# Traffic Volume Prediction System Documentation

## 1. Introduction

This document provides comprehensive documentation for the "Traffic Volume Prediction System" GitHub repository. The project aims to predict traffic volume based on various environmental and temporal factors using machine learning.

## 2. Project Purpose

The core purpose of this project is to develop a predictive model that can estimate traffic volume. This can be valuable for urban planning, traffic management, and real-time navigation systems. By considering factors such as weather conditions, temperature, and time-based attributes (holiday, day of the week, hour of the day), the system provides insights into expected traffic flows.

## 3. Repository Structure

The repository is structured as follows:

- `Flask/` : Contains the Flask web application for serving predictions.
- `app.py` : The main Flask application file, handling web routes, model loading, and prediction logic.
- `model.pkl` : The trained machine learning model (likely a regression model).
- `scale.pkl` : A scaler object used for feature scaling (e.g., StandardScaler).
- `encoder.pkl` : Encoders for categorical features (e.g., OneHotEncoder or LabelEncoder).
- `templates/` : HTML templates for the web interface.

    - `index.html`: The main landing page for the Flask application.
    - `predict.html`: The form for submitting prediction requests and displaying results.
- `model.pkl`: A copy of the trained machine learning model.
- `smartbridge_project.ipynb`: A Jupyter Notebook detailing the data preprocessing, model training, and evaluation steps.
- `traffic volume.csv`: The dataset used for training and evaluating the model.

## 4. Setup and Installation

To set up and run the project locally, follow these steps:

### 4.1. Clone the Repository

First, clone the GitHub repository to your local machine:

```
git clone https://github.com/ShaikMohammedAdhil/smart.git
cd smart
```

### 4.2. Create a Virtual Environment (Recommended)

It is highly recommended to create a virtual environment to manage project dependencies:

```
python3 -m venv venv
source venv/bin/activate  # On Windows, use `venv\Scripts\activate`
```

### 4.3. Install Dependencies

Navigate to the `Flask` directory and install the required Python packages. The project uses `Flask`, `numpy`, `pandas`, and `scikit-learn` (for `pickle` and `StandardScaler`/encoders). While a `requirements.txt` is not provided, based on the `app.py` and `smartbridge_project.ipynb` files, the dependencies are:

```
pip install Flask numpy pandas scikit-learn xgboost seaborn
```

### 4.4. Model and Scaler Files

Ensure that `model.pkl`, `scale.pkl`, and `encoder.pkl` are present in the `Flask` directory. These files are generated during the model training phase (as seen in `smartbridge_project.ipynb`) and are crucial for the Flask application to load the trained model and preprocessing objects.

## 5. Usage

### 5.1. Running the Flask Application

To start the web application, navigate to the `Flask` directory and run `app.py`:

```
cd Flask
python app.py
```

Once the application is running, you can access it through your web browser at `http://127.0.0.1:5000/` (or the address displayed in your console).

### 5.2. Making Predictions via Web Interface

1. Open your web browser and go to `http://127.0.0.1:5000/`.
2. Click on the "Make Prediction" link or navigate directly to `http://127.0.0.1:5000/predict`.
3. Fill in the form with the required details:
4. **Holiday**: Select from a dropdown list (e.g., None, Labor Day, Christmas).
5. **Temperature (°C)**: Enter the temperature.
6. **Rain (mm)**: Enter the amount of rain.
7. **Snow (mm)**: Enter the amount of snow.
8. **Weather Condition**: Select from a dropdown list (e.g., Clear, Clouds, Rain, Snow).
9. **Year, Month, Day, Hour, Minutes, Seconds**: Enter the date and time for the prediction.
10. Click "Predict Traffic Volume" to get the estimated traffic volume.

### 5.3. Making Predictions via API

The application also exposes a JSON API endpoint for predictions. You can send a POST request to `/api/predict` with the relevant data.

**Endpoint:** `http://127.0.0.1:5000/api/predict` **Method:** `POST` **Content-Type:** `application/json`

**Request Body Example:**

```json
{
    "holiday": "None",
    "temp": 288.28,
    "rain": 0.0,
    "snow": 0.0,
    "weather": "Clouds",
    "year": 2025,
    "month": 6,
    "day": 24,
    "hours": 10,
    "minutes": 30,
    "seconds": 0
}
```

**Response Body Example (JSON):**

```json
{
    "prediction": 5545,
    "status": "success"
}
```

## 6. Technical Details

### 6.1. Machine Learning Model

The project utilizes a machine learning model for traffic volume prediction. Based on the `smartbridge_project.ipynb` and `app.py`, the model is likely an `XGBoost Regressor` or a similar tree-based ensemble model, as `xgboost` is imported in the notebook and `model.pkl` is loaded. The model is trained on historical traffic data, weather conditions, and temporal features.

## 6.2. Data Preprocessing

The `smartbridge_project.ipynb` notebook outlines the data preprocessing steps, which include:

- **Loading Data**: Reading `traffic volume.csv`.
- **Feature Engineering**: Extracting temporal features such as year, month, day, hours, minutes, and seconds from the `date` and `Time` columns. The notebook also indicates that `date` and `Time` columns are dropped after processing.
- **Categorical Encoding**: Categorical features like `holiday` and `weather` are encoded. The `app.py` file explicitly shows a manual encoding fallback using dictionaries (`holiday_map`, `weather_map`) if the `encoder.pkl` is not loaded or fails, suggesting `LabelEncoder` or similar was used during training.
- **Feature Scaling**: Numerical features (e.g., `temp`, `rain`, `snow`, and the engineered temporal features) are scaled using a `StandardScaler` (indicated by `scale.pkl` and its usage in `app.py`). This ensures that features with larger values do not disproportionately influence the model.

## 6.3. Prediction Logic (`app.py`)

The `app.py` file implements the prediction logic:

- **`load_model_components()`**: This function loads the pre-trained `model.pkl`, `scale.pkl` (StandardScaler), and `encoder.pkl` (categorical encoders) using Python's `pickle` module. It includes error handling for missing files.
- **`encode_categorical_variable()`**: A helper function to safely encode categorical inputs (`holiday`, `weather`). It attempts to use the loaded encoders and falls back to a predefined mapping if encoders are not available or the value is unseen.
- **`create_prediction_dataframe()`**: This function takes raw input parameters from the web form or API request and transforms them into a Pandas DataFrame that matches the format and data types expected by the trained model. It applies the categorical encoding and ensures correct column order.
- **`make_prediction()`**: This function takes the prepared DataFrame, scales the numerical features using the loaded `scale` object, and then uses the loaded

`model` to make a prediction. It includes a fallback prediction mechanism if the model fails to load or predict, providing a basic estimate based on temperature, rain, snow, and time of day.

## 7. Future Enhancements

- **Improved Error Handling**: Enhance error messages and logging for better debugging and user feedback.
- **Comprehensive `requirements.txt`**: Provide a `requirements.txt` file for easier dependency management.
- **Dockerization**: Containerize the application using Docker for easier deployment and portability.
- **Database Integration**: Store historical predictions and user feedback in a database.
- **Advanced UI/UX**: Improve the web interface for a more interactive and user-friendly experience.
- **Model Retraining Pipeline**: Implement a pipeline for periodic model retraining with new data.

## 8. Conclusion

This Traffic Volume Prediction System provides a functional example of deploying a machine learning model as a web service. The clear separation of concerns between the data processing, model training (notebook), and serving (Flask app) makes it a good starting point for further development and integration into larger systems.