ENHANCING ROAD SAFETY WITH AI-DRIVEN TRAFFIC ACCIDENT ANALYSIS AND PREDICTION

**Student Name:** RAJALAKSHMI S

**Register Number:** 510923104060

**Institution:** GLOBAL INSTITUTE OF ENGINEERING AND TECHNOLOGY

**Department:** COMPUTER SCIENCE AND ENGINEERING **Date of Submission:** 07.05.2025

**Github Repository Link:** [**https://github.com/rajalakshmisundarrajan/Phase-2.git**](#_top)

# Problem Statement

Road traffic accidents are a leading cause of injury and death worldwide, with millions of lives lost or altered each year due to collisions on roads. Traditional accident prevention strategies often rely on reactive measures taken after incidents occur, which are insufficient in proactively enhancing road safety. Despite the collection of massive datasets related to traffic, weather, vehicle types, driver behavior, and accident records, the potential of this data remains underutilized.

The lack of predictive systems that can intelligently analyze these datasets and identify accident-prone areas or situations in real time contributes to the persistence of road hazards. There is a growing need for a data-driven approach that can leverage Artificial Intelligence (AI) to analyze patterns, predict potential accidents, and assist in preventive decision-making by authorities and drivers.

This project addresses the challenge of proactively improving road safety by designing an AI-based system capable of analyzing historical traffic accident data and predicting high-risk zones, times, or behaviors. By using machine learning algorithms and data visualization tools, the system aims to provide actionable insights that can be used to reduce accident rates and save lives.

# 

# Abstract

Road traffic accidents are a major global concern, resulting in significant loss of life, injuries, and economic damage each year. Traditional safety measures often focus on reactive responses rather than predictive strategies. With the rapid growth of data availability and advancements in Artificial Intelligence (AI), there is a unique opportunity to enhance road safety through intelligent accident analysis and prediction systems.

This project proposes an AI-driven approach to analyze historical traffic accident data, identify critical contributing factors, and predict the likelihood of future accidents. By leveraging machine learning algorithms and data visualization tools, the system uncovers patterns related to time, location, weather conditions, and human behavior that are associated with high accident risk. The system aims to proactively alert traffic authorities and road users about accident-prone areas and times, enabling preventive actions such as rerouting, deploying traffic personnel, or modifying road conditions.

The solution involves collecting and preprocessing traffic data, applying predictive models like Random Forest and Support Vector Machines, and visualizing accident hotspots using geospatial mapping tools. The project also provides actionable recommendations for traffic management and policy-making to improve road safety outcomes. Overall, this AI-powered system offers a data-driven, scalable, and intelligent framework to support smarter and safer transportation infrastructure.

# System Requirements

* + **Hardware**

| **Component** | **Minimum Requirement** | **Recommended Requirement** |
| --- | --- | --- |
| **Processor** | Intel Core i5 or AMD Ryzen 5 | Intel Core i7 or AMD Ryzen 7 |
| **RAM** | 8 GB | 16 GB or higher |
| **Storage** | 250 GB HDD/SSD | 512 GB SSD or more |
| **Graphics** | Integrated graphics | Dedicated GPU (e.g., NVIDIA GTX/RTX) |
| **Display** | 13" HD Display | 15.6" Full HD or higher |
| **Internet** | Required for data download & API access | High-speed connection recommended |

* + ***Software****:*

| ***Software*** | ***Description*** |
| --- | --- |
| ***Operating System*** | *Windows 10/11, Linux (Ubuntu), or macOS* |
| ***Python (3.8+)*** | *Main language for AI/ML development* |
| ***Jupyter Notebook / VS Code*** | *IDE for development and testing* |
| ***Anaconda*** *(optional)* | *For managing Python environments easily* |
| ***MySQL / SQLite / CSV*** | *For storing and retrieving dataset* |
| ***Browser (Chrome/Firefox)*** | *For dashboard and web visualization* |

