ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING PROJECT

TrafficTelligence: Advanced Traffic Volume Estimation with Machine Learning

1. Introduction

Project Title: TrafficTelligence: Advanced Traffic Volume Estimation with Machine Learning.

Team Members and Roles:

- **Konathala Pavithra (Team Lead):** Managed overall planning, oversaw dataset analysis, trained the machine learning model, integrated backend logic, and finalized deployment. Acted as the bridge between team efforts and guided the technical direction of the project.
- **Akkurthy Kavya:** Responsible for frontend design, building the HTML layout, improving UI/UX by embedding background images, and managing the visual appeal of the web interface.
- **Jeela Eesha:** Handled data preprocessing operations, including managing missing values, encoding categorical features like weather and holiday, and preparing data suitable for model input.
- **Gandla Padmasree:** Led testing efforts, managed Flask integration, coordinated model deployment, and helped with error handling and troubleshooting during execution.

2. Project Overview

Purpose:

The primary goal of this project is to help commuters, traffic authorities, and city planners make better, informed decisions about traffic by predicting traffic volume using historical data and machine learning. Rather than relying on hardware like traffic cameras or sensors, TrafficTelligence leverages data such as weather, date, time, and holiday status to estimate how congested a road might be at a given time. This predictive insight can reduce travel time, fuel usage, and traffic stress, especially in urban areas with complex road networks.

Key Features:

- A user-friendly HTML interface for traffic input
- Prediction of traffic volume using a trained Random Forest Regressor model
- Flask backend integration for server-side operations
- Use of static background images to enhance UI
- Time-efficient result delivery (~2–3 seconds)

3. Architecture

Frontend:

- Built using HTML and CSS
- The UI consists of:
  - A form (index.html) that collects weather conditions, date, time, and holiday status
  - A result page (output.html) that displays the predicted traffic volume with background styling
- Responsive design with static assets served from the /static folder

Backend:

- Powered by Python's Flask framework
- Handles HTTP POST requests, performs form data preprocessing, loads the model (model.pkl), makes predictions, and returns output.
- Lightweight and easy to deploy, with scope to integrate APIs **Database**: No live database is used. Data is loaded from a CSV file for model training
- Future versions can integrate MongoDB, Firebase, or SQL databases for logging predictions or storing user information

4. Setup Instructions

Prerequisites:

- Python 3.x
- Flask
- Pandas
- NumPy
- Scikit-learn
- Pickle module

Installation Steps:

1. Clone or download the project repository
2. Open the terminal and navigate to the project directory
3. Install the required libraries:

   pip install flask pandas numpy scikit-learn etc.
4. Ensure model.pkl and encoder.pkl are present in the root directory
5. Launch the application: python app.py
6. Open a browser and go to:

   http://127.0.0.1:5000/

5. Folder Structure

```
TrafficTelligence/
│
├── app.py              # Flask app backend
├── model.pkl           # Trained ML model
├── encoder.pkl         # Label encoder
│
├── templates/
│   ├── index.html      # Input form
│   └── output.html     # Output prediction
│
├── static/
│   ├── bg1.jpg         # Background for input page
└── bg2.jpg         # Background for result page
```

- Launch the Flask app by running:

  python app.py
- Open your browser and visit:

  http://127.0.0.1:5000
- Enter the input values and submit the form
- The backend processes the data and displays the predicted traffic volume on a new page

6.Running the Application – TrafficTelligence

To run the project locally, we used **Flask** for the backend and **HTML/CSS** for the frontend.

💻 **Steps to Run the Application Locally:**

- **Step 1:** Open your terminal or command prompt.
- **Step 2:** Navigate to the project folder where app.py is located.
- **Step 3:** Run the Flask application using the command:

python app.py

- **Step 4:** Open a web browser and go to:

http://127.0.0.1:5000/

7.API Documentation

This project uses a **Flask-based internal API** to process user inputs from a web form and return predicted traffic volume. While it is not exposed as a public REST API, the internal endpoint behaves like one during form submission.

Endpoint:

POST /predict
This route handles form data submission, processes the input, invokes the trained machine learning model, and returns the prediction rendered in an HTML page.

