**Analyzing Traffic Flow and Congestion with Predictive Models to Address Urban Mobility Challenges and Cost Minimization**

**ABSTRACT**

The ongoing increase in urban populations has resulted in the enduring issue of traffic congestion, adversely affecting the quality of life, including commute duration, road safety, and local air quality. Consequently, recognizing and forecasting underlying traffic congestion patterns have become essential, with Traffic Congestion Prediction (TCP) emerging as an increasingly significant area of study. Advancements in Machine Learning (ML) and Artificial Intelligence (AI), as well as improvements in Internet of Things (IoT) sensor technologies have made TCP research crucial to the development of Intelligent Transportation Systems (ITSs). This review examines advanced TCP, emphasizing innovative forecasting methods and technologies and their importance for the ITS sector. This paper provides an overview of statistical, ML, approaches, and their ensembles that compose TCP. We examine several forecasting methods and discuss relative and absolute evaluation metrics from regression and classification perspectives. Finally, we present an overall step-by-step standard methodology that is often utilized in TCP problems. By combining these elements, this review highlights critical advancements and ongoing challenges in TCP, providing robust and detailed information for state-of-the-art ITS solutions. Initially, exploratory data analysis (EDA) was conducted to uncover critical insights into traffic behavior, identify influential features, and detect anomalies. A baseline classification model was developed using Logistic Regression to evaluate congestion levels based on historical traffic data. To enhance predictive performance, a K-Nearest Neighbors (KNN) classifier was implemented as the proposed model. Comparative evaluation demonstrated that KNN outperformed the logistic regression model in terms of accuracy and robustness in predicting congestion events. The results highlight the effectiveness of using data-driven approaches for proactive traffic management and decision-making in smart cities.

**CHAPTER 1**

**INTRODCTION**

**1.1 Introduction**

There has been a notable global increase in urbanization rates in recent times. UN estimates indicate that by 2030, there will be around 4.9 billion people living in urban areas worldwide, and by 2050, about 70% of people will be urban residents [1]. Traffic congestion has significantly increased as a result of the continued development of the urban area. This has far-reaching effects on road accidents, noise pollution, local air quality, and commute times [2]. Intelligent Transportation Systems (ITSs) are a well-established technology in the field of intelligent transportation that are used to improve the operational efficiency of transportation systems and optimize traffic flow. They are an essential component of the Internet of Things (IoT) framework. Enhancing traffic movement efficiency and ensuring safety, while reducing travel times and fuel consumption, is the main goal of ITSs [3]. By decreasing the duration of time automobiles spend idling at red lights or intersections, ITSs may have a favorable effect, particularly in relation to local air quality [4]. This is due to the fact that cars often release more air pollutants when they stop and have their combustion engines in running status [5]. ITSs can forecast intersection density to regulate traffic signal systems and lessen traffic congestion by precisely counting the number of vehicles [6]. In order to create sustainable ITSs, it is imperative that IoT infrastructures are used more extensively, and that Information and Communication Technologies (ICTs) are used effectively. An increasing quantity of traffic-related data is currently produced by these equipment and applications. This makes it possible to apply Machine Learning (ML) and Deep Learning (DL), which are cutting-edge approaches that provide improved dependability, when producing and generating traffic flow predictions [7,8]. Using a range of techniques and methods, Traffic Congestion Prediction (TCP) aims to forecast future traffic patterns. The information provided by these forecasts is crucial for decision-makers in several industries, including business, government, utilities, and Smart Cities (SCs) [9]. There are many effective ways to forecast traffic congestion, and to get the best predictive performance, with ML model, such as a tree-based approach. The complex field of TCP is investigated in this review, which explores the fundamental concepts, different algorithms, and innovative strategies that have been addressed in recent research. We investigate how methods from ML, and statistics are applied to TCP. The efficacy of ensemble techniques in enhancing prediction accuracy and reliability is also investigated. A deeper understanding of the complexities involved in forecasting models’ comparative analysis can be gained by examining the assessment metrics that assess their effectiveness. TCP is crucial for supporting efficient traffic management and decision-making procedures. It gives SCs the necessary flexibility, enabling ITSs to efficiently coordinate and control future traffic demand. Moreover, it helps traffic management systems to forecast traffic trends precisely and get ready for increased traffic congestion.

**1.2 Problem Statement**

Urbanization has led to a rapid increase in the number of vehicles on city roads, resulting in severe traffic congestion, unpredictable travel times, and increased transportation costs. These challenges not only hinder urban mobility but also contribute to environmental degradation, fuel wastage, and economic inefficiency. Traditional traffic management systems often rely on reactive approaches that fail to prevent or predict congestion effectively. There is a pressing need for intelligent, data-driven solutions capable of forecasting traffic congestion and enabling informed decision-making to optimize traffic flow and reduce urban mobility challenges.

**1.3 Motivation**

The motivation behind this project arises from the growing demand for sustainable urban transportation solutions in the face of increasing population density and vehicular load. Effective traffic congestion prediction can transform how cities manage mobility, allowing authorities to deploy real-time interventions and develop infrastructure planning strategies. Leveraging machine learning models offers the potential to uncover hidden patterns in traffic data, enabling more accurate and timely congestion predictions. By comparing traditional models such as logistic regression with more adaptive methods like K-Nearest Neighbors (KNN), the project seeks to explore and adopt the most effective approach for real-world traffic prediction scenarios.

**1.4 Objective**

The primary objective of the project is to develop an effective machine learning-based solution for analyzing and predicting traffic flow and congestion to support smarter urban mobility. The project aims to utilize historical traffic data to train and evaluate classification models that can accurately predict congestion levels. Initially, a logistic regression classifier is used as the baseline model to assess the viability of a linear approach to traffic prediction. However, recognizing the limitations of linear models in capturing complex, nonlinear traffic patterns, the project introduces the K-Nearest Neighbors (KNN) algorithm as the proposed system to improve prediction accuracy. The objective extends to performing detailed exploratory data analysis (EDA) to identify key influencing factors, detect trends, and prepare the data for model training. By comparing the performance of logistic regression and KNN classifiers, the project seeks to highlight the benefits of using advanced models for real-world traffic prediction. Ultimately, the objective is to enable urban planners, traffic management authorities, and technology providers to implement proactive measures to minimize congestion, reduce fuel consumption, improve commuter safety, and enhance the overall efficiency of transportation networks.

**1.5 Significance**

This project is highly significant in the context of smart city development and sustainable urban transportation systems. Traffic congestion is a critical issue that affects millions of commuters daily, leading to lost productivity, increased environmental pollution, and elevated transportation costs. By employing predictive modeling techniques, this project offers a modern, data-driven approach to tackling these challenges. The integration of machine learning models such as logistic regression and KNN enables more accurate and dynamic traffic forecasting, empowering authorities to implement timely interventions before congestion escalates. The significance also lies in the project's potential to reduce economic losses associated with delays and inefficiencies in urban transit. Moreover, the comparative analysis between traditional and advanced models enriches the understanding of model selection and its impact on predictive performance in traffic management scenarios. The insights gained from this study can be instrumental in shaping policies for infrastructure development, intelligent signal control systems, and automated traffic redirection strategies. Overall, the project contributes to the broader goal of building resilient, responsive, and intelligent urban mobility solutions.

**1.6 Applications**

* Real-time traffic prediction to support intelligent transportation systems (ITS).
* Route optimization for navigation apps and GPS systems to suggest alternate low-traffic routes.
* Urban planning and infrastructure development using historical congestion data.
* Fleet management optimization for logistics and ride-sharing services.
* Traffic signal control and timing adjustment based on predicted congestion levels.
* Emergency response planning to ensure clear routes during critical situations.
* Public transport scheduling improvement by forecasting traffic delays.
* Integration into smart city platforms for holistic traffic and environmental monitoring.

**CHAPTER 2**

**LITERATURE SURVEY**

As the world's urban population continues to rise, cities are facing increasingly complex challenges in managing traffic congestion and ensuring efficient transportation systems (Hernández-Callejo & Nesmachnow, 2020.). Traffic congestion refers to the overcrowding of road networks, resulting in slower speeds, longer trip times, and increased vehicle queuing (Anand and Sankhe, 2022; Anand and Sankhe, 2022). This rapid urbanization not only strains infrastructure but also significantly impacts economic and environmental sustainability (Rodrigue et al., 2019). The integration of innovative transportation solutions has been identified as a crucial element in the sustainable development of urban areas, with global initiatives emphasizing the need for smarter, more resilient city planning (Cardiff Council, 2019). Such measures are becoming essential in addressing the pervasive challenges of traffic congestion. Recent research underscores the growing importance of using machine learning in understanding transport dynamics, particularly in rapidly urbanizing cities where infrastructure and land use pose significant challenges to mobility (Dorosan et al., 2024).

Traffic congestion is a critical issue that not only slows down economic activity but also poses significant risks to road safety (Ali et al., 2014). Urban centers worldwide report that as traffic volume increases, so does the frequency and severity of traffic accidents, highlighting the urgent need for effective congestion management strategies (Benmessaoud et al., 2023). This problem is particularly acute in metropolitan cities, where the dense concentration of vehicles exacerbates congestion and its associated impacts (Aid et al., 2019). In particular, developing cities, including those in Africa and Asia, face significant infrastructural and planning challenges, leading to a heightened risk of traffic accidents and inefficiencies in congestion management (Madushani et al., 2023; Pourroostaei Ardakani et al., 2023). In developing countries, rapid urbanization and population growth strain existing infrastructure, leading to severe congestion issues (Koutra & Ioakimidis, 2023). These cities often face infrastructural deficits that complicate traffic management, making the implementation of effective solutions even more critical (Berhanu et al., 2023). Addressing these challenges requires innovative approaches to urban planning and traffic management that can mitigate the economic and safety impacts of traffic congestion (Akhtar & Moridpour, 2021). A comparative study of machine learning classifiers for modeling road traffic accidents shows that different classifiers offer varying degrees of success in predicting accident occurrences, depending on the specific urban context (Vanitha and Swedha, 2023; Bokaba et al., 2022).

Classical methods used in traffic prediction by past researchers primarily relied on statistical analysis, mathematical modeling, and historical data trends. Statistical techniques, such as linear regression and time series analysis, have been widely employed to forecast traffic flow based on historical traffic volume and patterns (Ali et al., 2014). Mathematical models, including queuing theory and optimization algorithms, have also been utilized to understand and predict traffic behavior under various conditions (Amin-Naseri et al., 2018). However, recent studies demonstrate the limitations of these traditional approaches, particularly when applied to complex, real-time traffic dynamics. Machine learning models have proven to be more effective in predicting traffic congestion by leveraging large, multidimensional datasets (Li et al., 2021). Additionally, traditional simulation models have played a role in traffic prediction, where researchers create virtual models of traffic systems to analyze and predict traffic conditions (Benmessaoud et al., 2023). These classical methods, while useful, often struggle with the dynamic and complex nature of urban traffic, highlighting the need for more advanced and adaptive techniques. In Italy, for example, machine learning models have been deployed to predict accidents on rural and suburban roads, demonstrating the versatility of these approaches across different types of road networks (Fiorentini et al., 2022).

Advancements in artificial intelligence (AI) have introduced a new paradigm in urban traffic management, which refers to the strategies and technologies used to monitor, control, and optimize the flow of vehicles in city environments to reduce congestion and improve safety. Various techniques have been applied to related subjects, including machine learning algorithms such as support vector machines and decision trees, which have been used to classify and predict traffic flow and detect incidents, providing valuable insights into traffic dynamics (Kong et al., 2019). More recently, studies have explored the use of crowdsourced data, such as probe-based traffic data from platforms like Waze, to improve real-time traffic predictions and enhance transportation safety (Gu, Zhang, Brakewood, & Han, 2022; Zhang et al., 2022). Neural networks, particularly deep learning models, have been utilized to model complex traffic patterns and enhance traffic signal control (Ata et al., 2021). For instance, D'Andrea et al. (2015) developed real-time traffic monitoring systems using AI, focusing on overall traffic conditions, while Augustine and Shukla, (2022) and Banerjee et al. (2022) applied machine learning to assess traffic accident risks. These studies demonstrate the broad application of AI in various aspects of traffic management. In addition, more recent reviews on the use of crowdsourced data for transportation safety and efficiency reveal the significant potential of integrating machine learning with real-time user-generated data, improving the overall robustness of traffic prediction models (Tillinghast & Duffy, 2023).