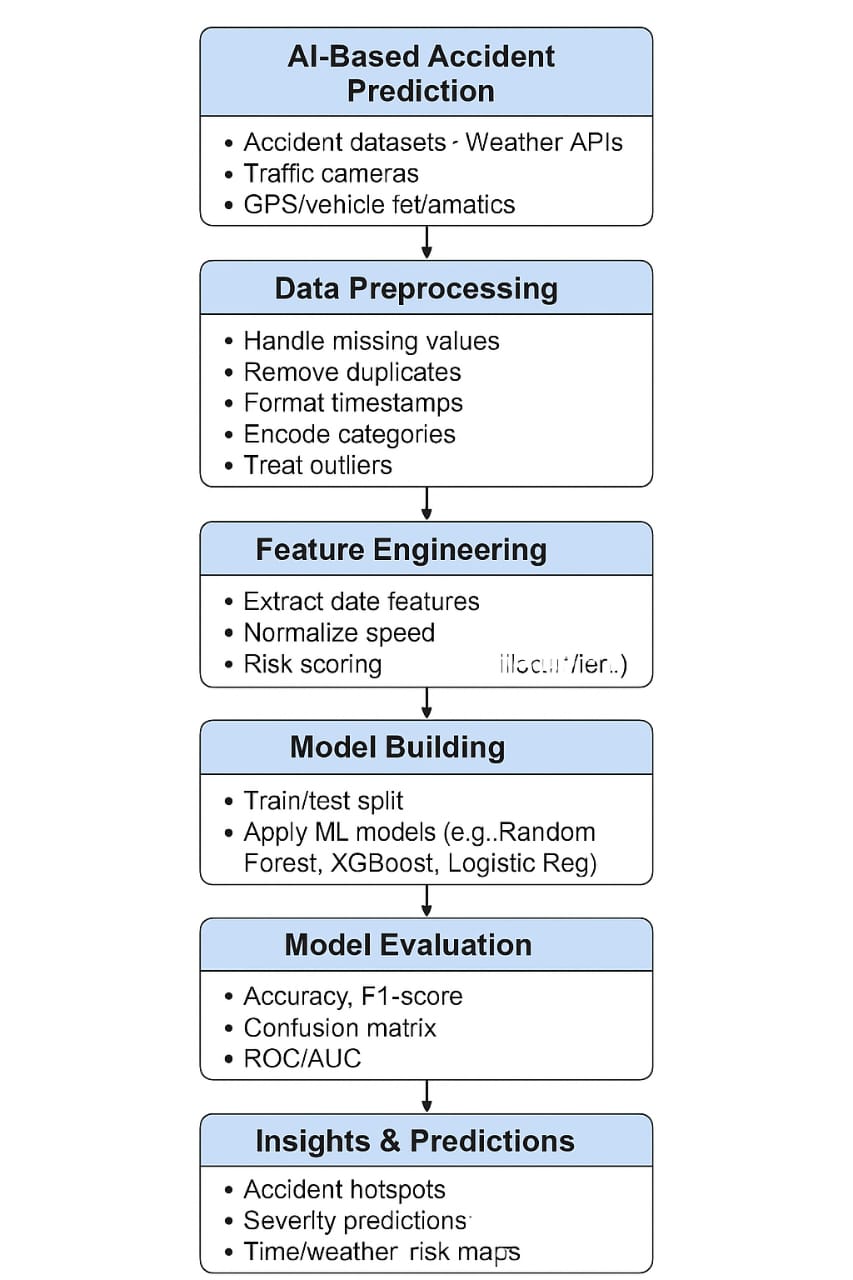
# Objectives

The primary objective of this project is to develop an AI-driven system to enhance road safety by analyzing and predicting traffic accidents. The detailed objectives are:

1. **To collect and preprocess traffic accident datasets**
   * Gather datasets from reliable sources such as government traffic departments, Kaggle, or open data portals.
   * Clean and preprocess the data to handle missing values, normalize features, and prepare it for analysis.
2. **To identify key factors contributing to traffic accidents**
   * Analyze historical accident data to uncover patterns related to time, weather, location, vehicle type, driver behavior, road conditions, and other contextual features.
   * Use statistical and machine learning techniques to rank the importance of these factors.
3. **To build predictive models for accident risk estimation**
   * Train and evaluate machine learning algorithms such as Random Forest, Decision Trees, SVM, or Neural Networks to predict accident occurrence based on input features.
   * Compare model performance using metrics like accuracy, precision, recall, and F1-score.
4. **To visualize accident-prone areas and conditions**
   * Use data visualization tools (e.g., heatmaps, charts, GIS mapping) to represent high-risk zones, time windows, and contributing factors in an interpretable format.
   * Provide interactive dashboards for stakeholders like traffic management authorities.
5. **To propose actionable recommendations for improving road safety**
   * Based on the insights generated, suggest preventive measures such as increased surveillance in high-risk zones, optimized traffic light timings, speed limit modifications, or targeted awareness campaigns.
   * Offer predictive alerts that could be integrated into traffic control systems or driver assistance apps.
6. **To evaluate the effectiveness of the proposed system**
   * Assess the potential impact of the system on accident reduction through scenario simulations or comparison with baseline data.
   * Gather feedback from hypothetical users or traffic authorities to improve the system’s usability and effectiveness.