Request Parameters (Form Inputs)

The following input fields are sent from index.html as part of the POST request:

- **weather** (String)
  The current weather condition selected by the user.
  Example: "Clear", "Cloudy", "Mist", "Rainy"
- **temperature** (Float)
  Ambient temperature in Kelvin.
  Example: 289.52
- **rain** (Float)
  Rainfall amount in milli meters.
  Example: 0.0, 2.3
- **snow** (Float)
  Snowfall amount in milli meters.
  Example: 0.0, 1.2

- **holiday** (Integer)
  Holiday status.
  0 = Not a Holiday, 1 = Holiday
- **Date and Time Details** (Integer/String)
  These are extracted and passed as individual values:
  - day – e.g., 25
  - month – e.g., 06
  - year – e.g., 2025
  - hours – e.g., 10
  - minutes – e.g., 15
  - seconds – e.g., 00

  These values are preprocessed (e.g., encoded, converted) before being fed into the model.

  Response

  The /predict route returns a rendered HTML page (output.html) that includes:

- The predicted traffic volume
- A styled layout using a background image (bg2.jpg)
- An option to navigate back to the input form

  Example Workflow

1. User submits the input form via index.html.
2. Flask receives the data at the /predict endpoint.
3. The input is preprocessed and passed to the machine learning model (model.pkl).
4. The model predicts the traffic volume.
5. The result is shown on the output page (output.html).

8. Authentication

Currently, authentication is not implemented. In the future, the following can be added:

- User Login/Register System
- JWT Tokens or Flask-Login
- Admin Role to Monitor Usage
- User Profiles for Saving History

9. User Interface

The user interface of *TrafficTelligence* was designed to be simple, intuitive, and focused on user experience. Our goal was to allow any user — technical or non-technical — to easily enter the required details and receive a meaningful traffic volume prediction without confusion.

 The input form (index.html) is structured with clearly labeled fields, allowing users to enter data such as:

- Weather condition (e.g., clear, rainy)
- Temperature, Rain, and Snow
- Date and Time values (hours, minutes, seconds)
- Holiday status (0 for No, 1 for Yes)

 Visual aesthetics were enhanced using static background images:

- bg1.jpg is used for the input page, featuring a city or road background to symbolize live traffic data entry.
- bg2.jpg is used for the output page, providing a professional look that visually separates the prediction result screen.

 The output page (output.html) is designed to clearly display the predicted traffic volume, with styled fonts and centered layout to immediately catch the user's attention.

 We intentionally kept the interface free from JavaScript to maintain simplicity and avoid external dependencies. However, the structure is flexible and can easily be extended in the future to support:

- Form validation
- Live previews
- Dynamic input suggestions

 All styling was done using inline CSS and basic HTML elements to keep the layout lightweight and fast to load.

 The design ensures that users have a seamless flow from input to result, reflecting the practical value of the project in real-world usage scenarios such as navigation apps or smart traffic control systems.

10. Testing

Types of Testing Done:

- Manual testing with sample inputs
- Cross-browser testing (Chrome, Firefox)
- Prediction accuracy validation
- Flask route and form behavior testing

Observations:

- The model loads successfully on every run
- All inputs are processed correctly if formats are maintained
- Predictions are returned within 2–3 seconds