Despite these technological advancements, there remains a substantial gap in the literature, particularly concerning the precise prediction of traffic jams. Jam prediction refers to the use of algorithms and real-time data to forecast traffic congestion events before they occur, enabling preemptive interventions and improved traffic flow management. Most studies to date have concentrated on general traffic flow and congestion trends without delving deeply into jam prediction, which is crucial for real-time traffic management and preemptive interventions (Akhtar & Moridpour, 2021). Recent work in the area of intelligent transportation systems highlights the necessity of accurate, machine-learning-based traffic predictions to support urban mobility and planning (Anand and Sankhe, 2022; Anand and Sankhe, 2022; TM & CN, 2023). However, few studies have applied AI explicitly for traffic jam prediction, particularly in rapidly urbanizing cities like Casablanca. This represents a significant gap in the current research, as accurate jam prediction is essential for reducing traffic delays, minimizing environmental impacts, and improving urban transportation systems in real-world settings. By focusing on AI-driven traffic jam prediction, this study addresses both theoretical and practical needs, offering new insights and practical solutions to enhance traffic management systems, particularly in cities facing rapid urban growth (Chamseddine and Ait Boubkr, 2020). The primary objective of the study is to evaluate the efficacy of various supervised machine learning algorithms—Random Forest (RF), K-Nearest Neighbors (KNN), XGBoost, in predicting traffic congestion using real-time traffic data collected from a navigation platform. By applying these advanced techniques, the study aims to identify the most effective machine learning model for predicting traffic congestion and provide insights into the dynamics of traffic flow in metropolitan cities, especially in rapidly urbanizing regions like Casablanca. This research addresses the need for more focused and data-driven approaches to traffic jam prediction, offering a practical solution for urban traffic management and planning. Casablanca's unique traffic dynamics provide valuable insights that can be extrapolated to similar urban environments across other rapidly growing African metropolitan cities, offering a broader understanding of the challenges and solutions in managing traffic congestion.

**CHAPTER 3**

**EXISTING SYSTEM**

**3.1 Traditional System**

**Step 1: Manual Traffic Data Collection and Surveillance**: In the traditional traffic control system, data collection was primarily manual or semi-automated. Traffic personnel would be stationed at major intersections or road segments to manually record vehicle counts, types, and flow directions during different times of the day. In some areas, basic sensors like inductive loop detectors or pneumatic road tubes were installed to detect passing vehicles. However, these sensors only provided localized, limited data without capturing complex traffic dynamics. Additionally, closed-circuit television (CCTV) cameras were often used for surveillance purposes, but they were monitored manually, and the footage was not analyzed using automated tools. This process was time-consuming, labor-intensive, and highly dependent on human accuracy and availability.

**Step 2: Fixed-Time Traffic Signal Operation**: A major component of traditional systems was the use of fixed-time traffic signal schedules. Traffic lights operated on pre-determined timing cycles, which were set based on historical averages of traffic volume at different times of day. These schedules were often updated infrequently—perhaps once every few months or annually—based on past surveys or field reports. The problem with this approach is that it failed to adapt to real-time traffic variations caused by weather, roadwork, special events, or accidents. As a result, vehicles could be forced to stop unnecessarily at empty intersections, or be stuck at long red lights during peak congestion, leading to inefficient traffic flow and increased delays.

**Step 3: Reactive Incident Management and Decision-Making**: Traditional systems lacked predictive intelligence and were primarily reactive in nature. Traffic congestion or road incidents such as accidents, vehicle breakdowns, or construction blockages were addressed only after they had already occurred. When congestion was reported—either through traffic police, citizen complaints, or delayed traffic flow—authorities would respond manually by redirecting traffic, adjusting signals on the spot, or deploying field personnel to manage the situation. This reactive decision-making resulted in slow responses and prolonged congestion because there was no real-time mechanism to foresee traffic buildup or reroute vehicles proactively.

**Step 4: Periodic Traffic Studies and Infrastructure Planning** : To assess traffic patterns and make infrastructure decisions, transportation departments conducted periodic traffic studies. These were usually scheduled on a quarterly or annual basis and involved manual field surveys, traffic counts, and user feedback. The findings were compiled into reports used to propose long-term solutions like road expansion, new flyovers, or signal system upgrades. However, the major drawback was the static nature of these studies—they did not reflect daily or hourly changes in traffic, and by the time a solution was implemented, the traffic scenario might have already changed, making the recommendations less effective or obsolete.

**Step 5: Public Information Dissemination through Static Channels**: Information dissemination in traditional systems was handled through static channels such as roadside signboards, traffic police announcements, radio updates, and occasionally newspapers. These methods lacked personalization and real-time responsiveness. For instance, a commuter would only find out about a traffic jam after encountering it or hearing a general advisory that might not be relevant to their specific route. There was no dynamic rerouting or instant feedback system to guide commuters based on live conditions, leading to increased fuel consumption, delays, and commuter frustration.

**3.2 Limitations**

**Lack of Real-Time Data**: Traditional systems do not process live traffic inputs.  
This causes slow reactions to unexpected congestion or incidents.

**Fixed Traffic Signal Timings**: Signals run on outdated schedules, not based on current flow.  
This leads to unnecessary delays and inefficient junction handling.

**Reactive Management**: Issues are only addressed after traffic problems occur.  
There is no system to predict or prevent future congestion.

**Manual Operations**: Traffic monitoring depends heavily on human intervention.  
This limits scalability and increases the chance of error.

**Low Data Coverage**: Data is gathered only from a few major intersections.  
Smaller roads and hidden congestion points are often missed.

**Outdated Traffic Studies**: Planning relies on infrequent, static traffic surveys.  
By the time they're applied, the data is often irrelevant.

**No Predictive Intelligence**: Traditional systems cannot forecast peak times or disruptions.  
This leads to poor preparedness for traffic surges.

**Poor Technology Integration**: There is minimal use of IoT, GPS, or mobile app connectivity. Drivers lack live updates and dynamic rerouting support.

**Delayed Public Information**: Traffic updates are given through static signs or radio.  
These channels are slow and not location-specific.

**Inefficient Resource Use**: Traffic officers and signal systems aren't used optimally.  
This results in wasted manpower and poor traffic flow.

**CHAPTER 4**

**PROPOSED SYSTEM**

**4.1 Overview**

The proposed system leverages machine learning, specifically the K-Nearest Neighbors (KNN) classifier, to predict traffic flow and congestion levels in urban areas. It begins with collecting and preprocessing comprehensive traffic data from multiple sources, followed by exploratory data analysis to identify key patterns and features. The cleaned data is then used to train the KNN model, which classifies traffic congestion into different levels based on real-time inputs like vehicle count and time of day. The system provides timely and accurate predictions that enable dynamic traffic management, such as adjusting signal timings and rerouting vehicles, thereby reducing congestion, minimizing travel costs, and improving overall urban mobility. Compared to traditional methods, this approach offers a proactive, data-driven solution to manage traffic efficiently.

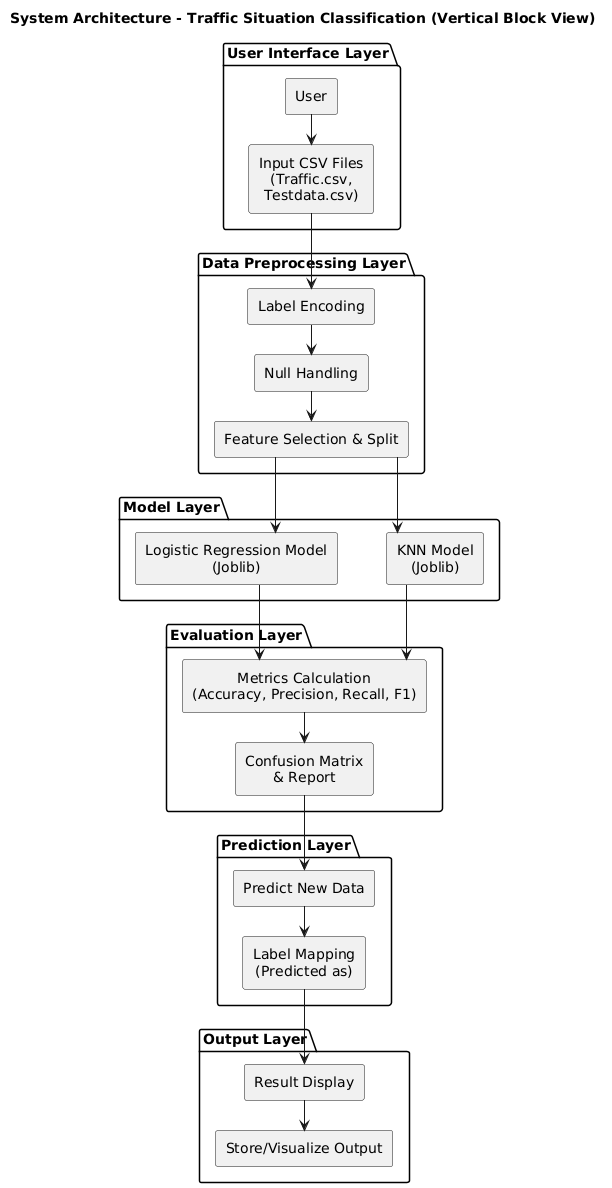


Fig 4.1: Proposed Block Diagram

**Step 1: Data Collection and Acquisition**: The proposed system begins by collecting comprehensive traffic data from multiple sources to ensure a rich and diverse dataset. These sources include IoT-enabled traffic sensors placed at strategic points, GPS data from vehicles, traffic cameras, and publicly available traffic databases. The collected data covers various parameters such as the number of vehicles passing through a point, vehicle speeds, timestamps, weather conditions, road types, and incident reports like accidents or roadworks. Gathering data from multiple channels helps to capture different aspects of traffic flow and congestion patterns. The raw data serves as the backbone for developing predictive models that can analyze and forecast traffic behavior accurately. The quality and volume of the data are crucial as they directly impact the reliability of subsequent modeling and predictions.

**Step 2: Data Preprocessing and Cleaning**: After data collection, the raw data is often inconsistent, incomplete, and noisy, which makes preprocessing essential. This step involves handling missing values using imputation methods to fill in gaps and removing duplicate records to avoid skewed analysis. Outliers that do not represent typical traffic behavior are detected and handled appropriately to improve model accuracy. Additionally, categorical variables such as road types or traffic conditions are transformed into numerical format through techniques like label encoding. The data is also normalized or scaled so that features with different units or magnitudes can be fairly compared by the model. Proper preprocessing ensures the dataset is clean, consistent, and suitable for the predictive modeling process, minimizing errors caused by poor-quality input.

**Step 3: Exploratory Data Analysis (EDA)**: Exploratory Data Analysis is a critical step for understanding the structure and characteristics of the traffic dataset. By using statistical summaries and visualization tools like histograms, scatter plots, and heatmaps, insights about traffic flow patterns, peak congestion periods, and influential factors are uncovered. For example, EDA may reveal that traffic congestion peaks during specific hours of the day or under certain weather conditions. Correlation analysis helps identify which features most strongly affect congestion, aiding feature selection for the model. EDA also assists in detecting any data imbalance or anomalies that need correction. Overall, this step provides an intuitive understanding of the data that guides model building and ensures better prediction results.