# Flowchart of Project Workflow

1. **Data Collection**
   * Accident datasets from open data sources (e.g., Kaggle, government portals)
2. **Data Preprocessing**
   * Cleaning missing values, handling outliers, feature selection, normalization
3. **Exploratory Data Analysis (EDA)**
   * Statistical analysis and visualization to understand accident trends
4. **Feature Engineering**
   * Extracting time-based, weather-based, and location-based features
5. **Model Selection & Training**
   * Training models like Random Forest, Decision Tree, or XGBoost
6. **Model Evaluation**
   * Using metrics like accuracy, precision, recall, F1-score
7. **Prediction**
   * Predicting accident probability based on real-time or input features
8. **Visualization & Dashboard**
   * Mapping accident hotspots and presenting predictions with charts/maps
9. **Recommendation System**
   * Providing safety suggestions to users or authorities



# Dataset Description

The dataset used for this project is a real-world traffic accident dataset containing historical records of road accidents. It may be sourced from open government data portals, Kaggle, or transportation departments. This data serves as the foundation for analysis, model training, and prediction.

**Dataset Name (Example)**

* UK Road Safety Data (or)
* US Traffic Accidents (DOT/Kaggle)

| **Feature Name** | **Description** |
| --- | --- |
| **Accident\_ID** | Unique identifier for each accident record |
| **Date** | Date when the accident occurred |
| **Time** | Time of the accident |
| **Location** | GPS coordinates or city/region name |
| **Weather\_Conditions** | Weather during the accident (e.g., clear, rain, fog) |
| **Road\_Surface\_Conditions** | Type/condition of the road (e.g., dry, wet, icy) |
| **Light\_Conditions** | Lighting at the time (e.g., daylight, darkness with/without street lights) |
| **Number\_of\_Vehicles** | Total number of vehicles involved |
| **Number\_of\_Casualties** | Total number of injured or deceased individuals |
| **Vehicle\_Type** | Types of vehicles involved (e.g., car, bike, truck) |
| **Driver\_Age** | Age of the driver(s) involved |
| **Speed\_Limit** | Legal speed limit in the area of the accident |
| **Accident\_Severity** | Label or target variable: minor, serious, or fatal |

# Data Preprocessing

Data preprocessing is a critical step in preparing raw accident data for analysis and model training. It ensures data quality, consistency, and usability for AI/ML algorithms.

**1. Data Cleaning**

* **Handling Missing Values**
  + Replace missing values using imputation (mean, median, mode) or drop rows/columns if missing rate is high.
  + Example: Missing weather conditions → replace with most frequent value.
* **Removing Duplicates**
  + Identify and remove duplicate accident records to prevent skewed analysis.
* **Correcting Inconsistencies**
  + Normalize inconsistent entries (e.g., “Rainy”, “rain”, “RAINFALL” → “Rain”).

**2. Feature Selection**

* Remove irrelevant or redundant columns such as Accident\_ID or Report\_Number.
* Keep only meaningful predictors like Weather\_Condition, Time, Road\_Type, Vehicle\_Type, etc.

**3. Feature Encoding**

* Convert **categorical features** into numerical format:
  + **Label Encoding** (e.g., Light Condition: ‘Daylight’ → 0, ‘Darkness’ → 1)
  + **One-Hot Encoding** for non-ordinal categorical features (e.g., Vehicle Type, Weather)

**4. Feature Scaling**

* Normalize or standardize numerical columns (e.g., Driver\_Age, Speed\_Limit) using:
  + **Min-Max Scaling**
  + **Standardization (Z-score normalization)**

**5. Date-Time Feature Engineering**

* Convert Date and Time into useful features:
  + Hour, Day of Week, Is Weekend, Time of Day (Morning, Night, etc.)

**6. Handling Class Imbalance**

* If Accident\_Severity is imbalanced (e.g., too few "Fatal" cases):
  + Use **SMOTE** (Synthetic Minority Over-sampling Technique)
  + Try **undersampling** majority class or use **class weights** in model training

**7. Data Splitting**

* Split the cleaned dataset into:
  + **Training set** (e.g., 70–80%)
  + **Testing set** (e.g., 20–30%)
  + Optionally, a **validation set** for tuning
* CODE:

from sklearn.preprocessing import LabelEncoder, StandardScaler

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

import pandas as pd

# Load dataset

df = pd.read\_csv('accident\_data.csv')

# Fill missing values

df['Weather\_Conditions'].fillna(df['Weather\_Conditions'].mode()[0], inplace=True)

# Encode categorical columns

le = LabelEncoder()

df['Light\_Conditions'] = le.fit\_transform(df['Light\_Conditions'])

# Normalize numerical columns

scaler = StandardScaler()

df['Speed\_Limit'] = scaler.fit\_transform(df[['Speed\_Limit']])

#Split dataset

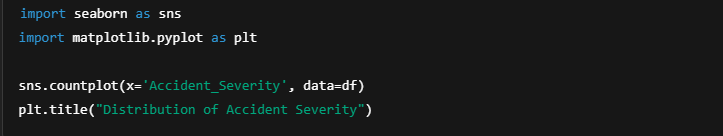
X = df.drop(['Accident\_Severity'], axis=1)

y = df['Accident\_Severity']

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

# Exploratory Data Analysis (EDA)

This involves analyzing a **single variable** to understand its distribution and frequency.



