

## **Project Title:**

# **TrafficTelligence: Advanced Traffic Volume Estimation with Machine Learning**

## **Team Details**

- **Team ID: LTVIP2025TMID34460**
- **Team Size: 4**
- **Team Leader: Chebrolu Sriharsha**
- **Team Members:**
  - **Byreddy Mokshith**
  - **Ch Ranjith**
  - **Ch. Devika**

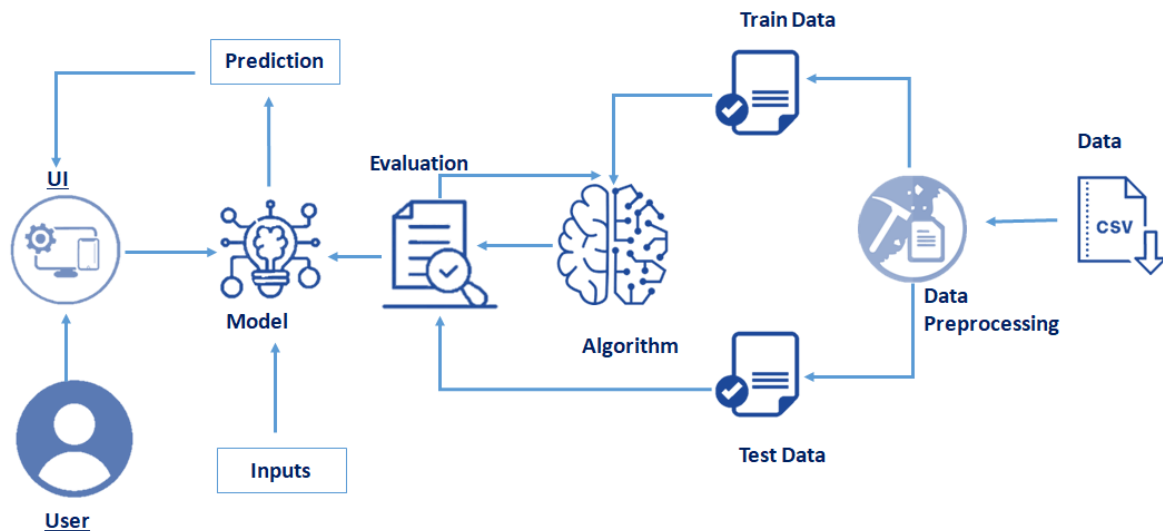
**Scenario 1:** Dynamic Traffic Management TrafficTelligence enables dynamic traffic management by providing real-time traffic volume estimations. Transportation authorities can use this information to implement adaptive traffic control systems, adjust signal timings, and optimize lane configurations to reduce congestion and improve traffic flow.

**Scenario 2:** Urban Development Planning City planners and urban developers can leverage TrafficTelligence predictions to plan new infrastructure projects effectively. By understanding future traffic volumes, they can design road networks, public transit systems, and commercial zones that are optimized for traffic efficiency and accessibility.

**Scenario 3:** Commuter Guidance and Navigation Individual commuters and navigation apps can benefit from TrafficTelligence's accurate traffic volume estimations. Commuters can plan their routes intelligently, avoiding congested areas and selecting optimal travel times based on predicted traffic conditions.

Navigation apps can provide real-time updates and alternative routes to improve overall travel experiences.

### **Technical Architecture:**



### **Project Flow:**

- User interacts with the UI (User Interface) to enter the input values.
- Entered input values are analyzed by the model which is integrated.
- Once the model analyses the input the prediction is showcased on the UI.

To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
  - Collect the dataset or Create the dataset
- Data Pre-processing.
  - Import the Libraries.
  - Importing the dataset.
  - Checking for Null Values.
  - Data Visualization.
  - Taking care of Missing Data.

- Feature Scaling.
- Splitting Data into Train and Test.
- Model Building
  - Import the model building Libraries
  - Initializing the model
  - Training and testing the model
  - Evaluation of Model
  - Save the Model
- Application Building
  - Create an HTML file
  - Build a Python Code
  - Run the App

## Data Pre-processing

- Data Pre-processing includes the following main tasks
  - Import the Libraries.
  - Importing the dataset.
  - Checking for Null Values.
  - Data Visualization.
  - Feature Scaling.
  - Splitting Data into Train and Test.
- **Import Necessary Libraries**
- It is important to import all the necessary libraries such as pandas, NumPy, matplotlib.
- **Numpy**- It is an open-source numerical Python library. It contains a multi-dimensional array and matrix data structures. It can be used to perform mathematical operations on arrays such as trigonometric, statistical, and algebraic routines.
- **Pandas**- It is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.
- **Seaborn**- Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- **Matplotlib**- Visualisation with python. It is a comprehensive library for creating static, animated, and interactive visualizations in Python
- **Sklearn** – which contains all the modules required for model building.

```
# importing the necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import sklearn as sk
from sklearn import linear_model
from sklearn import tree
from sklearn import ensemble
from sklearn import svm
import xgboost
```