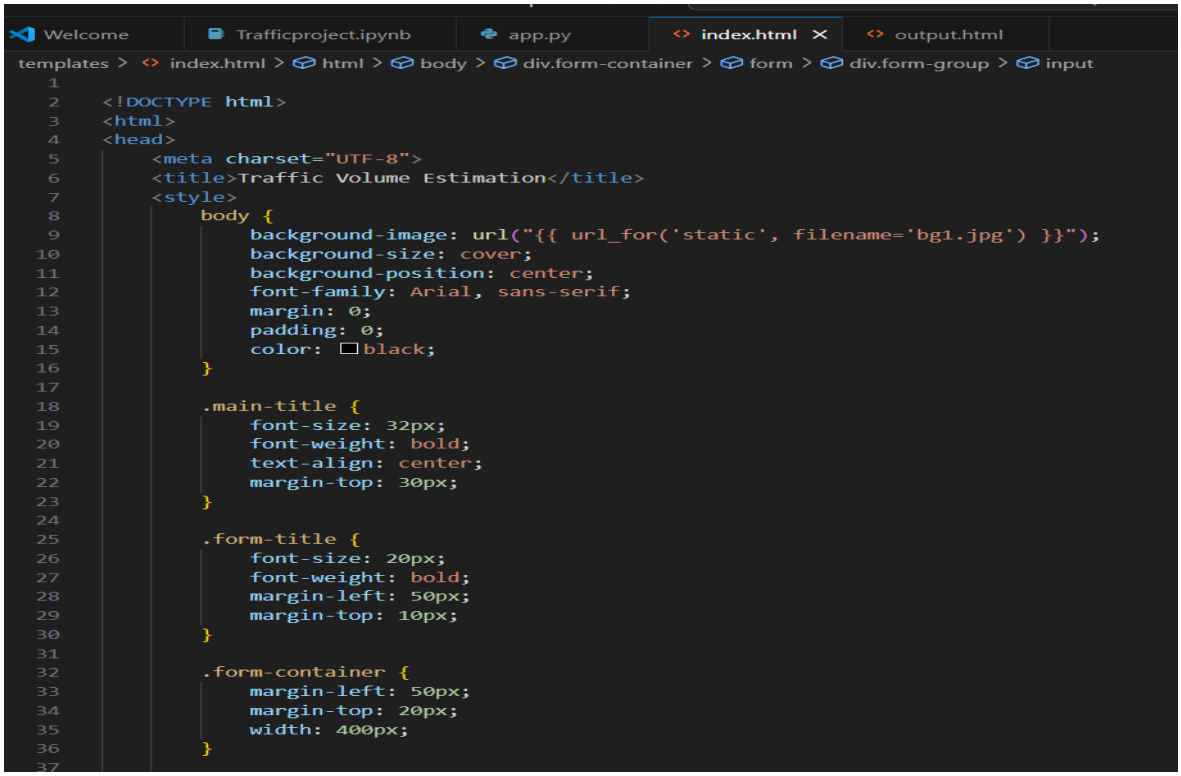
11. Screenshots or Demo

- **Live Demo Video:**

https://drive.google.com/file/d/1jlAPutb7_9xUVzKC-f2sMZian18goB1V/view?usp=sharing

The screenshots include:

- Input Form Page (index.html) with form and bg1.jpg

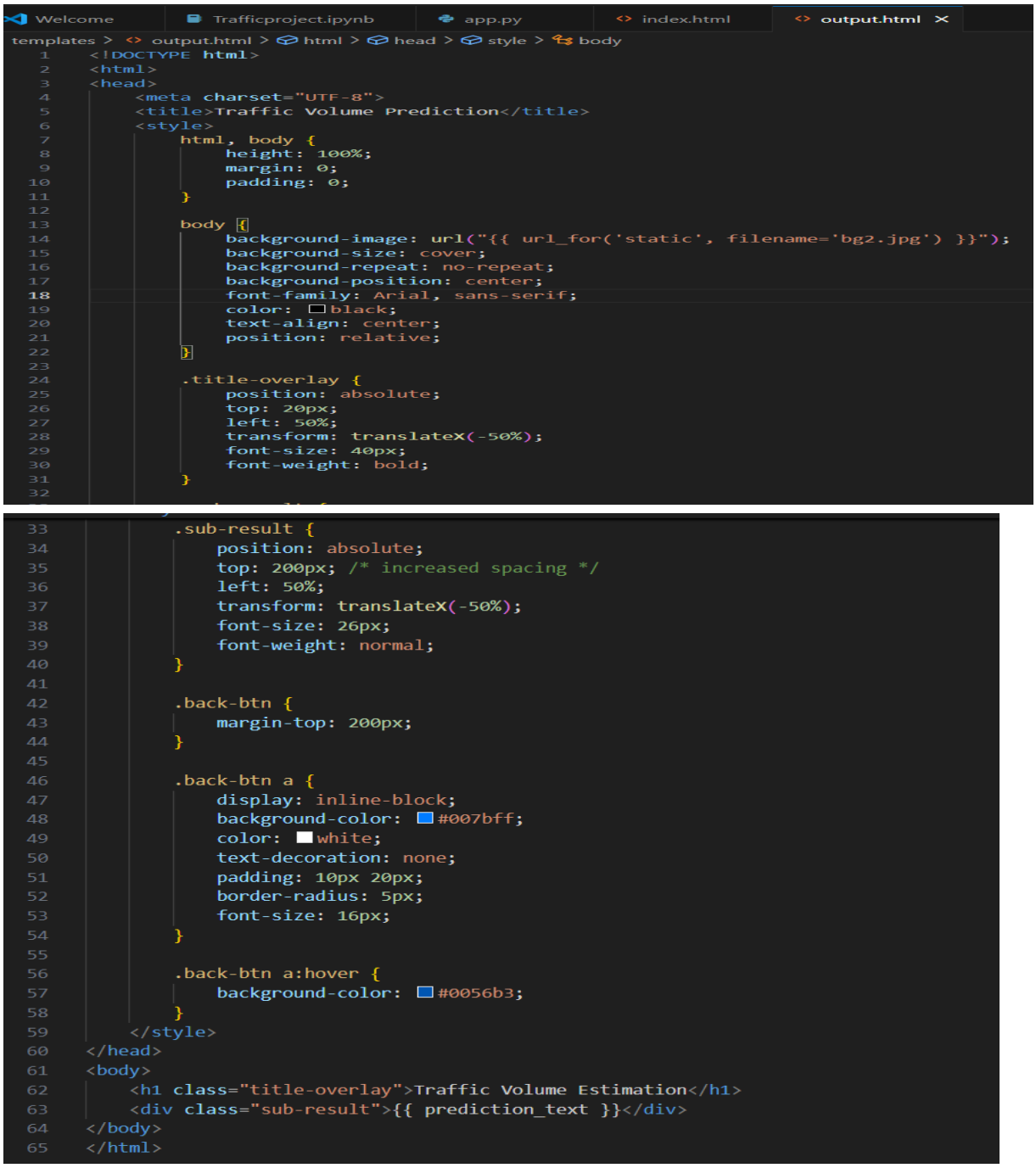


```
        .form-group {
            display: flex;
            align-items: center;
            margin-bottom: 10px;
        }

        .form-group label {
            width: 100px;
            font-weight: normal;
            text-align: left;
        }

        .form-group input,
        .form-group select {
            width: 180px;
            padding: 5px;
            border: 1px solid #ccc;
            border-radius: 3px;
        }

        input[type="submit"] {
            margin-top: 15px;
            padding: 6px 16px;
            background-color: white;
            color: black;
            border: 2px solid black;
            border-radius: 4px;
            font-weight: bold;
            cursor: pointer;
        }