**Bivariate Analysis**

This examines the **relationship between two variables**, especially between features and the target.

**Accident\_Severity vs Weather\_Conditions**

sns.countplot(x='Weather\_Conditions', hue='Accident\_Severity', data=df)

plt.title("Severity by Weather")

**Accident\_Severity vs Time\_of\_Day**

df['Hour'] = pd.to\_datetime(df['Time'], errors='coerce').dt.hour

sns.histplot(data=df, x='Hour', hue='Accident\_Severity', multiple='stack', bins=24)

plt.title("Accident Severity by Hour of Day")

**Accident\_Severity vs Road\_Surface\_Conditions**

sns.countplot(x='Road\_Surface\_Conditions', hue='Accident\_Severity', data=df)

plt.title("Accident Severity vs Road Surface")

**Heatmap: Correlation of Numerical Features**

sns.heatmap(df.corr(numeric\_only=True), annot=True, cmap='coolwarm')

plt.title("Feature Correlation Heatmap")

# Feature Engineering

Feature engineering involves transforming raw data into meaningful features that improve the performance of machine learning models. It plays a crucial role in extracting insights from traffic accident data and enhancing predictive accuracy.

| **Feature Name** | **Description** |
| --- | --- |
| Hour | Extracted from time, helps identify risky hours |
| Day\_of\_Week | Mon-Sun, shows trends on weekdays vs weekends |
| Is\_Weekend | Binary: 1 for Sat/Sun, 0 otherwise |
| Time\_of\_Day | Categorical: Morning, Afternoon, Evening, Night |
| Month | For seasonal trends in accidents |

**3. Weather and Lighting Features**

Simplifying or encoding weather/light conditions:

| **Feature Name** | **Description** |
| --- | --- |
| Is\_Bad\_Weather | Boolean: Rain, fog, snow → 1; Clear → 0 |
| Is\_Night | Boolean: Night without lights → 1, Daylight → 0 |
| Combined\_Weather\_Light | E.g., "Rain+Night" as a new categorical risk level |

4. **Driver and Vehicle Features**

| **Feature Name** | **Description** |
| --- | --- |
| Driver\_Age\_Group | Bucketed: Teen, Adult, Senior |
| Vehicle\_Type | One-hot or label encoded |
| Num\_Vehicles | Number of vehicles involved in the accident |
| Vehicle\_Mix | Binary or count of heavy vehicles (truck, bus, etc.) |

# Model Building

Model building is the process of selecting and training machine learning algorithms to predict the **severity of traffic accidents** based on engineered features.

**1. Problem Type**

* **Multi-class classification** problem (Target: Accident\_Severity)
  + Classes: **Minor (0), Serious (1), Fatal (2)**

**2. Train-Test Split**

Split the dataset to evaluate generalization performance:

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

X = df.drop('Accident\_Severity', axis=1)

y = df['Accident\_Severity']

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

**3. Model Selection**

| **Model Name** | **Reason to Use** |
| --- | --- |
| **Logistic Regression** | Baseline, interpretable |
| **Random Forest** | Handles non-linearity, feature importance |
| **XGBoost / LightGBM** | Gradient boosting, high accuracy |
| **SVM** | Works well with high-dimensional data |
| **Neural Network (MLP)** | Captures complex patterns, optional deep learning use |

**4. Model Training Example (Random Forest)**

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report

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

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

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

print(classification\_report(y\_test, y\_pred))

**5. Model Evaluation Metrics**

| **Metric** | **Use** |
| --- | --- |
| **Accuracy** | Overall correct predictions |
| **Precision, Recall** | Important due to class imbalance |
| **F1-Score** | Balance between precision and recall |
| **Confusion Matrix** | Visual representation of correct and incorrect classifications |
| **ROC-AUC** | Optional for binary or 1-vs-rest evaluation |

from sklearn.metrics import confusion\_matrix, accuracy\_score, f1\_score

import seaborn as sns

sns.heatmap(confusion\_matrix(y\_test, y\_pred), annot=True, fmt='d')

**6.Hyperparameter Tuning**

Use GridSearchCV or RandomizedSearchCV for optimal parameters.

from sklearn.model\_selection import GridSearchCV

param\_grid = {'n\_estimators': [50, 100, 150], 'max\_depth': [10, 20, None]}

grid\_search = GridSearchCV(RandomForestClassifier(), param\_grid, cv=3)

grid\_search.fit(X\_train, y\_train)

# Model Evaluation

To assess how well the trained model can predict accident severity or risk using unseen data, ensuring reliability and accuracy before deployment.