### Importing the Dataset

- You might have your data in .csv files, .excel files
- Let's load a .csv data file into pandas using read\_csv() function. We will need to locate the directory of the CSV file at first (it's more efficient to keep the dataset in the same directory as your program).
- If your dataset is in some other location, Then
- **Data=pd.read\_csv(r"File\_location/datasetname.csv")**

```
# importing the data
data = pd.read_csv(r"G:\AI&ML\ML projects\Traffic_volume\traffic volume.csv")
```

**Note:** r stands for "raw" and will cause backslashes in the string to be interpreted as actual backslashes rather than special characters.

- If the dataset is in the same directory of your program, you can directly read it, without giving raw as r.
- Our Dataset weatherAus.csv contains the following Columns
- Holiday - working day or holiday
- Temp- temperature of the day
- Rain and snow – whether it is raining or snowing on that day or not
- Weather = describes the weather conditions of the day
- Date and time = represents the exact date and time of the day
- Traffic volume – output column

The output column to be predicted is Traffic volume. Based on the input variables we predict the volume of the traffic. The predicted output gives them a fair idea of the count of traffic

### Analyse the data

**head()** method is used to return top n (5 by default) rows of a DataFrame or series.

```
# displaying first 5 columns of the data
data.head()
```

	holiday	temp	rain	snow	weather	date	Time	traffic_volume
0	None	288.28	0.0	0.0	Clouds	02-10-2012	09:00:00	5545
1	None	289.36	0.0	0.0	Clouds	02-10-2012	10:00:00	4516
2	None	289.58	0.0	0.0	Clouds	02-10-2012	11:00:00	4767
3	None	290.13	0.0	0.0	Clouds	02-10-2012	12:00:00	5026
4	None	291.14	0.0	0.0	Clouds	02-10-2012	13:00:00	4918

**describe()** method computes a summary of statistics like count, mean, standard deviation, min, max, and quartile values.

```
data.describe()
```

The output is as shown below

```
# used to understand the descriptive analysis of the data
data.describe()
```

	temp	rain	snow	traffic_volume
count	48151.000000	48202.000000	48192.000000	48204.000000
mean	281.205351	0.334278	0.000222	3259.818355
std	13.343675	44.790062	0.008169	1986.860670
min	0.000000	0.000000	0.000000	0.000000
25%	272.160000	0.000000	0.000000	1193.000000
50%	282.460000	0.000000	0.000000	3380.000000
75%	291.810000	0.000000	0.000000	4933.000000
max	310.070000	9831.300000	0.510000	7280.000000

From the data, we infer that there are only decimal values and no categorical values.

**info()** gives information about the data - paste the image here.

```
# used to display the basic information of the data  
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 48204 entries, 0 to 48203  
Data columns (total 8 columns):  
#   Column          Non-Null Count  Dtype  
---  -  
0   holiday         48204 non-null  object  
1   temp            48151 non-null  float64  
2   rain            48202 non-null  float64  
3   snow            48192 non-null  float64  
4   weather         48155 non-null  object  
5   date            48204 non-null  object  
6   Time            48204 non-null  object  
7   traffic_volume  48204 non-null  int64  
dtypes: float64(3), int64(1), object(4)  
memory usage: 2.9+ MB
```

### Handling Missing Values

1. The Most important step in data pre-processing is dealing with missing data, the presence of missing data in the dataset can lead to low accuracy.

2. Check whether any null values are there or not. if it is present then the following can be done.

```
# used to display the null values of the data  
data.isnull().sum()
```

```
holiday      0  
temp         53  
rain         2  
snow        12  
weather      49  
date         0  
Time         0  
traffic_volume  0  
dtype: int64
```

There are missing values in the dataset, we will fill the missing values in the columns.