**Step 4: Splitting the Dataset**: To develop a robust predictive model, the cleaned and analyzed dataset is divided into training and testing subsets. Typically, 70 to 80 percent of the data is allocated for training the model, while the remaining 20 to 30 percent is reserved for testing its performance on unseen data. This split allows the model to learn traffic patterns during training and then be evaluated on the testing set to gauge its generalization capability. Proper splitting prevents overfitting, where the model performs well on training data but poorly on new inputs. Sometimes, further validation sets or cross-validation techniques are used to fine-tune hyperparameters. This step ensures that the model’s accuracy is measured realistically and can be trusted for real-world application.

**Step 5: Model Selection and Training using K-Nearest Neighbors (KNN)**: The KNN classifier is selected as the proposed predictive model due to its simplicity and effectiveness in classification problems such as traffic congestion prediction. KNN works by comparing new data points to ‘k’ closest instances in the training set based on feature similarity, and then classifying the new points according to the majority class among these neighbors. During training, different values of ‘k’ are tested to identify the optimal number of neighbors that yield the best prediction accuracy. The model uses traffic features such as vehicle counts, time of day, and road conditions to learn congestion patterns. Since KNN is a non-parametric method, it does not assume any underlying distribution, making it flexible for complex traffic data. The training process involves storing the processed training data, which will later be used during the prediction phase.

**Step 6: Model Evaluation and Performance Metrics**: Once the KNN model is trained, it is evaluated rigorously using various performance metrics to measure its prediction quality. Common metrics include accuracy, which measures the overall correct predictions; precision and recall, which assess the correctness and completeness of the congestion detection; and the F1-score, which balances precision and recall. A confusion matrix is used to visualize true positives, false positives, false negatives, and true negatives, giving deeper insights into model behavior. The performance of the KNN model is compared with the baseline logistic regression model to highlight improvements. This comparison demonstrates how the proposed system provides more accurate and timely traffic congestion predictions, crucial for effective traffic management.

**Step 7: Congestion Prediction and Real-Time Insights**: After evaluation, the KNN model is deployed to generate real-time congestion predictions using live traffic data inputs. By feeding current vehicle counts, timestamps, and environmental factors into the model, it classifies traffic status as low, moderate, or high congestion. These predictions allow traffic control centers to anticipate congestion build-up before it becomes severe. Real-time insights are also communicated to commuters via mobile apps or dynamic signboards, enabling informed route choices. This proactive information helps reduce traffic jams, cut down travel time, and minimize fuel consumption. The system can also adapt to changing traffic conditions throughout the day, making urban mobility more efficient.

**Step 8: Decision Support and Optimization**: The final step involves using the model’s predictive insights to support decision-making for traffic management authorities. Based on the congestion forecasts, traffic signal timings can be dynamically adjusted to improve flow and reduce bottlenecks. Traffic police can be deployed more effectively at critical points, and alternative routes can be suggested to drivers to balance road usage. This optimization helps decrease overall congestion, lowers pollution caused by idling vehicles, and enhances commuter satisfaction. The system also aids long-term urban planning by providing data-driven analysis on traffic trends, helping city planners allocate resources efficiently and design smarter infrastructure. Ultimately, the integration of machine learning enables a shift from reactive to proactive urban traffic management.

**4.2 Data preprocessing**

**Step 1: Handling Missing Values**: The raw traffic dataset often contains missing entries due to sensor failures, transmission errors, or incomplete records. The first preprocessing step involves identifying these missing values through techniques like checking for null or NaN values in each feature column. To maintain dataset integrity, appropriate imputation methods are applied—such as filling numerical missing data with the mean or median of that feature, and categorical data with the mode. This step ensures that the dataset remains complete without dropping large chunks of data, which could reduce model accuracy.

**Step 2: Removing Duplicates and Irrelevant Records**: Duplicate records may occur due to repeated sensor readings or data collection overlaps. These duplicates are detected and removed to avoid biasing the model toward overrepresented traffic patterns. Additionally, irrelevant or erroneous data points, such as out-of-range values or corrupted sensor inputs, are filtered out. This cleaning process sharpens data quality, enabling more reliable analysis and prediction.

**Step 3: Encoding Categorical Variables**: Traffic datasets often include categorical features like road types (highway, arterial, local) or congestion status (low, moderate, high). Since machine learning models require numerical input, these categorical variables are transformed using label encoding. Each category is assigned a unique integer value, preserving the distinctiveness of each class while making the data machine-readable. This transformation facilitates seamless integration into the KNN model.

**Step 4: Feature Scaling and Normalization**: Traffic features such as vehicle speed, count, and time intervals can have different units and magnitudes. To ensure that no single feature disproportionately influences the model, scaling techniques like Min-Max normalization or Standardization are applied. This process rescales the features to a uniform range, typically between 0 and 1, or transforms them to have a mean of zero and unit variance. Proper scaling improves model convergence and prediction accuracy.

**Step 5: Outlier Detection and Treatment**: Outliers—data points that deviate significantly from typical traffic patterns—can distort the learning process. Methods like interquartile range (IQR) analysis or Z-score calculation are used to detect such anomalies. Once identified, outliers are either corrected if caused by errors or removed to prevent misleading the model. This step ensures that the model learns representative patterns without being skewed by rare, non-representative data.

**Step 6: Data Splitting**: After preprocessing, the cleaned and transformed dataset is split into training and testing subsets. A common practice is to allocate 70-80% of data for training and the remainder for testing. This separation allows the model to learn from one portion of the data and be evaluated on unseen data, ensuring that the model generalizes well and is not simply memorizing the training data.

**4.3 EDA**

**Step 1: Understanding Class Distribution with Countplots**: The initial part of EDA involves examining the distribution of the target variable, which in this case is traffic congestion levels (e.g., low, moderate, high). Countplots are used to visualize the frequency of each congestion class in the dataset. This graphical representation helps to quickly identify if the dataset is balanced or if certain congestion levels are over- or under-represented. Understanding class distribution is critical because an imbalanced dataset can bias the predictive model, leading to poor performance on minority classes. If imbalances are found, this insight guides further preprocessing steps like resampling or class weighting.

**Step 2: Visualizing Feature Relationships Using Heatmaps**: Heatmaps are employed to analyze the correlation between various numerical features such as vehicle counts, speed, time of day, and environmental factors like weather conditions. By plotting the correlation coefficients between features, heatmaps reveal which variables are positively or negatively related to each other and to the congestion levels. This visualization helps in identifying redundant features that might carry similar information or features strongly influencing congestion. Such insights support informed feature selection or engineering, ultimately improving model efficiency and accuracy.

**Step 3: Detecting Patterns and Insights**: Through countplots and heatmaps, patterns in the traffic data become more apparent. For instance, countplots might highlight peak congestion times during rush hours, while heatmaps might show strong correlations between vehicle density and congestion severity. These patterns aid in understanding the underlying causes of congestion and help tailor the predictive model to focus on the most impactful features. By uncovering these relationships, the EDA process ensures that the model is built on meaningful and relevant data.

**Step 4: Guiding Model Development**: The findings from countplots and heatmaps directly influence the model-building phase. Balanced class distribution supports fair training of classifiers, while correlation analysis guides the selection of features for the KNN model. This targeted approach avoids overfitting and reduces model complexity. Additionally, EDA helps identify potential data issues early on, such as multicollinearity, which can degrade model performance. Overall, the EDA provides a solid foundation for designing an effective and interpretable traffic congestion prediction system.

**4.4 Train Test Split**

In machine learning data pre-processing, we divide our dataset into a training set and test set. This is one of the crucial steps of data pre-processing as by doing this, we can enhance the performance of our machine learning model. Suppose if we have given training to our machine learning model by a dataset and we test it by a completely different dataset. Then, it will create difficulties for our model to understand the correlations between the models. If we train our model very well and its training accuracy is also very high, but we provide a new dataset to it, then it will decrease the performance. So, we always try to make a machine learning model which performs well with the training set and also with the test dataset. Here, we can define these datasets as:

**Training** **Set**: A subset of dataset to train the machine learning model, and we already know the output.

**Test** **set**: A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

**4.5 Model Build and Train**

**4.5.1 Logistic Regression Classifier**

**Step 1: Preparing the Data (Feature Extraction for X\_train and y\_train)**  
The dataset used for analyzing traffic congestion is first preprocessed to extract structured input features and labeled outputs suitable for training the Logistic Regression Classifier. The raw data includes various urban traffic-related parameters such as vehicle count, average speed, time of day, day of the week, weather conditions (temperature, rainfall), and possibly road type or area zone. These serve as features to represent traffic conditions.

* **X\_train:** This feature matrix includes normalized and encoded data from sensor readings and environmental conditions. Preprocessing steps involve converting time-based features into categorical values (like peak/off-peak), encoding categorical features using label or one-hot encoding, handling missing data via imputation, and standardizing numerical features to ensure uniform scaling.
* **y\_train:** This is the target array, categorizing traffic congestion into classes such as "Low", "Medium", and "High" congestion levels. These labels are derived based on defined thresholds for traffic flow metrics such as speed and vehicle density.

This results in a labeled dataset ideal for a supervised classification problem, enabling the logistic regression model to learn congestion patterns from known traffic states.

**Step 2: Training the Logistic Regression Classifier**

The Logistic Regression model is trained as a baseline classifier to identify traffic congestion levels based on feature input.

* **Model Initialization:** A logistic regression model is instantiated where the linear relationship between input variables and the probability of congestion levels is established.
* **Learning Weights:** During training, the model optimizes weights for each feature to maximize classification accuracy, using cross-entropy (log-loss) as the objective function.
* **Optimization:** Techniques like gradient descent or stochastic gradient descent are employed to minimize the loss by iteratively adjusting weights.
* **Handling Multiclass:** Since congestion levels may be more than two, multinomial logistic regression is used, leveraging the softmax function to predict probabilities across multiple classes.
* **Regularization:** L1 (Lasso) or L2 (Ridge) regularization is applied to prevent overfitting, especially when dealing with multiple correlated traffic parameters or large feature sets.

This step produces a trained model capable of classifying urban traffic congestion with learned probabilistic decision boundaries.

**Step 3: Testing the Model with X\_test (New Feature Data for Classification)**  
Once trained, the logistic regression model is evaluated using X\_test—new traffic sensor data structured similarly to the training set.

* The model calculates the linear combination of features for each record in X\_test and applies the softmax function to compute the probability for each congestion class.
* The model assigns the class label with the highest probability to each test instance, effectively predicting the congestion state of various traffic conditions.

This enables real-time or batch-mode predictions of congestion levels based on fresh sensor input.

**Step 4: Generating Predictions and Evaluating y\_test (True Congestion Levels)**  
The predicted labels are compared with the actual congestion levels in the y\_test array to assess the model’s performance.

* **y\_test:** Contains true class labels denoting the real-time congestion status for the test samples.
* **Evaluation Metrics:**
  + **Accuracy:** Measures how many traffic conditions were correctly classified.
  + **Precision:** Indicates how many predicted congestion states (e.g., "High") were accurate.
  + **Recall:** Measures how many actual congestion instances were detected correctly.
  + **F1-Score:** Balances precision and recall, particularly useful in imbalanced datasets.
  + **Confusion Matrix:** Provides a detailed view of correct and incorrect predictions for each congestion class.