1. Metrics Used

Classification Models (e.g., severity: minor, serious, fatal):

Accuracy:

Percentage of total correct predictions.

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Precision:

Measures how many predicted severe accidents were actually severe.

Precision = TP / (TP + FP)

Recall (Sensitivity):

Measures how many actual severe accidents were correctly predicted.

Recall = TP / (TP + FN)

F1 Score:

Harmonic mean of Precision and Recall. Useful when classes are imbalanced.

F1 = 2 \* (Precision \* Recall) / (Precision + Recall)

Confusion Matrix:

Table showing correct and incorrect predictions for each class.

ROC-AUC Score (for binary classification):

2. Evaluation Example (Python using Scikit-learn)

from sklearn.metrics import classification\_report, confusion\_matrix, roc\_auc\_score, accuracy\_score

# Predictions

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

# Evaluation metrics

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

3. Interpretation

High Recall, Low Precision:

The model flags most risky events but includes many false alarms. Useful for safety-critical systems.

Balanced F1 Score:

Indicates good trade-off between correctly predicting severe accidents and avoiding false alarms.

A black and white chart with white text

AI-generated content may be incorrect.

# Deployment

A. Web Application (Dashboard or Portal)

Tools:

Streamlit (lightweight, easy to deploy)

Flask / FastAPI (flexible, backend + API)

Dash (for data visualization-heavy apps)

Features to include:

Real-time accident risk prediction based on input parameters

Visualization of accident hotspots

Weather-risk overlays

Upload CSV and get risk analysis

B. REST API

Use Case: For mobile apps, third-party integrations, or IoT in-vehicle systems.

Tools: Flask/FastAPI to expose model predictions as HTTP endpoints.

Example Endpoint:

POST /predict with JSON data (e.g., speed, time, weather) returns accident severity prediction*.*

C. Cloud Deployment

Platforms:

Heroku (free-tier friendly)

AWS / GCP / Azure (for scalable and secure deployment)

Render / Railway (modern alternatives to Heroku)