3. We are using mean and mode methods for filling the missing values

- Columns such as temp, rain, and snow are the numeric columns, when there is a numeric column you should fill the missing values with the mean/median method. so here we are using the mean method to fill the missing values.
- Weather column has a categorical data type, in such case missing data needs to be filled with the most repeated/ frequent value. Clouds are the most repeated value in the column, so imputing with clouds value.

```
data['temp'].fillna(data['temp'].mean(),inplace=True)
data['rain'].fillna(data['rain'].mean(),inplace=True)
data['snow'].fillna(data['snow'].mean(),inplace=True)

print(Counter(data['weather']))

Counter({'Clouds': 15144, 'Clear': 13383, 'Mist': 5942, 'Rain': 5665, 'Snow': 2875, 'Drizzle': 1818, 'H
'Fog': 912, nan: 49, 'Smoke': 20, 'Squall': 4})

data['weather'].fillna('Clouds',inplace=True)
```

## Prior Knowledge:

### 1. Basics of Traffic Engineering

- **Traffic Volume:** The number of vehicles that pass a point on a roadway over a specific time period.
- **Congestion:** Occurs when demand for road space exceeds supply, leading to slower speeds and delays.
- **Traffic Signals and Control:** Systems that manage traffic flow through intersections using timing algorithms.

---

### 2. Fundamentals of Machine Learning

- **Supervised Learning:** Algorithms learn from labeled historical data (e.g., past traffic volumes) to make future predictions.
- **Regression Models:** Used to predict continuous values like vehicle count or traffic speed.
- **Time Series Analysis:** A technique for analyzing data points collected or recorded at time intervals, crucial for predicting future traffic volumes.

---

### 3. External Influencing Factors

- **Weather Conditions:** Rain, snow, and fog can significantly alter traffic flow and volume.
  - **Events and Holidays:** Local events, school schedules, and holidays can cause temporary but predictable traffic pattern shifts.
  - **Urban Layout:** Road capacity, number of intersections, and type of area (residential, commercial, industrial) all affect traffic behavior.
- 

#### 4. Smart Cities & Intelligent Transportation Systems (ITS)

- **Adaptive Traffic Signals:** These adjust in real time based on current traffic conditions.
  - **Sensor Networks:** Cameras, inductive loops, and GPS data provide real-time inputs.
  - **Big Data & IoT Integration:** Allows collection, analysis, and action on massive volumes of data from urban infrastructure.
- 

#### 5. Applications of Traffic Prediction

- **Navigation Systems (e.g., Google Maps, Waze):** Use traffic prediction to suggest optimal routes.
- **Urban Planning:** Relies on traffic data to design efficient transportation infrastructure.
- **Government Policy and Investment:** Traffic data supports decisions on road expansions, public transport improvements, etc.
- **Features:**
  - Web-based UI for traffic volume prediction –
  - Form-based feature input via browser –
  - Machine Learning model built on traffic data - Flask-based backend for inference and display
  - One-Hot Encoding for categorical input processing
  - Prediction displayed in real-time via a simple interface



- Project Overview Frontend: - Built using HTML and basic styling
- User input collected through index.html form Backend:
- Python with Flask handles routing and predictions Model and encoder loaded using pickle
- app.py handles feature processing and ML inference.
- Architecture
- User Input (HTML Form) Flask Backend (app.py) Input Processing + One-Hot Encoding Model Prediction (model.pkl) Prediction Rendered on output.html
- Prerequisites
- Python , Flask ,NumPy,Pandas - scikit-learn - Pickle - Any IDE (e.g., VS Code)

### **Installation:**

1. Clone the repository
2. Create a virtual environment
3. Install dependencies using: pip install -r requirements.txt
4. Run the app: python app.py