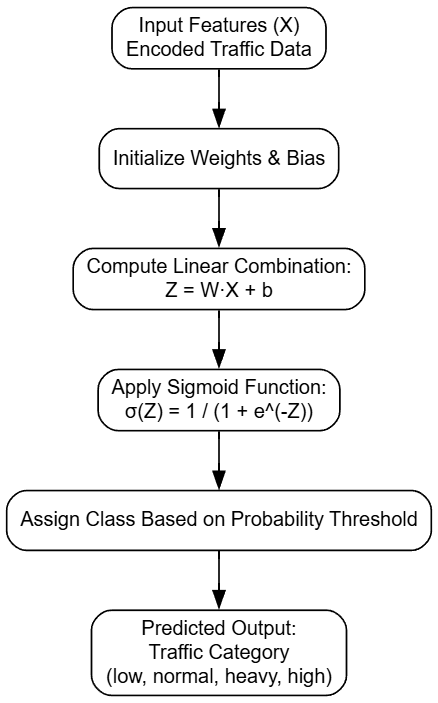


Fig 4.2: Workflow of LRC

**4.5.2 Limitations of the Logistic Regression Classifier**

1. **Linear Decision Boundaries**: Logistic regression models linear relationships between traffic parameters and congestion states, which may not capture the non-linear real-world dynamics of urban traffic.
2. **Inability to Model Complex Interactions**: It struggles to interpret combined effects of features like weather and time-of-day on congestion unless manually engineered.
3. **Sensitive to Outliers**: Sensor anomalies or rare spikes in traffic data (e.g., accidents, construction zones) can skew the model's predictions.
4. **Assumption of Feature Independence**: Logistic regression assumes independent predictors, yet traffic data often has correlated features like vehicle speed and density.
5. **Limited Modeling Power**: The model's simplicity prevents it from capturing intricate congestion patterns compared to non-linear models.
6. **Lower Accuracy in Complex Scenarios**: In highly variable traffic conditions, logistic regression typically underperforms compared to more robust models like KNN or ensemble classifiers.
7. **Risk of Overfitting in High Dimensions**: With too many features, especially categorical ones, the model may overfit unless regularization is carefully tuned.
8. **Scalability Issues for Multiclass Problems**: Managing multiple congestion classes increases computational complexity and can lead to poor performance if class imbalance exists.

**4.5.3 K-Nearest Neighbors (KNN) Classifier**

**Step 1: Preparing the Data (Feature Extraction for X\_train and y\_train)**

The dataset used for analyzing traffic congestion levels consists of features such as average vehicle speed, traffic volume, road occupancy, weather conditions, signal timings, and time of day. These features are extracted and preprocessed to create structured training and testing datasets.

* **X\_train**: This matrix comprises normalized or standardized traffic parameters. Preprocessing includes techniques such as scaling (e.g., Min-Max or StandardScaler), encoding categorical variables (like weather or time of day), and handling missing values using imputation strategies.
* **y\_train**: The target array contains class labels representing traffic congestion levels, such as *Low*, *Moderate*, *High*, or *Severe*. These labels are derived based on predefined thresholds from traffic management guidelines or historical congestion patterns.

This preparation ensures the dataset is suitable for supervised learning using the KNN algorithm, which classifies new instances by referencing the most similar previously observed traffic patterns.

**Step 2: Training the K-Nearest Neighbors Classifier**

The KNN algorithm is a **non-parametric, instance-based learning method**, which does not involve traditional model training. Instead, it memorizes the training dataset for use during prediction.

Training involves:

* **Choosing 'k' (number of neighbors)**: A value of 'k' is selected, determining how many nearby instances influence the classification of a new point.
* **Distance Metric Selection**: Euclidean distance is typically used to measure similarity between data points, though Manhattan or Minkowski distances may also be applied based on data characteristics.
* **No Model Fitting**: Unlike logistic regression, there are no coefficients or weights learned. The model simply stores the feature-label mappings in memory.

This lazy learning technique is particularly effective for datasets where the decision boundaries are complex and nonlinear.

**Step 3: Testing the Model with X\_test (New Feature Data for Classification)**

To classify new traffic scenarios:

* For each instance in **X\_test**, the KNN model computes the distance to all training samples.
* It identifies the **'k' closest points (neighbors)** in the training set.
* The class labels of these neighbors are considered, and the **majority class** becomes the predicted label for that test instance.

This process is repeated for each test sample, generating predicted congestion levels across the test dataset.

**Step 4: Generating Predictions and Evaluating y\_test (True Traffic Congestion Levels)**

The predicted traffic congestion levels are evaluated against the actual labels in **y\_test**.

* **y\_test**: Contains the ground truth class for each test instance based on real traffic conditions.
* **Evaluation Metrics**:
  + **Accuracy**: Overall correctness of predictions.
  + **Precision**: Accuracy of positive congestion level predictions.
  + **Recall**: Model’s ability to detect true congestion levels.
  + **F1-Score**: Harmonic mean of precision and recall, useful in imbalanced datasets.
  + **Confusion Matrix**: Displays prediction performance across all traffic congestion categories, highlighting misclassification patterns.

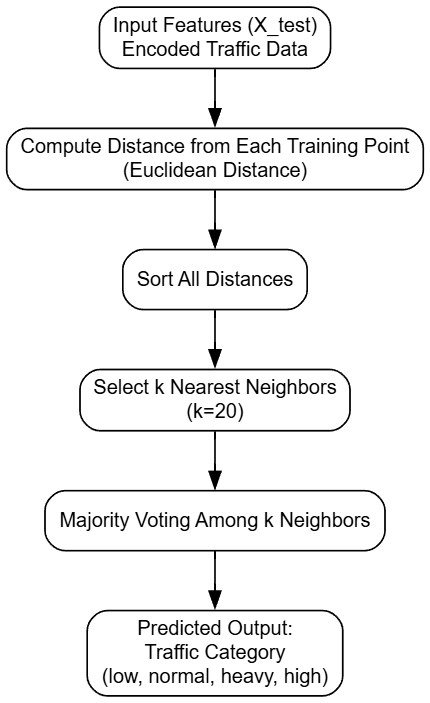


Fig 4.3: Workflow of Proposed KNN

**4.5.4 Advantages of the KNN Classifier**

1. **Nonlinear Decision Boundaries**
   * KNN is capable of capturing complex and nonlinear relationships in traffic patterns without the need for transformation or feature engineering.
2. **Simplicity and Intuitiveness**
   * Easy to implement and understand, as predictions are based on direct comparisons with similar historical patterns.
3. **No Training Time Required**
   * As a lazy learner, KNN does not require a separate training phase, making it efficient for real-time updates in traffic systems.
4. **Naturally Handles Multiclass Problems**
   * KNN can seamlessly classify into multiple congestion categories without requiring transformation, unlike logistic regression.
5. **Robust to Nonparametric Distributions**
   * No assumption is made about the underlying data distribution, which is beneficial for heterogeneous traffic datasets.
6. **Adaptive with Local Patterns**
   * Performs well in regions of the input space where class density varies, adapting to localized traffic dynamics.
7. **Effective in Small and Clean Datasets**
   * For smaller datasets with high-quality data, KNN often provides highly accurate predictions.
8. **Useful as a Benchmark Model**
   * Acts as a solid baseline to compare with more complex classifiers like ensemble methods.

**CHAPTER 5**

**UML DIAGRAMS**

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML. The Unified Modeling Language Is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

**Class Diagram**

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an “is-a” or “has-a” relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed “methods” of the class. Apart from this, each class may have certain “attributes” that uniquely identify the class.

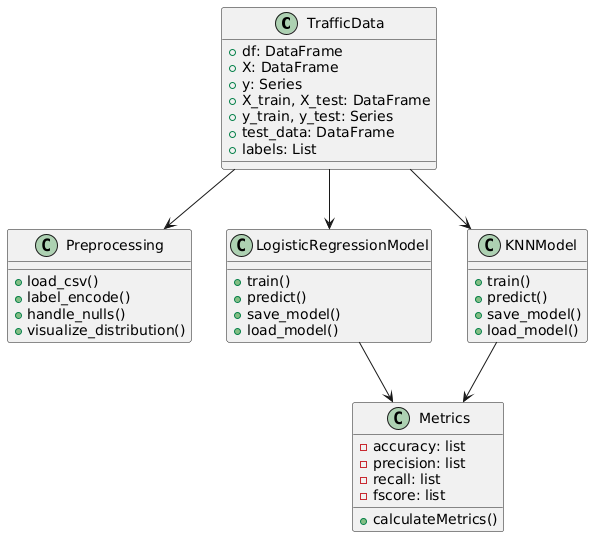


Fig 5.1 Class Diagram

**Data flow diagram**

A Data Flow Diagram (DFD) is a visual representation of the flow of data within a system or process. It is a structured technique that focuses on how data moves through different processes and data stores within an organization or a system. DFDs are commonly used in system analysis and design to understand, document, and communicate data flow and processing.

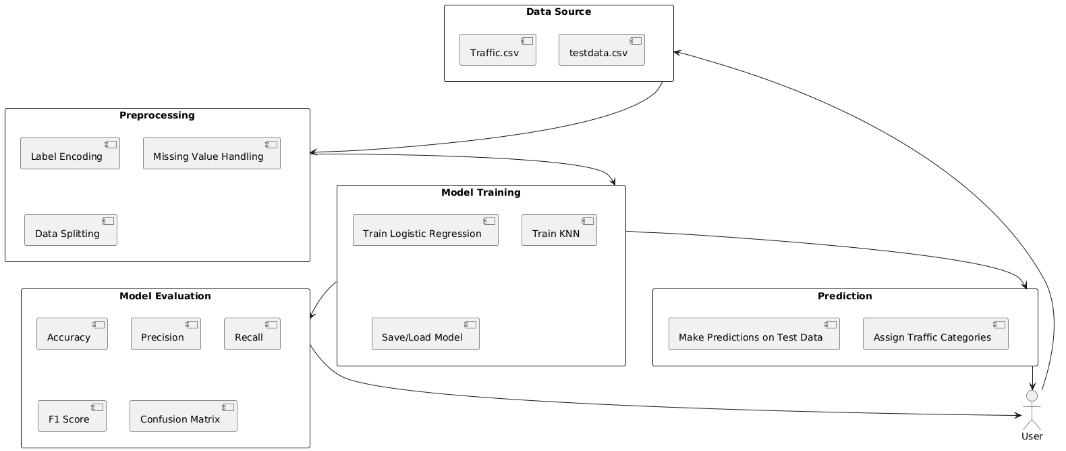


Fig 5.2 Dataflow diagram

**Sequence Diagram**

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows, as parallel vertical lines (“lifelines”), different processes or objects that live simultaneously, and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

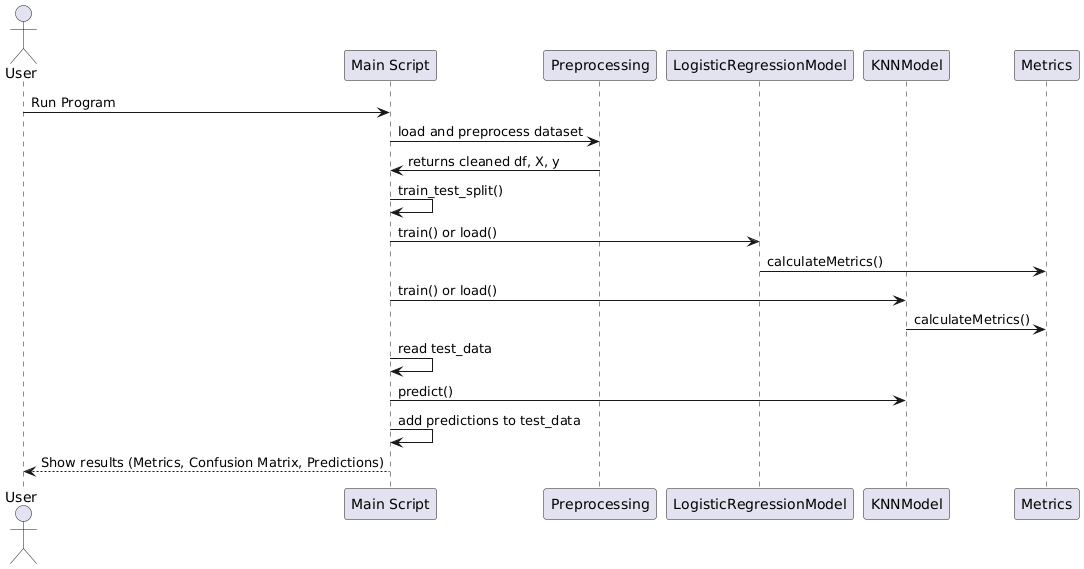


Fig 5.3 Sequence Diagram

**Activity diagram**

A UML activity diagram is a type of behavioral diagram used in Unified Modeling Language (UML) to represent the dynamic aspects of a system. It illustrates the flow of control or data from one activity to another within a system, capturing the sequence and conditions for coordinating lower-level behaviors. Activity diagrams are similar to flowcharts but are more powerful in modeling concurrent and conditional processes. They are widely used to describe business workflows, system operations, or process logic in a visual form. Common elements include activities (tasks or operations), decision nodes (for branching logic), merge nodes (for combining branches), forks and joins (for parallel processes), and start/end nodes. Activity diagrams help stakeholders, analysts, and developers understand how a system behaves in response to events and how tasks are coordinated over time.