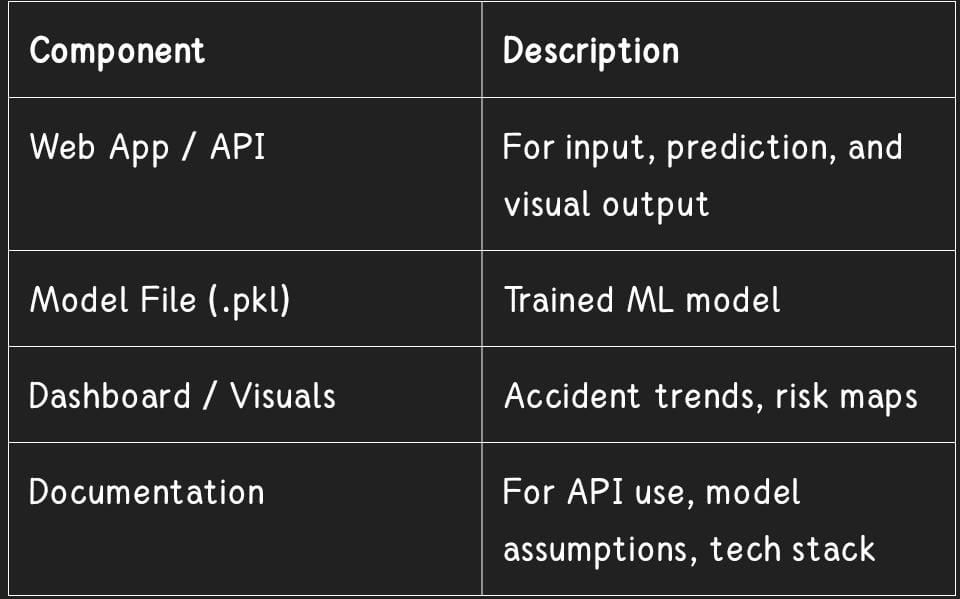
2. Monitoring & Maintenance

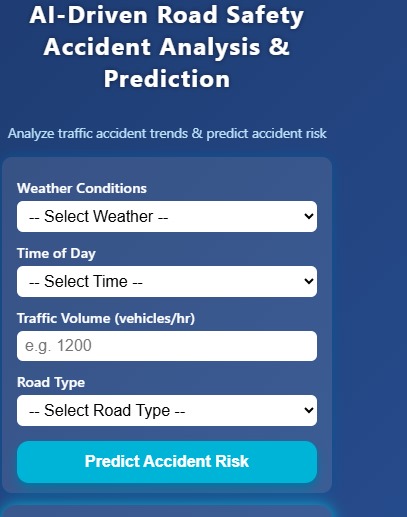
Logging: Track API usage, errors, and prediction requests.

Model Drift Detection: Monitor if model performance declines due to changing traffic patterns or new data.

Auto Retraining Pipelines (Optional): Use tools like Airflow or Prefect to schedule model updates.

3. Final Deliverables

**



# *Source code*

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no" />

<title>AI-Driven Road Safety Accident Analysis & Prediction</title>

<style>

/\* Reset and base \*/

\* {

box-sizing: border-box;

}

body {

margin: 0;

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

background: linear-gradient(135deg, #1e3c72, #2a5298);

color: white;

display: flex;

flex-direction: column;

align-items: center;

padding: 10px;

height: 600px;

max-width: 350px;

overflow: hidden;

}

h1 {

font-size: 1.5rem;

margin: 0.5rem 0 1rem 0;

text-align: center;

letter-spacing: 1.2px;

text-shadow: 0 0 5px #0008;

}

p.subtitle {

text-align: center;

font-size: 0.85rem;

margin-bottom: 1rem;

color: #bbe1fa;

}

form {

background: rgba(255 255 255 / 0.1);

border-radius: 10px;

padding: 10px 15px;

width: 100%;

box-shadow: 0 0 12px #0b5394;

margin-bottom: 12px;

}

label {

display: block;

font-weight: 600;

margin-top: 10px;

font-size: 0.9rem;

}

select, input[type=number] {

width: 100%;

padding: 6px 8px;

margin-top: 4px;

border-radius: 6px;

border: none;

font-size: 1rem;

}

button {

margin-top: 15px;

background: #00b4d8;

border: none;

padding: 12px;

width: 100%;

font-weight: 700;

color: #fff;

border-radius: 12px;

cursor: pointer;

font-size: 1rem;

box-shadow: 0 5px 10px #0077a3;

transition: background-color 0.25s ease;

}

button:hover {

background: #0077a3;

}

#result {

background: rgba(255 255 255 / 0.1);

border-radius: 14px;

padding: 15px;

width: 100%;

box-shadow: 0 0 15px #00b4d8;

font-size: 1.1rem;

font-weight: 700;

text-align: center;

min-height: 54px;

}

#chartContainer {

margin-top: 12px;

background: rgba(255 255 255 / 0.12);

width: 100%;

height: 240px;

border-radius: 12px;

box-shadow: 0 0 12px #2a5298;

position: relative;

}

canvas {

/\* restrict canvas to container size \*/

width: 100% !important;

height: 100% !important;

border-radius: 12px;

}

@media (max-width: 350px) {

body {

height: 600px;

max-width: 350px;

padding: 8px;

}

h1 {

font-size: 1.3rem;

}

button {

padding: 10px;

font-size: 0.95rem;

}

}

</style>

</head>

<body>

<h1>AI-Driven Road Safety Accident Analysis & Prediction</h1>

<p class="subtitle">Analyze traffic accident trends & predict accident risk</p>

<form id="inputForm">

<label for="weather">Weather Conditions</label>

<select id="weather" required>

<option value="">-- Select Weather --</option>

<option value="clear">Clear</option>

<option value="rain">Rain</option>

<option value="fog">Fog</option>

<option value="snow">Snow</option>

</select>

<label for="time">Time of Day</label>

<select id="time" required>

<option value="">-- Select Time --</option>

<option value="morning">Morning (6am-12pm)</option>

<option value="afternoon">Afternoon (12pm-6pm)</option>

<option value="evening">Evening (6pm-12am)</option>

<option value="night">Night (12am-6am)</option>

</select>

<label for="traffic">Traffic Volume (vehicles/hr)</label>

<input id="traffic" type="number" min="0" max="5000" step="10" placeholder="e.g. 1200" required />