### **Folder Structure**

/template/

index.html

/app.py

/traffic\_volume.csv /

/model.pkl

/encoder.pkl

### **User Interface :**

index.html collects user inputs

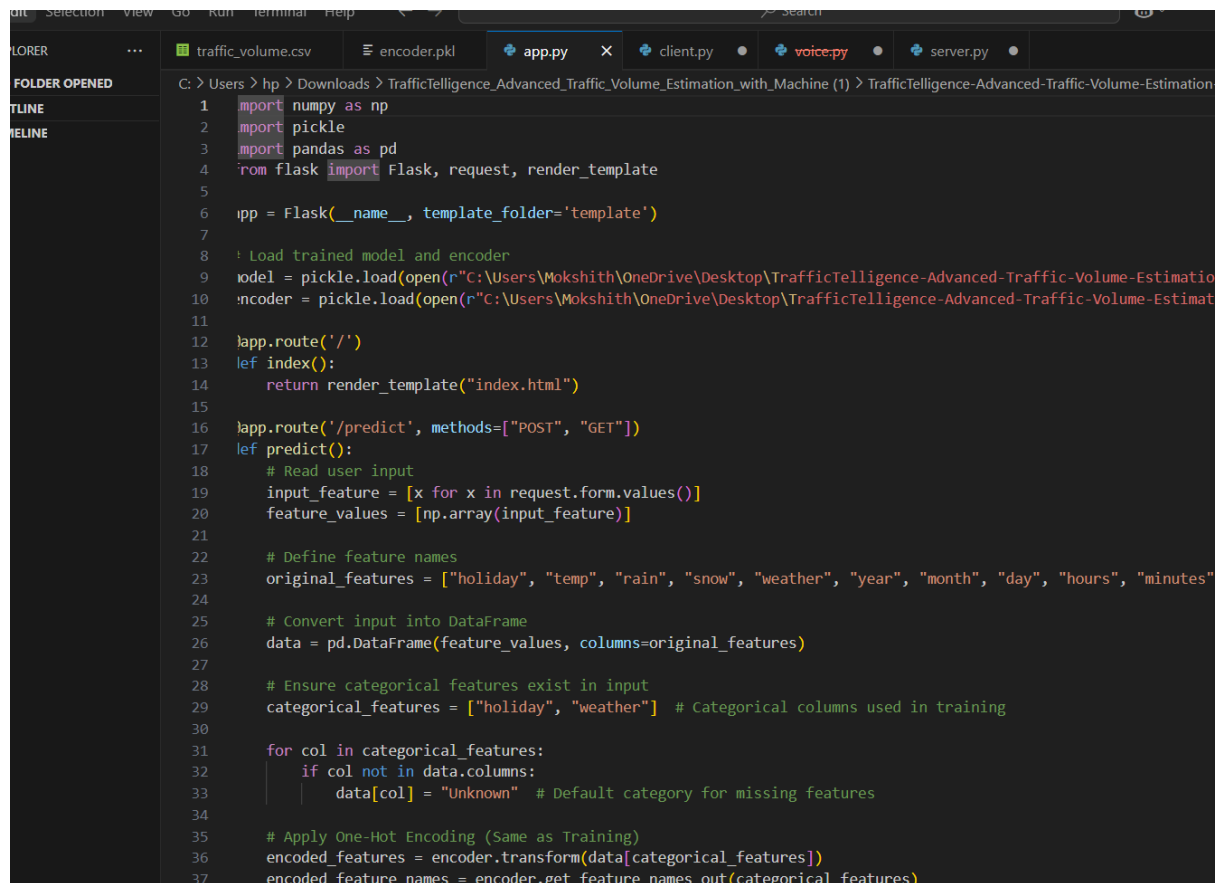
Input fields include: -

Holiday type - Weather conditions –

Temperature, Rain, Snow

Timestamp: Year, Month, Day, Hour, Minute, Second

Predict button triggers backend and displays result



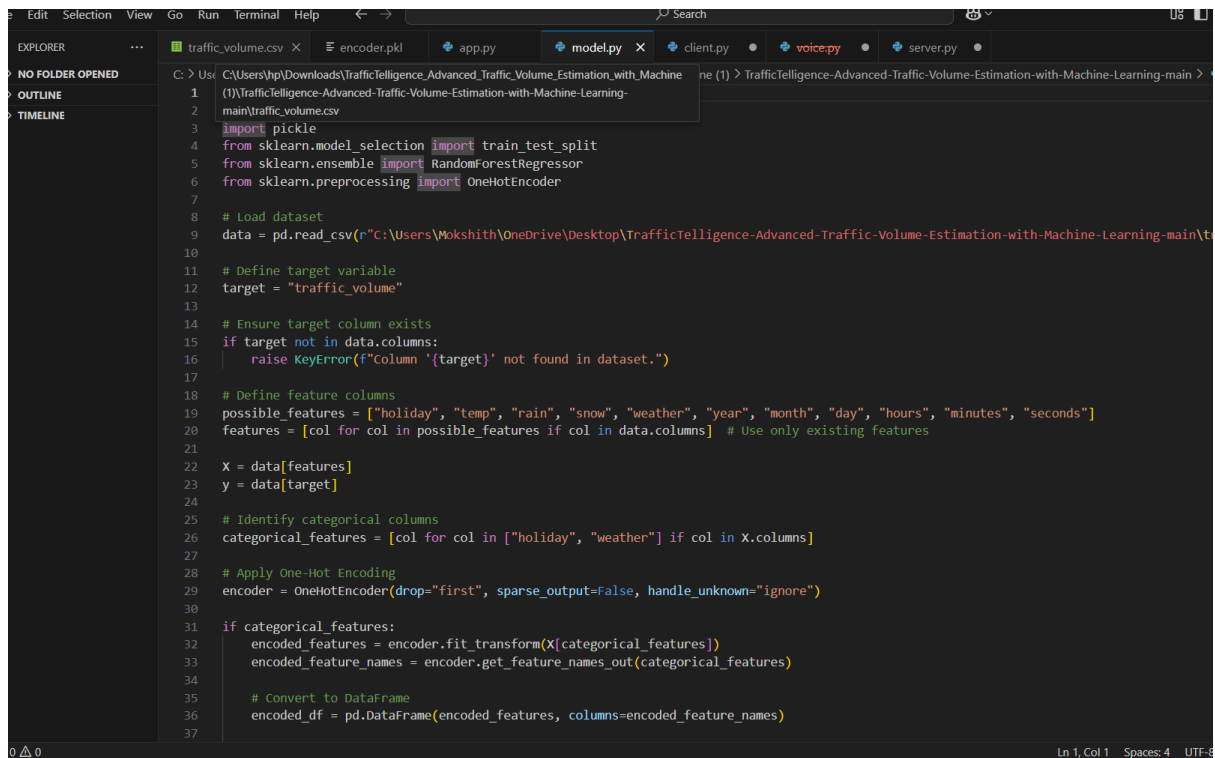
```
1 import numpy as np
2 import pickle
3 import pandas as pd
4 from flask import Flask, request, render_template
5
6 app = Flask(__name__, template_folder='template')
7
8 # Load trained model and encoder
9 model = pickle.load(open(r"C:\Users\Mokshith\OneDrive\Desktop\TrafficTelligence-Advanced-Traffic-Volume-Estimation\encoder.pkl", 'rb'))
10 encoder = pickle.load(open(r"C:\Users\Mokshith\OneDrive\Desktop\TrafficTelligence-Advanced-Traffic-Volume-Estimation\encoder.pkl", 'rb'))
11
12 app.route('/')
13 def index():
14     return render_template("index.html")
15
16 app.route('/predict', methods=["POST", "GET"])
17 def predict():
18     # Read user input
19     input_feature = [x for x in request.form.values()]
20     feature_values = [np.array(input_feature)]
21
22     # Define feature names
23     original_features = ["holiday", "temp", "rain", "snow", "weather", "year", "month", "day", "hours", "minutes"]
24
25     # Convert input into DataFrame
26     data = pd.DataFrame(feature_values, columns=original_features)
27
28     # Ensure categorical features exist in input
29     categorical_features = ["holiday", "weather"] # Categorical columns used in training
30
31     for col in categorical_features:
32         if col not in data.columns:
33             data[col] = "Unknown" # Default category for missing features
34
35     # Apply One-Hot Encoding (Same as Training)
36     encoded_features = encoder.transform(data[categorical_features])
37     encoded_feature_names = encoder.get_feature_names_out(categorical_features)
```

## Testing:

Manual input testing via browser