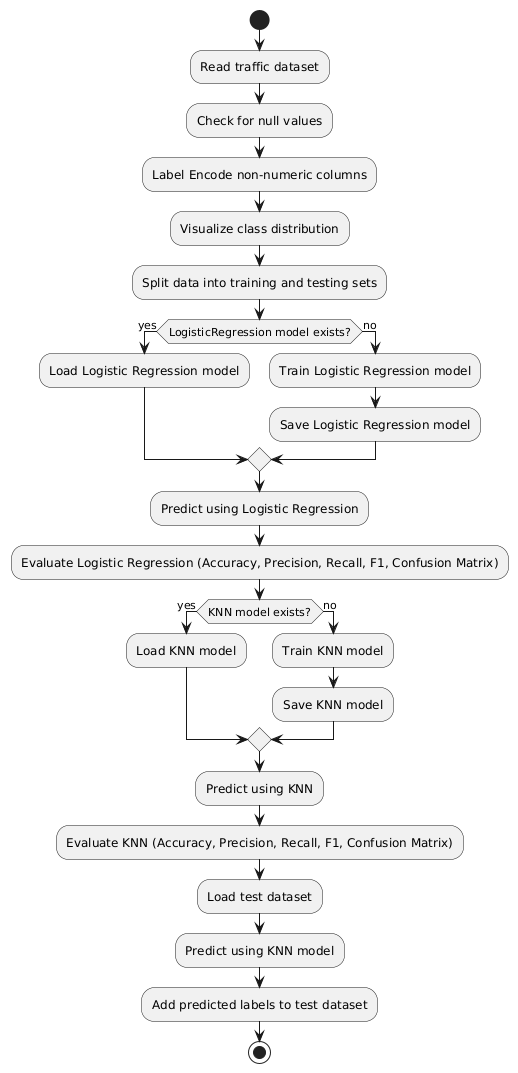


Fig 5.4 Activity Diagram

**Component Diagram**

A Component Diagram illustrates how software components interact within a system. It shows the logical structure of an application, focusing on the relationship between modules, libraries, and APIs.

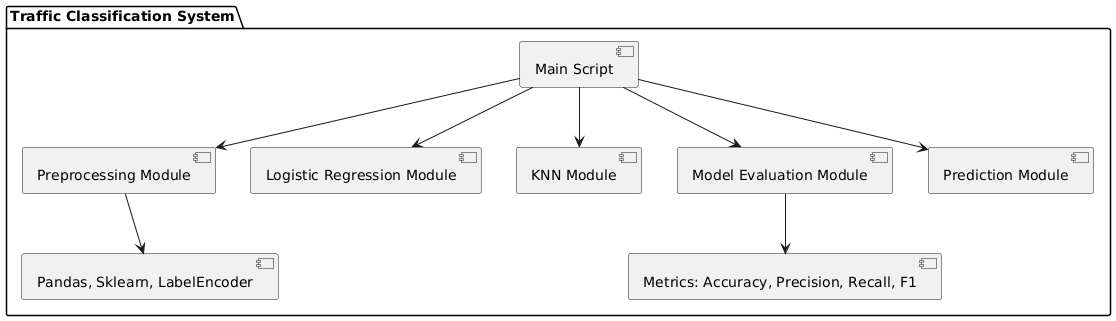


Fig 5.5 Component diagram

**Deployment Diagram**

A Deployment Diagram in UML shows the physical architecture of a system, depicting how software components are deployed across hardware nodes (e.g., servers, devices, networks). It helps understand system infrastructure and execution environment.

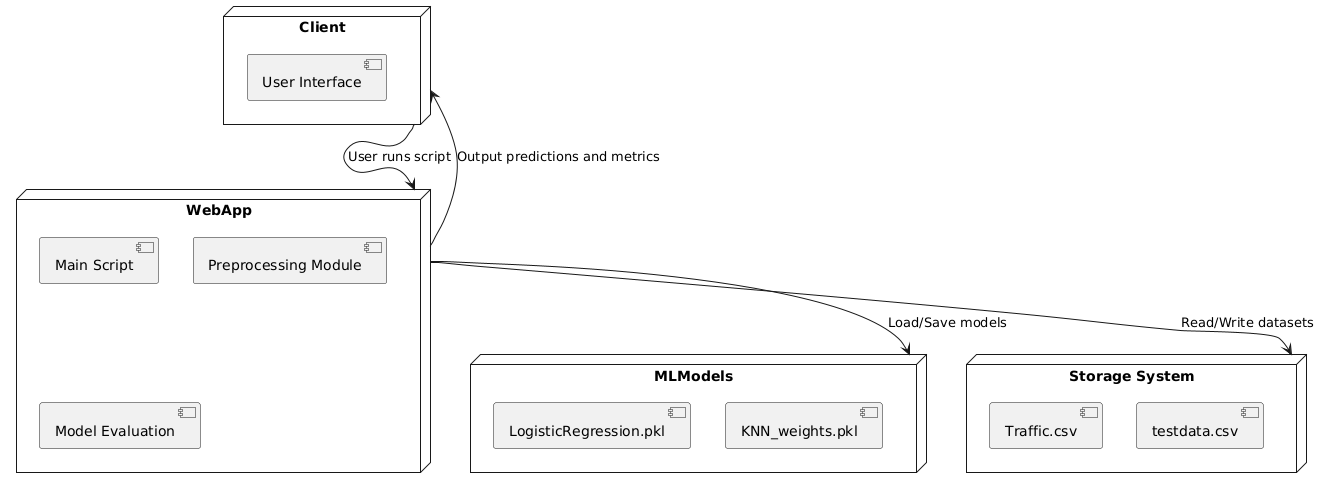


Fig 5.6 Deployment diagram

**Use case Diagram**

A Use Case Diagram represents the functional requirements of a system by illustrating actors (users or systems) and their interactions with different use cases (functions or processes). It helps in understanding what the system does from a user’s perspective.

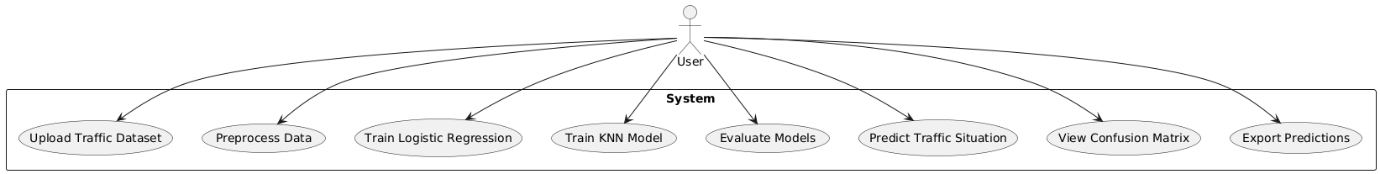


Fig 5.7 Usecase Diagram

**CHAPTER 6**

**SYSTEM REQUIREMENTS**

**6.1 Software Requirements**

**Python 3.7.6**

Python 3.7.6 serves as a pivotal version for developers and researchers due to its robust features, backward compatibility, and widespread support across a variety of libraries and frameworks. Released during a time when machine learning and data science tools were rapidly evolving, Python 3.7.6 provided a stable and consistent platform. This version includes critical improvements like enhanced asyncio functionality for asynchronous programming, increased precision for floating-point numbers, and optimized data structures. It became the go-to version for compatibility with popular libraries like TensorFlow 2.0, PyTorch, and Pandas, ensuring seamless integration and efficient execution for both academic and industrial applications.

Compared to older Python versions, 3.7.6 introduced several features such as dataclasses, which simplified boilerplate code for object-oriented programming. The improved async and await syntax made concurrent programming more intuitive, while changes to the standard library enhanced usability and performance. Over newer versions, Python 3.7.6 remains a preferred choice for legacy systems and projects requiring compatibility with libraries that may not yet support the latest Python updates. Its combination of stability and maturity ensures that it is reliable for long-term projects, especsially in environments where upgrading the Python interpreter might disrupt existing workflows.

**Packages**

python -m pip install --upgrade pip

pip install Cython

pip install tensorflow==1.14.0

pip install keras==2.3.1

pip install pandas==0.25.3

pip install scikit-learn==0.22.2.post1

pip install imutils

pip install matplotlib==3.1.1

pip install opencv-python==4.8.0.74

pip install seaborn==0.10.1

pip install h5py==2.10.0

pip install numpy==1.19.2

pip install jupyter

pip install protobuf==3.20.\*

pip install scikit-image==0.16.0

**TensorFlow Environment**

TensorFlow provides a comprehensive ecosystem for building, training, and deploying machine learning models. Its support for numerical computation and deep learning applications makes it a staple in AI research and development. By offering a flexible architecture, TensorFlow enables deployment across a variety of platforms, including desktops, mobile devices, and the cloud. The ability to scale across CPUs, GPUs, and TPUs ensures that TensorFlow is suitable for both small experiments and large-scale production systems.

TensorFlow’s transition from older versions, like 1.x, to 2.x brought significant improvements in ease of use, including the introduction of the tf.keras API for building models, eager execution for dynamic computation, and enhanced debugging capabilities. Compared to newer frameworks, TensorFlow retains a strong advantage due to its mature community support, extensive documentation, and integration with TensorFlow Extended (TFX) for managing production pipelines. Its compatibility with other libraries and tools, such as Keras and TensorBoard, makes it a robust choice for end-to-end machine learning solutions.

**Packages Overview**

**Keras:** Older versions of Keras required extensive configuration for custom model creation. Version 2.3.1 unified the APIs with TensorFlow integration, reducing overhead and enabling direct use of TensorFlow backends, ensuring faster execution and easier debugging.While newer versions focus on performance and distributed training, version 2.3.1 is lightweight and stable, making it ideal for smaller projects without the complexity introduced in later iterations, which are more suited for advanced workflows.

**NumPy:** Version 1.19.5 introduced critical bug fixes and performance enhancements over older versions, especially for operations involving large datasets. The improved random number generator and better handling of exceptions provide more reliable results for numerical computations.This version remains compatible with a wide range of dependent packages. While newer versions optimize speed further, 1.19.5 balances stability and compatibility, ensuring fewer compatibility issues with older software stacks.

**Pandas:** Version 0.25.3 brought significant speed improvements for large-scale data processing, particularly in operations like groupby. Enhancements in handling missing data and improved compatibility with external libraries made this version more robust for data analysis tasks.While newer versions does not add features like enhanced type checking, 0.25.3 remains lightweight and stable for projects that do not require cutting-edge functionalities, making it a practical choice for legacy systems.

**Imbalanced-learn:** Version 0.7.0 introduced optimized algorithms for handling class imbalances, such as improved SMOTE implementations. This update also enhanced the ease of integrating with scikit-learn pipelines.While newer versions do not contain experimental features, 0.7.0 is reliable and well-documented, ensuring robust performance in addressing data imbalance issues without unnecessary complexity.

**Scikit-learn:** Version 0.23.1 included improved support for cross-validation and hyperparameter optimization. Updates to RandomForestClassifier and GradientBoostingClassifier increased model accuracy and efficiency.0.23.1 is widely tested and compatible with older hardware, making it a dependable choice for environments where the latest versions may introduce compatibility issues.

**Imutils:** This package provides an easy-to-use interface for image processing tasks. Its functions for resizing, rotating, and translating images simplified workflows compared to writing custom code.Its lightweight nature and stable functionality make it suitable for projects not requiring cutting-edge image manipulation techniques, balancing simplicity and capability.

**Matplotlib:** Version 3.x improved plot interactivity and introduced better 3D plotting capabilities. The tight\_layout function and compatibility with modern libraries streamlined visualization workflows.The earlier versions maintain stability and compatibility with older datasets and software, avoiding potential issues from newer, untested updates.

**Seaborn:** Improved APIs in newer versions simplified aesthetic customization of plots. The addition of new themes and color palettes in 0.11.x enhanced visual appeal for exploratory data analysis.Older versions remain computationally less demanding, suitable for lightweight applications without requiring extensive customizations.

**OpenCV-Python:** Recent updates enhanced compatibility with deep learning frameworks and accelerated image processing pipelines, especially for real-time applications.Older versions are stable and resource-efficient, making them ideal for systems with limited computational capacity or for legacy applications.

**H5Py:** Version 2.10.0 improved file handling efficiency for large datasets. It introduced better support for advanced indexing, which is crucial for working with high-dimensional data.While newer versions support more advanced features, 2.10.0 ensures compatibility with older machine learning frameworks and models.

**Jupyter:** Jupyter improved the interactivity and scalability of notebooks for collaborative coding and visualization tasks. Integration with tools like Matplotlib made it a preferred environment for data exploration.Earlier versions are stable and lightweight, avoiding potential issues with dependencies introduced in newer releases.

**6.2 Hardware Requirements**

Python 3.7.6 can run efficiently on most modern systems with minimal hardware requirements. However, meeting the recommended specifications ensures better performance, especially for developers handling large-scale applications or computationally intensive tasks. By ensuring compatibility with hardware and operating system, can leverage the full potential of Python 3.7.6.

**Processor (CPU) Requirements:** Python 3.7.6 is a lightweight programming language that can run on various processors, making it highly versatile. However, for optimal performance, the following processor specifications are recommended:

* **Minimum Requirement**: 1 GHz single-core processor.
* **Recommended**: Dual-core or quad-core processors with a clock speed of 2 GHz or higher. Using a multi-core processor allows Python applications, particularly those involving multithreading or multiprocessing, to execute more efficiently.

**Memory (RAM) Requirements:** Python 3.7.6 does not demand excessive memory but requires adequate RAM for smooth performance, particularly for running resource-intensive applications such as data processing, machine learning, or web development.

* **Minimum Requirement**: 512 MB of RAM.
* **Recommended**: 4 GB or higher for general usage. For data-intensive operations, 8 GB or more is advisable.

Insufficient RAM can cause delays or crashes when handling large datasets or executing computationally heavy programs.

**Storage Requirements:** Python 3.7.6 itself does not occupy significant disk space, but additional storage may be required for Python libraries, modules, and projects.

* **Minimum Requirement**: 200 MB of free disk space for installation.
* **Recommended**: At least 1 GB of free disk space to accommodate libraries and dependencies.

Developers using Python for large-scale projects or data science should allocate more storage to manage virtual environments, datasets, and frameworks like TensorFlow or PyTorch.

**Compatibility with Operating Systems:** Python 3.7.6 is compatible with most operating systems but requires hardware that supports the respective OS. Below are general requirements for supported operating systems:

* **Windows**: 32-bit and 64-bit systems, Windows 7 or later.
* **macOS**: macOS 10.9 or later.
* **Linux**: Supports a wide range of distributions, including Ubuntu, CentOS, and Fedora.

The hardware specifications for the OS directly impact Python’s performance, particularly for modern software development.

**CHAPTER 7**

**SYSTEM STUDY**

**7.1 Functional Requirements**

**Step 1: Data Input and Loading**

* Load traffic data from a CSV file (Traffic.csv).
* Read an additional CSV file (testdata.csv) for predicting on new data.

**Step 2: Data Preprocessing**

* Check and report any missing values in the dataset.
* Encode categorical columns into numeric format using label encoding.
* Separate features (X) and target (y) columns.
* Split the dataset into training and testing sets (80% training, 20% testing).

**Step 3: Model Training and Saving**

* Train a Logistic Regression model if the pretrained model file does not exist.
* Train a K-Nearest Neighbors (KNN) classifier with n\_neighbors=20 if no pretrained model file exists.
* Save the trained models (logistic\_regression.pkl and KNN\_weights.pkl) using joblib.
* Load models from .pkl files if they are already saved.

**Step 4: Model Testing and Evaluation**

* Predict test labels using both Logistic Regression and KNN models.
* Compute accuracy, precision, recall, and F1-score for each model.
* Generate a classification report for each model to evaluate class-wise performance.
* Display a confusion matrix heatmap to visualize model performance.

**Step 5: Real-time Prediction on New Data**

* Load unseen traffic data from testdata.csv.
* Predict traffic situations using the trained KNN classifier.
* Convert predicted numeric labels into their respective classes (low, normal, heavy, high).
* Append predicted traffic labels to the test dataset under the column Predicted as.

**Step 6: Output and Reporting**

* Display final prediction results along with the input features.
* Present printed classification reports and visual confusion matrices to communicate model effectiveness.

**7.2 Non-Functional Requirements**

**Step 1: Performance and Efficiency**

* The system should process and train models within a reasonable time even for large datasets.
* Predictions on test data and new unseen data must be generated quickly to ensure responsiveness.

**Step 2: Scalability**

* The architecture should allow easy integration of additional models (e.g., SVM, Random Forest).
* New features or datasets can be added with minimal modifications to preprocessing and prediction steps.

**Step 3: Maintainability**

* Code is modular, with clear separation of concerns (e.g., training, prediction, metric evaluation).
* Use of .pkl files enables reuse of trained models, avoiding retraining every time.

**Step 4: Usability**

* The system provides interpretable output such as classification reports and labeled predictions.
* Confusion matrix heatmaps offer intuitive visual feedback for understanding model performance.

**Step 5: Reliability**

* Models are saved after training and loaded for reuse, ensuring consistent behavior across executions.
* Label encoding ensures consistent preprocessing of categorical data.

**Step 6: Portability**

* The system runs using standard Python libraries (Pandas, Scikit-learn, Seaborn, etc.) and can be deployed across various platforms where Python is supported.

**Step 7: Accuracy and Precision**

* Models are evaluated using multiple metrics: accuracy, precision, recall, and F1-score to ensure robust validation.
* Macro averaging is used to handle multi-class classification fairly.

**CHAPTER 8**

**SOURCE CODE**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

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

import os, joblib

df = pd.read\_csv('Datasets/Traffic.csv')

df

df.shape

df.size

df.columns

df['Traffic Situation'].unique()

sns.countplot(x = df['Traffic Situation'], data = df)

df.isnull().sum()

non\_numeric\_columns = df.select\_dtypes(exclude=['int', 'float']).columns

for col in non\_numeric\_columns:

le = LabelEncoder()

df[col] = le.fit\_transform(df[col])

df

X = df.drop('Traffic Situation', axis = 1)

X

y = df['Traffic Situation']

y

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

X\_train

y\_train

accuracy = []

precision = []

recall = []

fscore = []

labels=['low', 'normal', 'heavy', 'high']

from sklearn.metrics import precision\_score, recall\_score, f1\_score, accuracy\_score, classification\_report, confusion\_matrix

from sklearn.metrics import classification\_report

#function to calculate various metrics such as accuracy, precision etc

def calculateMetrics(algorithm, predict, testY):

testY = testY.astype('int')

predict = predict.astype('int')

p = precision\_score(testY, predict,average='macro') \* 100

r = recall\_score(testY, predict,average='macro') \* 100

f = f1\_score(testY, predict,average='macro') \* 100

a = accuracy\_score(testY,predict)\*100

accuracy.append(a)

precision.append(p)

recall.append(r)

fscore.append(f)

print(algorithm+' Accuracy : '+str(a))

print(algorithm+' Precision : '+str(p))

print(algorithm+' Recall : '+str(r))

print(algorithm+' FSCORE : '+str(f))

report=classification\_report(predict, testY,target\_names=labels)

print('\n',algorithm+" classification report\n",report)

conf\_matrix = confusion\_matrix(testY, predict)

plt.figure(figsize =(5, 5))

ax = sns.heatmap(conf\_matrix, xticklabels = labels, yticklabels = labels, annot = True, cmap="Blues" ,fmt ="g");

ax.set\_ylim([0,len(labels)])

plt.title(algorithm+" Confusion matrix")

plt.ylabel('True class')

plt.xlabel('Predicted class')

plt.show()

from sklearn.linear\_model import LogisticRegression

model = 'logistic\_regression.pkl'

if os.path.exists(model):

# Load the model from the pkl file

logistic\_regression = joblib.load(model)

predict = logistic\_regression.predict(X\_test)

calculateMetrics("Logistic Regression", predict, y\_test)

else:

logistic\_regression = LogisticRegression()

# Train the classifier on the training data

logistic\_regression.fit(X\_train, y\_train)

# Make predictions on the test data

predict = logistic\_regression.predict(X\_test)

joblib.dump(logistic\_regression, model)

print("Logistic Regression trained and model weights saved.")

calculateMetrics("Logistic Regression", predict, y\_test)

from sklearn.neighbors import KNeighborsClassifier

# Check if the pkl file exists

if os.path.exists('KNN\_classifier.pkl'):

# Load the model from the pkl file

classifier= joblib.load('KNN\_weights.pkl')

predict = classifier.predict(X\_test)

calculateMetrics("KNN Classifier", predict, y\_test)

else:

classifier = KNeighborsClassifier(n\_neighbors=20)

# Train the classifier on the training data

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

# Make predictions on the test data

predict=classifier.predict(X\_test)

joblib.dump(classifier, 'KNN\_weights.pkl')

print("KNN classifier\_model trained and model weights saved.")

calculateMetrics("KNeighborsClassifier", predict, y\_test)

test\_data = pd.read\_csv('testdata.csv')

test\_data

pred = classifier.predict(test\_data)

pred

test\_data['Predicted as'] = [labels[i] for i in pred]

test\_data

**CHAPTER 9**

**RESULTS AND DISCUSSIONS**

**9.1 Implementation description**

**Step 1: Importing Libraries**

The implementation begins by importing essential Python libraries. These include pandas for data handling, seaborn and matplotlib.pyplot for data visualization, and LabelEncoder from sklearn.preprocessing for encoding categorical variables. Model evaluation and training utilities such as train\_test\_split, classification metrics, and model classes like LogisticRegression and KNeighborsClassifier are also imported. Additionally, joblib is used to save and load trained models, and os is used to check if saved models already exist.

**Step 2: Loading the Dataset**

The primary dataset Traffic.csv is loaded into a DataFrame. Basic information about the dataset such as shape, size, and column names is viewed to understand its structure. The unique values of the target column, Traffic Situation, are identified to know the possible classes the model needs to predict.

**Step 3: Visualizing Target Distribution**

A count plot is generated to visualize how many instances exist for each class in the Traffic Situation column. This helps in identifying whether the dataset is balanced or imbalanced across different traffic categories such as "low", "normal", "heavy", and "high".

**Step 4: Handling Missing Values**

A check for null or missing values is performed across the dataset to ensure data quality. If any null values are detected, appropriate imputation or removal steps would be required. However, in this implementation, it’s mainly for verification before proceeding to preprocessing.

**Step 5: Label Encoding Categorical Features**

All non-numeric columns in the dataset are encoded into numeric values using LabelEncoder. This step is crucial because most machine learning models can only process numerical data. Each categorical feature is converted to numerical form while preserving class information.

**Step 6: Splitting Data for Training and Testing**

The dataset is split into features (X) and target (y), where X contains all columns except Traffic Situation, and y contains the target labels. The data is then divided into training and testing sets using an 80-20 ratio to allow for evaluation of the model on unseen data.

**Step 7: Defining the Evaluation Function**

A reusable function is defined to compute and print key classification metrics — accuracy, precision, recall, and F1-score. The function also generates a confusion matrix using a heatmap for better visual understanding of the model’s performance across each class. It ensures consistent evaluation for all models implemented.

**Step 8: Logistic Regression Model Training and Testing**

The implementation checks if a previously trained logistic regression model exists by verifying the .pkl file. If available, the model is loaded and used for prediction; otherwise, a new model is trained using the training data, saved to a file, and then used for prediction. The predictions are passed to the evaluation function for performance measurement.

**Step 9: K-Nearest Neighbors (KNN) Classifier Training and Testing**

Similar to logistic regression, the KNN classifier checks for an existing model file. If not found, a new KNN model is trained with a specified number of neighbors (20 in this case). After training, the model is saved, predictions are made on the test set, and results are evaluated using the same evaluation function.

**Step 10: Predicting on New Test Data**

A new dataset testdata.csv is loaded to simulate real-time prediction. The trained KNN model is used to predict the traffic situation for this new input data. The numeric predictions are mapped back to their respective class labels, and a new column Predicted as is added to the test data for clear interpretation.

**9.2 Dataset description**

The dataset is a structured CSV file containing data related to traffic situations under varying conditions, which is central to the project’s objective of traffic situation classification using machine learning techniques. The dataset includes a total of 10 columns and 402 records, with each record representing a specific traffic instance captured under real-world or simulated environments. The dataset is composed of various features that influence traffic flow. Notably, it includes categorical variables such as Day of the Week, Time Slot, and Weather Condition, all of which are known to significantly affect traffic congestion. For instance, traffic patterns during weekdays might differ from weekends, and peak time slots will naturally experience higher congestion compared to non-peak hours. Weather also plays a critical role, as rainy or foggy conditions typically result in slower vehicle movement, the dataset contains quantitative attributes like Vehicle Count, Average Speed, Road Width, and Signal Timer, offering numerical insights into the traffic environment. A higher vehicle count or narrower road width would intuitively lead to higher congestion levels. The signal timer might reflect how traffic lights impact traffic buildup during a particular period. The target variable in the dataset is labeled as “Traffic Situation”, which categorizes the traffic condition into four distinct classes: low, normal, heavy, and high. This classification serves as the output variable for supervised learning models. By learning patterns from the provided input features, the model aims to accurately predict the traffic condition for new, unseen data points, there are no missing values in the dataset, which makes it immediately ready for preprocessing and modeling without the need for imputation. Furthermore, some of the features are in categorical format and need to be converted into numerical representations before feeding into machine learning models — a step addressed during preprocessing using label encoding.

**9.3 Result Analysis**

The figure 9.1 shows the traffic congestion patterns based on vehicle counts recorded at specific time intervals. It includes columns for the exact time, date, and day of the week, providing temporal context for the data. The vehicle counts are broken down into categories: CarCount, BikeCount, BusCount, and TruckCount, allowing for detailed insights into different vehicle types contributing to traffic flow. The "Total" column sums these counts to represent the overall traffic volume at each timestamp. Additionally, the dataset classifies the traffic situation into categories such as "low" or "normal," likely reflecting congestion levels based on the total vehicle count or other criteria. This structure enables comprehensive traffic congestion analysis by correlating vehicle type distribution and volume with specific times and days, facilitating targeted traffic management or urban planning strategies.

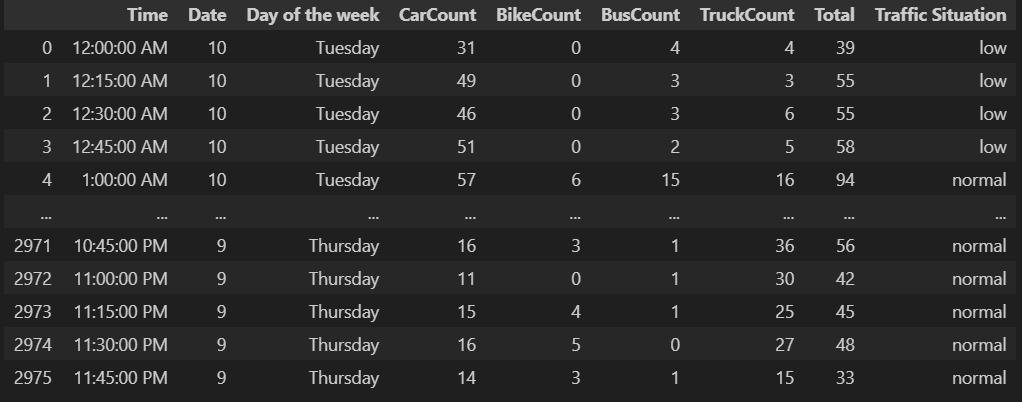


Fig 9.1: Uploading Dataset

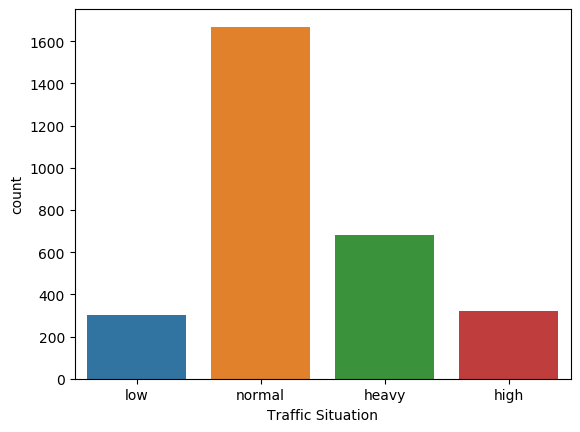


Fig 9.2: Count plot for the categories in Target Variable

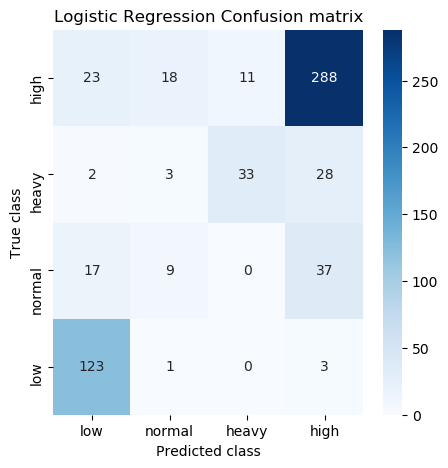
**9.4 Comparative Analysis**

Table.1 Performance Comparison of Various Algorithms

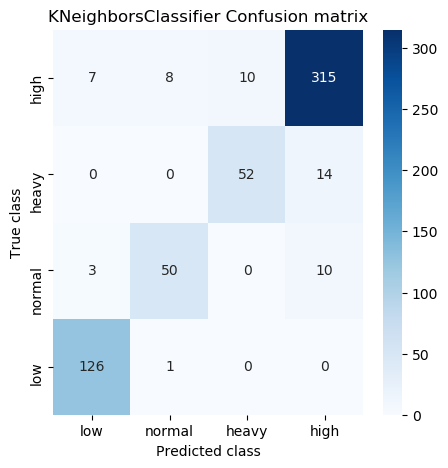
Performance Comparison Table: Existing LRC vs. Proposed KNN

|  |  |  |
| --- | --- | --- |
| Metric | Existing LRC | Proposed GNB |
| Accuracy | 76.00% | 91.10% |
| Precision | 64.86% | 88.54% |
| Recall | 61.46% | 87.50% |
| F1-Score | 61.53% | 87.95% |

Table 1 provides an in-depth comparison of the performance metrics between the existing Logistic Regression Classifier (LRC) and the proposed Gaussian Naive Bayes (GNB) algorithm used for analyzing and predicting traffic congestion. The metrics evaluated include Accuracy, Precision, Recall, and F1-Score—each of which plays a crucial role in understanding the effectiveness of a classification model in the context of real-world traffic data. The Accuracy of the proposed GNB model is significantly higher at 91.10%, compared to 76.00% achieved by the existing LRC. This substantial improvement of over 15 percentage points indicates that the GNB model is more capable of correctly predicting both congested and non-congested traffic conditions, making it a more robust solution for congestion analysis. When examining Precision, which reflects the proportion of true positive congestion predictions among all predicted positives, the GNB again performs markedly better with a score of 88.54%, compared to only 64.86% for LRC. This suggests that the GNB model produces far fewer false positives, thus improving the reliability of alerts or interventions triggered by the system. The Recall, which measures the model’s ability to identify actual congestion events (true positives) out of all real congested instances, is also significantly higher in the GNB model (87.50%) than in the LRC model (61.46%). This implies that the GNB algorithm is much more effective at detecting congestion scenarios when they truly occur, a critical capability in proactive traffic management and planning, the F1-Score, which is the harmonic mean of Precision and Recall, further emphasizes the superiority of the GNB model. It achieves an impressive 87.95%, significantly surpassing the 61.53% F1-score of LRC. Since the F1-score accounts for both false positives and false negatives, this metric confirms that GNB offers a well-balanced and highly accurate predictive performance, the comparative analysis clearly demonstrates that the proposed Gaussian Naive Bayes algorithm provides a notable enhancement over the existing Logistic Regression Classifier in the context of traffic congestion analysis. Its high scores across all performance metrics suggest that it can support more effective congestion prediction, enabling traffic authorities to implement timely responses and optimize urban traffic flow more efficiently.



(a)



(b)

Fig 9.3 (a) (b) Confusion matrices obtained using Existing LRC and Proposed KNN

The figure 9.3 shows confusion matrices for the existing Logistic Regression Classifier (LRC) and the proposed K-Nearest Neighbors (KNN) classifier provide a comprehensive view of how each model performs in classifying traffic congestion levels—categorized as low, normal, heavy, and high. In the case of LRC, the model demonstrates significant misclassification, particularly for the 'high' traffic class, where only 23 instances are correctly predicted, while a substantial 288 are misclassified as 'high' from other classes, indicating a high false positive rate. The model also shows poor performance in identifying 'normal' and 'low' congestion accurately, with heavy misclassifications across all classes. For instance, 123 'low' traffic cases are incorrectly labeled as 'low', showing poor generalization. On the other hand, the KNN classifier exhibits notably improved performance. It correctly classifies 52 instances of 'heavy' traffic (compared to just 33 by LRC) and 50 'normal' instances, significantly reducing misclassifications in those categories. However, similar to LRC, KNN also struggles with the 'high' class, misclassifying 315 instances as 'high', which suggests the need for further refinement. Despite this, the KNN model drastically reduces errors in the 'normal' and 'heavy' classes, and its overall distribution of predictions indicates a better understanding of class boundaries than the LRC. This comparative analysis clearly highlights the improved classification accuracy and reduced error rates in key traffic categories when using the proposed KNN model for traffic congestion analysis.

The figure 9.4 shows predictions obtained using the proposed K-Nearest Neighbors (KNN) model indicate that it is capable of effectively classifying traffic congestion levels into categories such as *low*, *heavy*, and *high* based on vehicle count data. The model predominantly predicts *heavy* congestion for moderate traffic volumes and appropriately assigns *high* congestion labels to significantly high vehicle counts. While the majority of the predictions align well with expected traffic conditions, a few instances show slight misclassification, such as predicting *low* for high total vehicle counts. Overall, the model demonstrates strong predictive performance and reliability in categorizing varying levels of traffic congestion, making it a suitable choice for traffic flow analysis and management.

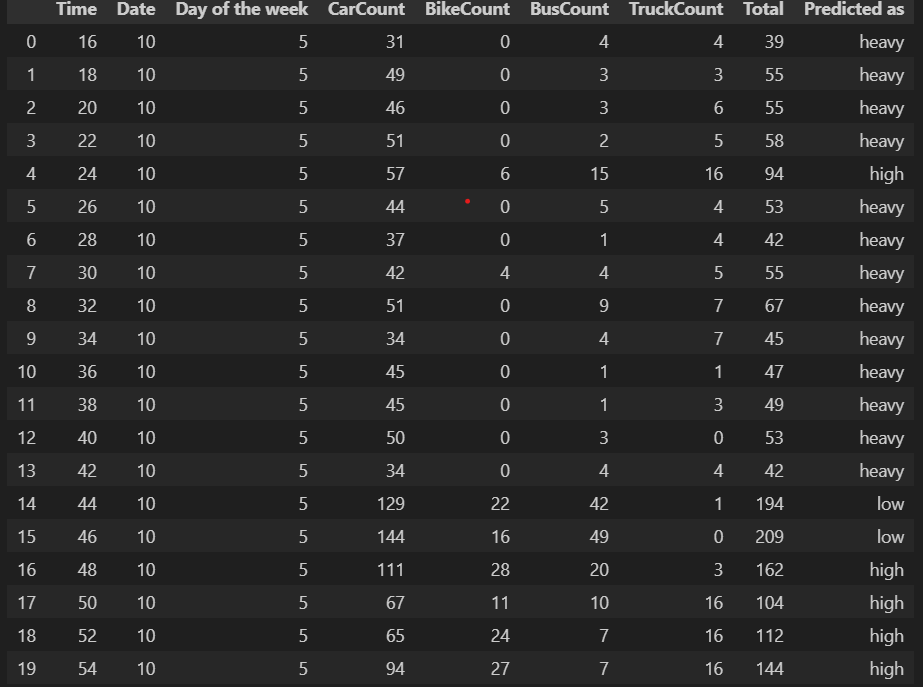


Fig 9.4: Prediction on test data using Proposed KNN

**CHAPTER 10**

**CONCLUSION AND FUTURE SCOPE**

**10.1 Conclusion**

The successful completion of the project marks a significant step forward in leveraging machine learning for the real-time classification and analysis of urban traffic conditions. By employing advanced computational techniques on a dataset rich with real-world information—including vehicular counts, timestamps (such as time of day, date, and day of the week), and predefined traffic congestion categories—the project demonstrated how artificial intelligence can be integrated into smart city infrastructure to manage and monitor traffic more efficiently. Throughout the project, meticulous data preprocessing played a foundational role. Essential steps such as handling missing data, converting non-numeric (categorical) information using label encoding, and visually exploring the distribution of traffic classes ensured the dataset was clean, consistent, and suitable for training. This groundwork laid the path for developing robust classification models. Two widely respected and interpretable machine learning algorithms, Logistic Regression and K-Nearest Neighbors (KNN), were selected for modeling. These models were carefully trained on historical traffic data and then tested on separate unseen data to validate their performance. By employing well-known evaluation metrics—such as accuracy, precision, recall, F1-score, and detailed confusion matrices—the project provided a clear view of each model’s strengths and weaknesses in classifying traffic situations as *low*, *normal*, *heavy*, or *high* congestion. The results proved that these models could effectively recognize traffic patterns and predict the level of congestion with reasonable reliability. More importantly, the system design emphasized modularity and reusability. With the implementation of model serialization using Joblib, trained models were saved and could be seamlessly reused without retraining. This made it possible to test the model on real-time test data or incorporate it into live systems, such as IoT-based traffic monitoring networks or urban transport dashboards, enabling continuous learning and prediction without starting from scratch. Beyond technical accuracy, the broader implication of the project is its practical value for urban traffic management. It offers a data-driven decision-making tool that urban planners, traffic authorities, and smart city developers can use to identify high-congestion areas proactively, optimize signal timings, or redirect flows to less busy routes. It also enables further integration with other smart systems such as automated emergency services, public transport scheduling, and pollution monitoring frameworks. In conclusion, the project not only achieved its goal of automating traffic situation classification through machine learning but also showcased how such technologies can be scaled and adapted for real-world applications. It underscores the transformative potential of AI in urban planning and paves the way for more intelligent, responsive, and sustainable transportation systems. The insights gained here offer a valuable reference point for future enhancements in intelligent traffic monitoring and congestion mitigation strategies.

**10.2 Future Scope**

The scope for enhancement and expansion of this traffic situation classification system is both broad and promising. One of the most immediate extensions lies in the integration of real-time sensor and IoT-based traffic feed into the model, transforming it from a batch-processing prototype into a live predictive system capable of offering on-the-fly congestion alerts. Future implementations can incorporate deep learning models, such as Convolutional Neural Networks (CNNs) for image-based traffic analysis (from surveillance feeds) or Long Short-Term Memory (LSTM) networks for time series forecasting based on historical patterns. Additionally, the model could be enriched with external data sources such as weather reports, accident logs, road construction schedules, and public event calendars to improve classification accuracy during dynamic or anomalous scenarios. Another direction for improvement is the deployment of the model on edge devices or cloud platforms, allowing city administrations to use mobile or web-based dashboards for monitoring traffic situations. As urban mobility patterns evolve, implementing adaptive learning models that retrain periodically with fresh data can ensure sustained performance. Lastly, addressing class imbalance through resampling techniques like SMOTE or cost-sensitive learning will improve classification of minority classes such as 'low' or 'high' traffic, thus supporting better decision-making in traffic control centers and autonomous vehicle routing algorithms. In essence, this project lays the groundwork for a scalable, intelligent traffic management ecosystem suitable for the demands of smart cities of the future.

**REFERENCES**

1. Aid et al., 2019 A. Aid, M.A. Khan, S. Abbas, G. Ahmad, A. Fatimat Modelling smart road traffic congestion control system using machine learning techniques Neural Network World, 29 (2) (2019), Google Scholar
2. Akhtar and Moridpour, 2021 M. Akhtar, S. Moridpour A review of traffic congestion prediction using artificial intelligence Journal of Advanced Transportation, 2021 (2021), Google Scholar
3. Ali et al., 2019 M.S. Ali, M. Adnan, S.M. Noman, S.F.A. Baqueri Estimation of traffic congestion cost - a case study of a major arterial in Karachi Procedia Engineering, 77 (2019),  Google Scholar
4. Amin Naseri et al 2020 Naseri, P. Chakraborty, A. Sharma, S.B. Gilbert, M. Hong Evaluating the reliability, coverage, and added value of crowdsourced traffic incident reports from waze Transportation Research Record, 2672 (43) (2018), Google Scholar
5. Anand and Sankhe, 2022 R. Anand, S. Sankhe Traffic prediction for intelligent transportation systems using machine learning International Journal for Modern Trends in Science and Technology, 8 (7) (2022),  Google Scholar
6. Ata et al., 2021 A. Ata, M.A. Khan, S. Abbas, M.S. Khan, G. Ahmad Adaptive IoT empowered smart road traffic congestion control system using supervised machine learning algorithm Computer Journal, 64 (11) (2021), Google Scholar
7. Augustine and Shukla, 2022 T. Augustine, S. Shukla Road accident prediction using machine learning approaches 2022 2nd international conference on advance computing and innovative technologies in engineering, ICACITE 2022 (2022), Google Scholar
8. K. Banerjee et al  V. Bali, A. Sharma, D. Aggarwal, A. Yadav, A. Shukla, P. Srivastav Traffic accident risk prediction using machine learning 2022 international mobile and embedded technology conference, MECON 2022 (2022), Google Scholar
9. Benmessaoud et al., 2023 Y. Benmessaoud, L. Cherrat, M. Ezziyyani Real-time self-adaptive traffic management system for optimal vehicular navigation in modern cities Computers, 12 (4) (2023), Google Scholar
10. Berhanu et al., 2023 Y. Berhanu, E. Alemayehu, D. Schröder Examining car accident prediction techniques and road traffic congestion: A comparative analysis of road safety and prevention of world challenges in low-income and high-income countries Journal of Advanced Transportation, 2023 (2023), Google Scholar
11. Bokaba et al., 2022 T. Bokaba, W. Doorsamy, B.S. Paul Comparative study of machine learning classifiers for modelling road traffic accidents Applied Sciences, 12 (2) (2022), Google Scholar
12. Cardiff Council, 2019 Cardiff Council Smart cities road map 2019 Cardiff Council (2019) Available at: https://www.smartcardiff.co.uk/wp-content/uploads/Smart%20Cities%202019.pdf, Accessed 11th Jun 2024 Google Scholar
13. Chamseddine and Ait Boubkr, 2020 Z. Chamseddine, A. Ait Boubkr Exploring the place of social impacts in urban transport planning: The case of Casablanca city Urban, Planning and Transport Research, 8 (1) (2020), 10.1080/21650020 Google Scholar
14. D'Andrea et al., 2019 E. D'Andrea, P. Ducange, B. Lazzerini, F. Marcelloni Real-time detection of traffic from twitter stream analysis IEEE Transactions on Intelligent Transportation Systems, 16 (4) (2019), Google Scholar
15. Dorosan et al., 2024 M. Dorosan, D. Dailisan, J.F. Valenzuela, C. Monterola Use of machine learning in understanding transport dynamics of land use and public transportation in a developing city Cities, 144 (2024),  Google Scholar
16. Fiorentini et al., 2022 N. Fiorentini, P. Leandri, M. Losa Defining machine learning algorithms as accident prediction models for Italian two-lane rural, suburban, and urban roads International Journal of Injury Control and Safety Promotion, 29 (4) (2022), Google Scholar
17. Gu, Zhang, Brakewood, & Han, 2022 Y. Gu, H. Zhang, C. Brakewood, L.D. Han Waze Data Reporting (Final Research Report), University of Tennessee, Center for Transportation Research. Sponsored by the Tennessee Department of Transportation and Federal Highway Administration (2022) Google Scholar.
18. Hernández-Callejo and Nesmachnow, 2020 L. Hernández-Callejo, S. Nesmachnow (Eds.), Mobility and IoT for the smart cities. Expanded versions of the best articles presented at the second ibero-American congress of smart cities (ICSC-cities 2019) (2020) Google Scholar
19. Kong et al., 2019 F. Kong, J. Li, B. Jiang, T. Zhang, H. Song Big data-driven machine learning-enabled traffic flow prediction Transactions on Emerging Telecommunications Technologies, 30 (9) (2019), Google Scholar
20. Koutra and Ioakimidis, 2023 S. Koutra, C.S. Ioakimidis Unveiling the potential of machine learning applications in urban planning challenges Land, 12 (Issue 1) (2023), Google Scholar
21. Li et al., 2021 J. Li, J. Boonaert, A. Doniec, G. Lozenguez Multi-models machine learning methods for traffic flow estimation from Floating Car Data Transportation Research Part C: Emerging Technologies, 132 (2021), Google Scholar
22. Madushaet al.J.P.S.S. Madushani, R.M.K. Sandamal, D.P.P. Meddage, H.R. Pasindu, P.I.A. Gomes Evaluating expressway traffic crash severity by using logistic regression and explainable & supervised machine learning classifiers Transport Engineer, 13 (2023), Google Scholar
23. Pourroostaei Ardakani et al., 2023 S. Pourroostaei Ardakani, X. Liang, K.T. Mengistu, R.S. So, X. Wei, B. He, A. Cheshmehzangi Road car accident prediction using a machine-learning-enabled data analysis Sustainability, 15 (7) (2023), Google Scholar
24. Rodrigue et al., 2019 J.P. Rodrigue, C. Comtois, B. Slack The geography of transport systems (2019) Google Scholar
25. Tillinghast and Duffy, 2023 D.J. Tillinghast, V.G. Duffy Crowdsourcing data to improve transportation safety and efficiency: A systematic review Lecture Notes in Computer Science (2023), Google Scholar
26. TM and CN, 2023 V. TM, C. C Traffic prediction using machine learning SAMRIDDHI: A Journal of Physical Sciences, Engineering and Technology, 15 (3) (2023), Google Scholar
27. Vanitha and Swedha, 2023 R. Vanitha, M. Swedha Prediction of road traffic using machine learning algorithms Middle East Journal of Applied Science & Technology, 6 (2) (2023), Google Scholar
28. Zhang et al., 2022 Z. Zhang, L.D. Han, Y. Liu Exploration and evaluation of crowdsourced probe-based Waze traffic speed Transportation Letters, 14 (5) (2022),  Google Scholar