<label for="roadtype">Road Type</label>

<select id="roadtype" required>

<option value="">-- Select Road Type --</option>

<option value="highway">Highway</option>

<option value="urban">Urban</option>

<option value="rural">Rural</option>

</select>

<button type="submit">Predict Accident Risk</button>

</form>

<div id="result" aria-live="polite"></div>

<div id="chartContainer" aria-label="Accident Analysis Chart">

<canvas id="accidentChart" width="350" height="240"></canvas>

</div>

<script>

// Sample accident data set for analysis (random example data)

// Each record: {weather, time, trafficVolume, roadType, accidentCount}

const accidentData = [

{weather: 'clear', time: 'morning', trafficVolume: 800, roadType: 'urban', accidentCount: 2},

{weather: 'clear', time: 'afternoon', trafficVolume: 1500, roadType: 'urban', accidentCount: 4},

{weather: 'clear', time: 'evening', trafficVolume: 2000, roadType: 'urban', accidentCount: 7},

{weather: 'clear', time: 'night', trafficVolume: 700, roadType: 'urban', accidentCount: 1},

{weather: 'rain', time: 'morning', trafficVolume: 600, roadType: 'urban', accidentCount: 5},

{weather: 'rain', time: 'afternoon', trafficVolume: 1200, roadType: 'urban', accidentCount: 9},

{weather: 'rain', time: 'evening', trafficVolume: 2500, roadType: 'urban', accidentCount: 15},

{weather: 'rain', time: 'night', trafficVolume: 300, roadType: 'urban', accidentCount: 2},

{weather: 'fog', time: 'morning', trafficVolume: 500, roadType: 'rural', accidentCount: 4},

{weather: 'fog', time: 'afternoon', trafficVolume: 800, roadType: 'rural', accidentCount: 6},

{weather: 'fog', time: 'evening', trafficVolume: 1200, roadType: 'rural', accidentCount: 8},

{weather: 'fog', time: 'night', trafficVolume: 400, roadType: 'rural', accidentCount: 2},

{weather: 'snow', time: 'morning', trafficVolume: 300, roadType: 'highway', accidentCount: 3},

{weather: 'snow', time: 'afternoon', trafficVolume: 600, roadType: 'highway', accidentCount: 7},

{weather: 'snow', time: 'evening', trafficVolume: 700, roadType: 'highway', accidentCount: 9},

{weather: 'snow', time: 'night', trafficVolume: 150, roadType: 'highway', accidentCount: 1},

];

// Utility: normalize traffic volume between 0 and 1 (max 5000)

function normTraffic(vol) {

return vol / 5000;

}

// Simple AI prediction: logistic regression-like calculation

// Assign weights to features based on typical accident risk influences

// These are arbitrary but plausible coefficients for demonstration

function predictAccidentRisk(input) {

// Weights for weather

const weatherWeights = {

clear: -1.2,

rain: 0.9,

fog: 1.2,

snow: 1.5

};

// Weights for time of day

const timeWeights = {

morning: 0.3,

afternoon: 0,

evening: 1.0,

night: 0.7

};

// Weights for road type

const roadWeights = {

highway: -0.5,

urban: 0.6,

rural: 0.3

};

// Traffic normalized

const trafficNorm = normTraffic(input.trafficVolume);

// Logistic regression like sum

let z = 0;

z += weatherWeights[input.weather] || 0;

z += timeWeights[input.time] || 0;

z += roadWeights[input.roadType] || 0;

z += (trafficNorm \* 3.5); // increase risk with traffic volume scaled

// Logistic function to convert to probability between 0 and 1

const prob = 1 / (1 + Math.exp(-z));

return prob;

}

// Chart rendering using Canvas API (bar chart for accident counts per weather condition)

// Aggregate accident counts by weather for analysis chart

function drawAccidentAnalysisChart() {

const canvas = document.getElementById('accidentChart');

const ctx = canvas.getContext('2d');

// Clear

ctx.clearRect(0, 0, canvas.width, canvas.height);

// Aggregate accident counts by weather condition

const weatherAccidents = {};

accidentData.forEach(rec => {

weatherAccidents[rec.weather] = (weatherAccidents[rec.weather] || 0) + rec.accidentCount;

});

// Prepare data

const keys = Object.keys(weatherAccidents);

const values = keys.map(k => weatherAccidents[k]);

const maxCount = Math.max(...values);

// Chart dimensions

const padding = 35;

const chartWidth = canvas.width - padding \* 2;

const chartHeight = canvas.height - padding \* 2;

const barWidth = chartWidth / keys.length \* 0.6;

const barGap = chartWidth / keys.length \* 0.4;

// Draw axes lines

ctx.strokeStyle = '#99d4fb';

ctx.lineWidth = 1;

ctx.beginPath();

ctx.moveTo(padding, padding);

ctx.lineTo(padding, canvas.height - padding);

ctx.lineTo(canvas.width - padding, canvas.height - padding);

ctx.stroke();