```

```
        input[type="submit"]:hover {
            background-color: #f0f0f0;
        }

        .prediction-text {
            margin-top: 20px;
            font-weight: bold;
        }
    </style>
</head>
<body>
    <h1 class="main-title">Traffic Volume Estimation</h1>
    <h1 class="form-title">Please enter the following details</h1>

    <div class="form-container">
        <form action="/predict" method="post">
            <div class="form-group">
                <label for="holiday">holiday:</label>
                <select name="holiday" id="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>
            </div>
```

```
102
103             <div class="form-group">
104                 <label for="temp">temp:</label>
105                 <input type="text" name="temp" id="temp" placeholder="temp">
106             </div>
107
108             <div class="form-group">
109                 <label for="rain">rain:</label>
110                 <input type="text" name="rain" id="rain" placeholder="rain">
111             </div>
112
113             <div class="form-group">
114                 <label for="snow">snow:</label>
115                 <input type="text" name="snow" id="snow" placeholder="snow">
116             </div>
117
118             <div class="form-group">
119                 <label for="weather">weather:</label>
120                 <select name="weather" id="weather">
121                     <option value="1">Clouds</option>
122                     <option value="0">Clear</option>
123                     <option value="6">Rain</option>
124                     <option value="2">Drizzle</option>
125                     <option value="5">Mist</option>
126                     <option value="4">Haze</option>
127                     <option value="3">Fog</option>
128                     <option value="10">Thunderstorm</option>
129                     <option value="8">Snow</option>
130                     <option value="9">Squall</option>
131                     <option value="7">Smoke</option>
132                 </select>
133             </div>
134
135             <div class="form-group">
136                 <label>year:</label>
137                 <input type="number" min="2012" max="2025" name="year" placeholder="year" required>
138             </div>
139
140             <div class="form-group">
141                 <label>month:</label>
142                 <input type="number" min="1" max="12" name="month" placeholder="month" required>
143             </div>
144
145             <div class="form-group">
146                 <label>day:</label>
147                 <input type="number" min="1" max="31" name="day" placeholder="day" required>
148             </div>
149
150             <div class="form-group">
151                 <label>hours:</label>
152                 <input type="number" min="0" max="24" name="hours" placeholder="hours" required>
153             </div>
154
155             <div class="form-group">
156                 <label>minutes:</label>
157                 <input type="number" min="0" max="60" name="minutes" placeholder="minutes" required>
158             </div>
159
160             <div class="form-group">
161                 <label>seconds:</label>
162                 <input type="number" min="0" max="60" name="seconds" placeholder="seconds" required>
163             </div>
164
165             <input type="submit" value="Predict">
166         </form>
167
168             <div class="prediction-text">
169                 {{ prediction_text }}
170             </div>
171         </div>
172     </body>
173     </html>
```

- Prediction Output Page (output.html) showing predicted traffic and bg2.jpg



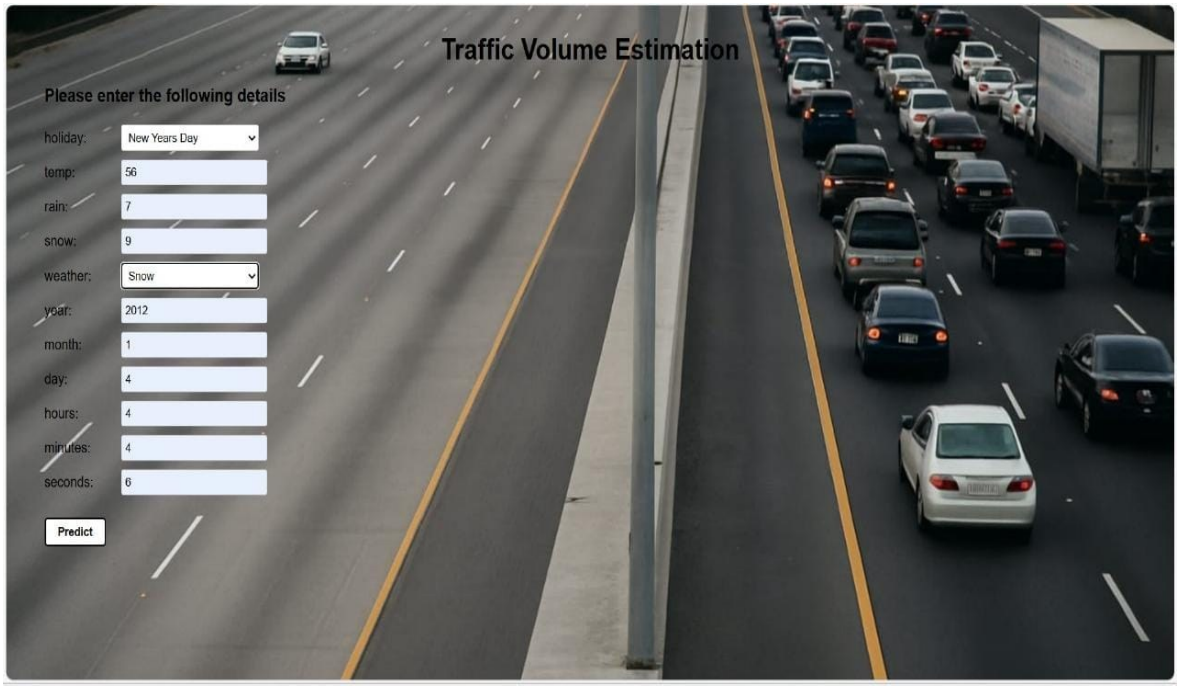
```
templates > output.html > html > head > style > body
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Traffic Volume Prediction</title>
    <style>
        html, body {
            height: 100%;
            margin: 0;
            padding: 0;
        }

        body {
            background-image: url("{{ url_for('static', filename='bg2.jpg') }}");
            background-size: cover;
            background-repeat: no-repeat;
            background-position: center;
            font-family: Arial, sans-serif;
            color: black;
            text-align: center;
            position: relative;
        }

        .title-overlay {
            position: absolute;
            top: 20px;
            left: 50%;
            transform: translateX(-50%);
            font-size: 40px;
            font-weight: bold;
        }

        .sub-result {
            position: absolute;
            top: 200px; /* increased spacing */
            left: 50%;
            transform: translateX(-50%);
            font-size: 26px;
            font-weight: normal;
        }

        .back-btn {
            margin-top: 200px;
        }

        .back-btn a {
            display: inline-block;
            background-color: #007bff;
            color: white;
            text-decoration: none;
            padding: 10px 20px;
            border-radius: 5px;
            font-size: 16px;
        }

        .back-btn a:hover {
            background-color: #0056b3;
        }
    </style>
</head>
<body>
    <h1 class="title-overlay">Traffic Volume Estimation</h1>
    <div class="sub-result">{{ prediction_text }}</div>
</body>
</html>
```

- **Flask file -app.py**

```python
app.py > predict
1
2    from flask import Flask, render_template, request
3    import numpy as np
4    import pickle
5    import warnings
6    warnings.filterwarnings("ignore", category=UserWarning)
7
8
9    app = Flask(__name__)
10
11   # Load the trained model
12   model = pickle.load(open('model.pkl', 'rb'))
13
14   @app.route('/')
15   def home():
16       return render_template('index.html')
17
18   @app.route('/predict', methods=['POST'])
19   def predict():
20       try:
21           # Get input from form
22           features = [
23               int(request.form['holiday']),
24               float(request.form['temp']),
25               float(request.form['rain']),
26               float(request.form['snow']),
27               int(request.form['weather']),
28               int(request.form['year']),
29               int(request.form['month']),
30               int(request.form['day']),
31               int(request.form['hours']),
32               int(request.form['minutes']),
33               int(request.form['seconds'])
34           ]
35
36           # Make prediction
37           prediction = model.predict([features])
⚠ 0   Connect
```

```python
38           result = int(prediction[0])
39
40           return render_template('output.html', prediction_text=f"Estimated Traffic Volume is: {result} units")
41
42       except Exception as e:
43           return render_template('output.html', prediction_text=f"Error: {str(e)}")
44
45   if __name__ == '__main__':
46       app.run(debug=True)
```

- **Output**


Traffic Volume Estimation
Please enter the following details
holiday:
New Years Day
temp:
56
rain:
7
snow:
9
weather:
Snow
year:
2012
month:
1
day:
4
hours:
4
minutes:
4
seconds:
6
Predict


Traffic Volume Estimation
Estimated Traffic Volume is: 1316 units
192 UYP

12. Known Issues

While *TrafficTelligence* performs effectively as a functional prototype, there are several known limitations that can be addressed in future versions:

- **No Field-Level Error Messages**
  The current version does not validate user inputs at the form level. If a user leaves a required field empty or enters an invalid format (e.g., text instead of numbers), the application does not display a clear error message. This may lead to confusion or backend errors.
- **No Real-Time API Integration**
  All input data is entered manually by the user. The system does not currently fetch live weather, traffic, or GPS data from APIs such as OpenWeatherMap or Google Maps. This reduces the effectiveness of the model in live or real-world deployment scenarios.
- **Model is Static (Trained Once)**
  The machine learning model was trained once on a static dataset and saved as model.pkl. It does not adapt or retrain over time with new user inputs or traffic trends. This limits its long-term accuracy and responsiveness to changing patterns.
- **Limited Dataset Scope**
  The training dataset is based on specific regions and timeframes. If the system is used in a different location or under unusual conditions (e.g., festivals, events, rural areas), the prediction may not be accurate due to lack of relevant training data.
- **UI is Not Mobile-Responsive**
  The HTML interface was designed for desktop use. On smaller screens or mobile devices, elements may not align properly, and the user experience can be degraded. This can be improved by incorporating responsive design techniques using CSS frameworks.

13. Future Enhancements

As a working prototype, *TrafficTelligence* demonstrates the potential of using machine learning for traffic prediction. However, to make it production-ready and more impactful, several enhancements are proposed:

🔄 **Live Data from Weather and GPS APIs**

- Integrate APIs such as **OpenWeatherMap**, **Google Maps**, or **HERE Traffic API** to fetch **real-time weather**, **location**, and **traffic conditions**.
- This would eliminate manual data entry and ensure more accurate, location-aware predictions in real-time scenarios.

🌐 **Mobile Responsiveness with Bootstrap or Tailwind CSS**

- Improve the frontend by integrating **responsive CSS frameworks** like Bootstrap or Tailwind CSS.
- This would ensure the interface works seamlessly on all devices, including tablets and smartphones, making it suitable for everyday commuter use.

🔐 **Authentication System with User Profiles**

- Implement **login and registration systems** to allow personalized access for users.
- Features like **saved history**, **user-specific dashboards**, and **access control** can be enabled, improving security and user engagement.

☁️ **Cloud Deployment Using Render, Vercel, or AWS**

- Deploy the application on platforms such as **Render, Heroku, Vercel**, or **AWS EC2/S3** for public access.
- This ensures scalability, faster access, uptime monitoring, and secure hosting across various environments.

🧠 **Advanced Models like XGBoost or Deep Learning**

- Explore more powerful algorithms such as **XGBoost**, **Gradient Boosting**, or **Deep Learning (LSTM, ANN)** for improved prediction accuracy.
- Ensemble techniques and time-series models can further enhance performance and generalization.

📊 **Logging Prediction Data to Cloud Database**

- Store user inputs and prediction outputs in a **cloud-based database** like **Firebase**, **MongoDB Atlas**, or **Google Cloud Firestore**.