.js / Footer.js** – (Optional) Reusable layout components

---

◈ 2. State Management

- Utilizes **React Hooks** like useState and useEffect
- Stores and updates:
  - User input values
  - Server response (prediction)
  - Loading/error states

◈ 3. API Communication

- Uses **Axios** to make a **POST request** to the Flask backend (http://127.0.0.1:5000/predict)
- Sends input as JSON and receives the predicted traffic volume in the response

---

◈ 4. Styling

- Styling is handled using:
  - Plain **CSS**
  - OR **Tailwind CSS** for utility-first, responsive design
- UI is kept minimal, clean, and mobile-friendly

🔷 5. Conditional Rendering

- React dynamically updates the DOM to show prediction only after the API response is received

• **Backend:** The backend of the application is built using **Node.js** with the **Express.js** framework to handle routing, API logic, and communication with the machine learning model.

🔷 1. Server Setup (Express.js)

- **Express.js** is used to create a lightweight web server
- Listens on a specific port (e.g., PORT 5000)
- Handles HTTP requests (GET for UI, POST for prediction)

🔷 2. API Endpoints

- **POST /predict**: Receives weather input from the React frontend
- Sends input to the **Python ML model** (via child_process or API call)
- Responds with the predicted traffic volume

🔷 3. Communication with Python Model

- The backend uses one of these approaches:
    - **Python Shell (child_process)** to run the Python script directly
    - OR **HTTP Request** to a separate Flask API hosting the ML model
- Sends inputs → Receives prediction → Sends response back to frontend

---

🔷 4. Middleware and JSON Handling

- **Body-parser** or built-in Express JSON parser handles incoming form data
- CORS middleware allows cross-origin requests from the React frontend

---

🔷 5. File Structure Example

Go

```
backend/
 ├── server.js           // Main Node.js server
 ├── routes/
 │   └── predict.js      // Handles /predict POST route
 ├── ml_model/
 │   └── traffic_model.py // Python script for prediction
 └── package.json        // Dependencies and scripts
```

• **Database:** For the current version of the **TrafficTelligence** project:

❌ **No database is used by default**
✅ All predictions are done **in real-time** and returned directly to the user without being stored.

💡 **Why No Database?**

The project is a **stateless application**:

- User enters values → Flask processes → Prediction is shown
- No login, no data saving, no analytics — so no database is needed at this stage

4. Setup Instructions

• **Prerequisites:** List software dependencies (e.g., pandas,numpy,seaborn,pickle,etc.,).

• **Installation:** Step-by-step guide to clone, install dependencies, and set up the environment variables.

5. Folder Structure

```
TrafficTelligence/
 ├── app.py              # Flask backend application
 ├── model_training.py       # Script to train the ML model
 ├── traffic_model.pkl       # Saved trained model (using pickle)
 ├── traffic_volume.csv      # Dataset used for training/testing
 │
 ├── templates/              # HTML files for frontend (Flask uses this)
 │   └── index.html          # User interface for input & output
 │
 ├── static/                 # CSS or image files (used by HTML)
 │   └── style.css           # Styling for the UI
 │
 ├── __pycache__/            # Auto-generated Python cache (ignore this)
 │
 └── README.md (optional)    # Description and usage instructions
```

.

6. Running the Application

• Provide commands to start the frontend and backend servers locally.

o Frontend: **Frontend (Basic HTML/CSS)**

- HTML – Structure for the input form
- CSS – Styling the user interface

o Backend:

```
TrafficTelligence/
├── app.py             # Main backend script
├── traffic_model.pkl  # Trained ML model
```

7. API Documentation

• Document all endpoints exposed by the backend.

• Include request methods, parameters, and example responses.

8. Authentication

Authentication in TrafficTelligence

⚠ **Note:** The current version of **TrafficTelligence** does **not include authentication** by default.

✅ **Optional: Adding Authentication (For Future Upgrade)**

9. User Interface

The **UI (User Interface)** of TrafficTelligence is designed to be **simple, clean, and user-friendly**, allowing users to input data and instantly get a traffic prediction.

10. Testing

• Describe the testing strategy and tools used.

11. Screenshots or Demo

• Provide screenshots or a link to a demo to showcase the application.

12. Known Issues

• Document any known bugs or issues that users or developers should be aware of.

13. Future Enhancements

• Outline potential future features or improvements that could be made to the project.