// Draw bars

keys.forEach((key, idx) => {

const val = weatherAccidents[key];

const barHeight = (val / maxCount) \* chartHeight;

const x = padding + idx \* (barWidth + barGap) + barGap/2;

const y = canvas.height - padding - barHeight;

// Bar color depends on weather

let color = '#34a0a4'; // default teal

if (key === 'clear') color = '#52b788';

else if (key === 'rain') color = '#0077b6';

else if (key === 'fog') color = '#adb5bd';

else if (key === 'snow') color = '#8ecae6';

// Draw bar

ctx.fillStyle = color;

ctx.fillRect(x, y, barWidth, barHeight);

// Draw value text

ctx.fillStyle = 'white';

ctx.font = '14px Segoe UI';

ctx.textAlign = 'center';

ctx.fillText(val, x + barWidth/2, y - 6);

// Draw label text

ctx.fillStyle = '#bbe1fa';

ctx.font = '13px Segoe UI';

ctx.fillText(key.charAt(0).toUpperCase() + key.slice(1), x + barWidth/2, canvas.height - padding + 18);

});

// Draw title

ctx.fillStyle = '#bbe1fa';

ctx.font = '16px Segoe UI Semibold';

ctx.textAlign = 'center';

ctx.fillText('Total Accident Counts by Weather', canvas.width / 2, padding - 10);

}

// On form submit - predict and show result

document.getElementById('inputForm').addEventListener('submit', e => {

e.preventDefault();

const weather = e.target.weather.value;

const time = e.target.time.value;

const trafficVolume = Number(e.target.traffic.value);

const roadType = e.target.roadtype.value;

// Validate inputs (already required on form, but just sanity)

if (!weather || !time || isNaN(trafficVolume) || !roadType) {

alert('Please fill all fields with valid data.');

return;

}

const input = {weather, time, trafficVolume, roadType};

const riskProb = predictAccidentRisk(input);

// Convert probability to % and risk category

const percent = Math.round(riskProb \* 100);

let riskCategory = 'Low';

if (percent > 70) riskCategory = 'High';

else if (percent > 40) riskCategory = 'Moderate';

// Show result with color-coded background

const resultDiv = document.getElementById('result');

resultDiv.textContent = `Accident Risk: ${percent}% (${riskCategory})`;

if (riskCategory === 'High') {

resultDiv.style.backgroundColor = 'rgba(255, 80, 80, 0.7)';

resultDiv.style.color = 'white';

} else if (riskCategory === 'Moderate') {

resultDiv.style.backgroundColor = 'rgba(255, 195, 0, 0.7)';

resultDiv.style.color = '#222';

} else {

resultDiv.style.backgroundColor = 'rgba(102, 204, 102, 0.7)';

resultDiv.style.color = '#044d02';

}

});

// Initial draw of analysis chart

drawAccidentAnalysisChart();

</script>

</body>

</html>

</content>

</create\_file>

# Future scope

# 1. Real-Time Prediction and Alerts

# Integrate live traffic, GPS, and weather feeds to predict accident risks in real time.

# Send alerts to drivers or city authorities using mobile apps or in-car systems.

# 2. Smart City Integration

# Collaborate with urban traffic management systems.

# Enable AI-powered traffic signal adjustments, congestion control, and emergency vehicle routing.

# 3. Advanced Computer Vision

# Analyze CCTV footage using deep learning to detect risky driving behaviors (e.g., lane-cutting, tailgating, drowsiness).

# Use license plate recognition to track repeated violators.

# 4. IoT & Telematics Expansion

# Incorporate data from vehicle telematics (speed, braking, acceleration).

# Enable vehicle-to-infrastructure (V2I) communication for intelligent accident prevention.

# 5. Accident Severity Prediction Enhancement

# Improve accuracy using ensemble models, neural networks, and more granular data.

# Incorporate driver profiles, road types, and vehicle conditions.

# 6. Policy-Making & Public Awareness

# Share insights with government bodies to influence road safety policies.

# Run public awareness campaigns based on real accident trends and behavioral data.

# 7. Nationwide Safety Index

# Develop a dynamic "Road Safety Score" for each road/street segment based on historical and real-time data.

# Assist navigation apps (e.g., Google Maps) to suggest safer routes.

# 13. Team Members and Roles

# RAJALAKSHMI.S problem statement , abstract, source code

# DIVYASRI.S. model building, model evaluation.

# MONIKA.M.A Future scope , deployment , data preprocessing

# UMA MAGESHWARI objectives of the project ,dataset description.