Model tested on validation set during training

Accuracy metric used for performance evaluation



```
C:\> Us/
1 C:\Users\hp\Downloads\TrafficTelligence_Advanced_Traffic_Volume_Estimation_with_Machine_Learning-main>
2 main\traffic_volume.csv
3 import pickle
4 from sklearn.model_selection import train_test_split
5 from sklearn.ensemble import RandomForestRegressor
6 from sklearn.preprocessing import OneHotEncoder
7
8 # Load dataset
9 data = pd.read_csv(r"C:\Users\Wokshith\OneDrive\Desktop\TrafficTelligence-Advanced-Traffic-Volume-Estimation-with-Machine-Learning-main\
10
11 # Define target variable
12 target = "traffic_volume"
13
14 # Ensure target column exists
15 if target not in data.columns:
16     raise KeyError(f"Column '{target}' not found in dataset.")
17
18 # Define feature columns
19 possible_features = ["holiday", "temp", "rain", "snow", "weather", "year", "month", "day", "hours", "minutes", "seconds"]
20 features = [col for col in possible_features if col in data.columns] # Use only existing features
21
22 X = data[features]
23 y = data[target]
24
25 # Identify categorical columns
26 categorical_features = [col for col in ["holiday", "weather"] if col in X.columns]
27
28 # Apply One-Hot Encoding
29 encoder = OneHotEncoder(drop="first", sparse_output=False, handle_unknown="ignore")
30
31 if categorical_features:
32     encoded_features = encoder.fit_transform(X[categorical_features])
33     encoded_feature_names = encoder.get_feature_names_out(categorical_features)
34
35 # Convert to DataFrame
36 encoded_df = pd.DataFrame(encoded_features, columns=encoded_feature_names)
37
```

## Known Issues:

Model may underperform on unseen weather/holiday types

UI lacks responsiveness on smaller screens

Missing input validation for form values.

## Future Enhancements:

Add user authentication and session logging

Enhance UI with Bootstrap or responsive design

Use a database to store predictions and analytics

Improve model using time-series approaches or deep learning

## Execution:

