TRAFFIC FLOW PREDICTION USING MACHINE LEARNING

A Project Report submitted in the partial fulfillment of

the requirements for the award of the Degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted By

K. DEEPTHI **(Regd.No:20NE1A0579)**

Under the Esteemed Guidance Of

Dr. LALU NAIK **M. Tech, Ph.D**

Professor

Department of Computer Science & Engineering

TIRUMALA ENGINEERING COLLEGE

(Approved by AICTE & Affiliated to JNTU, KAKINADA, Accredited by NAAC & NBA) Jonnalagadda, Narasaraopet, GUNTUR (Dt.), A.P.

2020-2024

TIRUMALA ENGINEERING COLLEGE

(Approved by AICTE & Affiliated to JNTU KAKINADA, Accredited by NAAC & NBA) Jonnalagadda, Narasaraopet-522601, Guntur (Dist) A.P . 2023-2024

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the project report entitled **"TRAFFIC FLOW PREDICTION"** is the bonafide work carried out by **K. DEEPTHI (20NE1A0579)** in partial fulfillment of the requirements for the award of **"Bachelor of Technology"** degree in the **Department of CSE** from J.N.T.U. KAKINADA during the year 2023-2024 under our guidance and supervision and worth of acceptance of requirements of the university.

Project Guide	**Head of the Department**
Dr. R LALU NAIK M. Tech, Ph.D	**Dr. N. Gopala krishna** M. Tech, Ph.D
Project Coordinator	**External Examiner**
Mr. S. Anil Kumar M. Tech(Ph.D)	

ACKNOWLEDGEMENT

I wish to express my thanks to carious personalities who are responsible for the completion of the project. I am extremely thankful to our beloved chairman Sri. Bolla Brahma Naidu, our secretary **Sri. R. Satyanarayana**, who took keen interest in our every effort throughout this course. I owe out gratitude to our principal sir **Dr. Y. V. Narayana M.E, Ph.D, FIETE** for his kindattention and valuable guidance throughout the course.

I express our deep felt gratitude to our H.O.D **Dr.N.GOPALA KRISHNA,M.tech, P.hD**, Professor and **Mr. S. Anil KumarM.Tech**, coordinator of the project for extending their encouragement. Their profoundknowledge and willingness have been a constant source of inspiration for us throughout theproject work.

I wish to express our sincere deep sense of gratitude to our, **Dr.Lalu Naik,Ph.D**, for significant suggestions and help in every respect to accomplish the project work. His persisting encouragement, everlasting patience and keen interest in discussions have benefited us to be extent that cannot be spanned by words to our college management for providing excellent lab facilities for completion of project within our campus.

I extend my sincere thanks to all other teaching and non-teaching staff of the department of CSE for their cooperation andencouragement during our B. Tech course. I have no words to acknowledge the warm affection, constant inspiration and encouragement that I received from my parent.

I affectionately acknowledge the encouragement received from my friends and those who involved in giving valuable suggestions had clarifying out doubts which had really helped us in successfullycompleting our project.

By

K. DEEPTHI (20NE1A0579)

ABSTRACT

In this project, we focus on the challenging task of predicting traffic congestion, with aparticular emphasis on considering the influence of weather conditions. To address this complex problem, we employ a diverse set of machine learning (ML) and deep learning (DL) algorithms, including Support Vector Machines (SVM), Decision Trees (DT), Bayesian Algorithm, Random Forest, and deep learning models such as Long Short-Term Memory (LSTM) and Convolutional Neural Networks (CNN). We rigorously analyse the performance of each model using key metrics, specifically Mean Squared Error (MSE) andRoot Mean Squared Error (RMSE).

Our findings reveal that the Random Forest algorithm emerges as the standout performer in accurately predicting traffic congestion, achieving the lowest error rates on our dataset. This exceptional performance is further supported by the quantitative evaluation, where Random Forest exhibited the lowest MSE and RMSE values. LSTM and CNN closely follow, demonstrating commendable predictive accuracy. Our comprehensive analysis underscores the practical applicability of these algorithms in addressing real-world traffic management challenges, with Random Forest leading the way as a robust solution for traffic congestion prediction in the presence of varying weather conditions.

INDEX

1. INTRODUCTION

1.1 BACKGROUND STUDY:

Traffic prediction using machine learning and deep learning has gained significant attention in recent years due to its potential to address urban traffic congestion and improve transportation management. In this section, we draw insights from different research papers and identify areas where each may have missed critical aspects, laying the groundwork for our project that aims to address those gaps.

Traffic flow prediction based on weather is a research field in intelligent transportation systems. Traditional prediction methods have limitations in accurately predicting short-term traffic flow due to the complexity of influencing factors. Lu Wang et all proposed, a shortterm traffic flow prediction model using short-term and long-term memory networks, along with variational modal decomposition, has shown promising results in achieving accurate predictions . Jingyi Lu et all In addition, incorporating weather information into traffic flow forecasting models has been found to improve forecast accuracy. Studies have shown that nextday weather forecasts can significantly improve the accuracy of electricity price forecasts in day-ahead markets, suggesting that weather forecasts can also play a role in predicting traffic flow . Bashir Mohammed et all Furthermore, the use of weather variables, such as clearsky index, cloud cover, relative humidity, atmospheric pressure, precipitation, temperature, and wind speed, in soybean volatility forecasting models has been found to outperform models without weather information, indicating the predictive power of weather in this domain as well.

Wang et al. as discussed by the authors proposed a novel approach that leverages deep learning techniques, specifically Long Short-Term Memory (LSTM) neural networks, AdaBoost, and gradient descent, to enhance the accuracy of traffic flow predictions. Wang et al. as discussed by the authors proposed a novel approach that leverages deep learning techniques, specifically Long Short-Term Memory (LSTM) neural networks, AdaBoost, and gradient descent, to enhance the accuracy of traffic flow predictions. Aditya and et all The use of Support Vector Regression algorithm for traffic flow prediction has shown

Aditya and et all The use of Support Vector Regression algorithm for traffic flow prediction has shown promising results in various studies, and it can be used to predict traffic flow at different time intervals, such as hourly, daily, weekly, or even yearly.

PROBLEM STATEMENT:

Traffic congestion is a widespread issue that leads to inefficiencies in transportation systems, increased commute times, and environmental concerns. The aim of this research project is to develop and evaluate machine learning and deep learning models for accurate and real-time prediction of traffic congestion levels. These models will leverage historical traffic data and incorporate weather conditions as a key factor affecting traffic flow. The project seeks to address the challenge of providing commuters with timely information about congestion, enabling them to make informed decisions and improve their commuting experience. Additionally, the research aims to identify the most effective algorithms for congestion prediction and assess their performance in diverse weather conditions and traffic scenarios.

1.2 RESEARCH QUESTION:

The primary research question addressed in this project is as follows:

- Can machine learning and deep learning algorithms accurately predict traffic flow while considering environmental variables?

AIMS OF THE RESEARCH:

Develop Accurate Traffic Congestion Prediction Models: The primary aim of this project is to develop machine learning and deep learning models that can accurately predict traffic congestion levels based on historical traffic and weather data.

1. **Weather Data:** Incorporating weather data into congestion prediction models is a key aim. Weather conditions can have a significant impact on traffic, and accounting for these variables can lead to more precise predictions.

2. **Evaluate Algorithm Performance:** The project aims to assess the performance of various machine learning and deep learning algorithms in traffic congestion prediction. This involves comparing the accuracy, speed, and resource requirements of different models.

1.3 OBJECTIVES:

1. **Data Collection:** Gather and pre-process historical traffic data, including congestion levels, and weather data from reliable sources.

2. **Feature Engineering:** Identify and select relevant features from the dataset including traffic flow, and weather conditions. These are helpful in predicting the output.

3. **Model Selection:** Experiment with a range of machine learning and deep learning algorithms to determine the most suitable models for congestion prediction.

4. **Model Training:** Train selected models using a portion of the dataset, ensuring that they learn to recognize patterns and relationships between traffic and weather variables.

5. **Hyperparameter Tuning:** Fine-tune the models by optimizing hyperparameters to improve prediction accuracy.

6. **Evaluation Metrics:** Evaluate model performance using appropriate metrics such as

Mean Squared Error (MSE), Root Mean Squared Error (RMSE)

7. **Real-time Prediction:** Implement a system capable of making real-time or nearrealtime traffic congestion predictions based on live weather and traffic data.

8. **Comparison of Algorithms:** Compare the performance of different algorithms to determine which one provides the most accurate and efficient predictions.

9. **Documentation and Reporting:** Create comprehensive documentation of the project, including methodology, results, and conclusions. Present findings and insights in a clear and organized report.

10. **Future Recommendations:** Provide recommendations for further improvements or enhancements to the congestion prediction system.

1.4. ORGANISATION OF THE REPORT:

This report is organized into several chapters, each focusing on specific aspects of the project:

1. Introduction

It serves as the introduction to the research report. It introduces the research topic, outlines the aims and objectives, and provides a concise background study. This chapter sets the stage for the subsequent chapters, providing readers with a clear understanding of the research's purpose and context

2. Literature Review

It is dedicated to conducting a comprehensive literature survey. This chapter delves into existing research, theories, and relevant literature pertaining to the research topic.

It critically evaluates and synthesizes the knowledge and insights from previous studies, providing a foundation for the research methodology.

3. System analysis

the research project outlines its system analysis. This chapter describes the systematic approach used in the project, including the selection of models, data collection processes, feasibility study, existed system and evaluation metrics. It provides a detailed insight into how the research was conducted, ensuring transparency in the research process and setting the stage for the subsequent chapters that will present and analyse the research results.

4. Design

It presents the design models such uml models, data flow diagrams, class diagrams, state diagrams, sequence diagrams and activity diagrams. These diagrams are very useful to Understand the problem and solution.

5. Implementation

It is focuses on the practical implementation of the methodology outlined .It provides a detailed account of how the selected approach and models were put into action in a real-world environment. This chapter often includes information about data gathering, model training, and the application of the chosen methodology to address the research objectives.

6. System Testing

Chapter 6 presents the results and performance evaluation of the research project. This section provides an in-depth analysis of how each selected model performed, including a thorough examination of performance metrics. This chapter is crucial for drawing conclusions and insights from the data collected and analysed throughout the project.

7. Conclusion

Chapter 7 serves as the culmination of the research report. It presents the key findings and insights derived from the study, summarizing the main outcomes and their significance.

8. Future Work

This chapter contains the future work of the traffic flow prediction, how the traffic predictions will be high accuracy in future.

9. Appendices

Chapter 9 houses the appendices section, where supplementary materials relevant to the research are provided. This may include additional data, charts, tables, detailed technical information, or any other materials that support and enhance the understanding of the research but are not part of the main body of the report.

10. Bibliography

Chapter 10, the final chapter, contains the list of references used throughout the research report. It provides complete and accurate citations for all sources, ensuring transparency and credibility by acknowledging the work of others that countries.

2. LITERATURE REVIEW

INTRODUCTION:

This chapter delves into existing research, theories, and relevant literature pertaining to the research topic. It critically evaluates and synthesizes the knowledge and insights from previous studies, providing a foundation for the research methodology and helping to identify gaps in the current body of knowledge.

OBSERVATIONS:

Ayele Gobezie et al. explore a range of machine learning and deep learning models for traffic flow prediction, though they do not specifically address weather-based traffic flow prediction. They investigate various models utilizing machine learning and deep learning techniques for traffic flow prediction. Additionally, they review the impact of different factors on the performance of traffic flow prediction, including benchmark performance evaluation metrics.

Lingbo Liu et al.propose a model known as the Attentive Traffic Flow Machine (ATFM), which effectively captures spatio-temporal features of traffic flow using an attention mechanism. The ATFM model consists of a convolutional layer connecting two Convolutional Long Short-Term Memory (ConvLSTM) units. It predicts short-term and long-term flow patterns within metropolitan areas by incorporating time series data, periodicity information, and other influential variables. This model surpasses existing methods in traffic flow prediction and can also be applied to predict pickup and dropoff demands.

Yanli Shao et al. introduce IL-TFNet, an Incremental Learningbased CNN-LSTM model designed for traffic flow prediction in mobile applications. They propose a lightweight CNN architecture that combines spatiotemporal and external environmental features to enhance prediction performance and efficiency. The model also extracts unknown traffic accident information using a K-means clustering algorithm. During model training, incremental learning algorithms are employed to reduce updating costs, ensuring high realtime performance and low computational overhead. Furthermore, the paper suggests combining incremental learning with active learning for fine-tuning the prediction model in specific scenarios. Experimental results confirm the strong performance of IL-TFNet in short-term traffic flow prediction, with preliminary data preprocessing, including feature extraction of uncertainty features such . They propose a new deeplearning-based approach that integrates contextual information.

Chen, Z et al. enhance the efficiency of urban road networks in the Internet of Vehicles (IoV) environment by utilizing computational intelligence technologies, including deep reinforcement learning (DRL) and long short-term memory (LSTM) models, to forecast traffic flow and represent traffic situations. Experimental results demonstrate the effectiveness of the proposed DRL-based LSTM model in predicting traffic flow data. Additionally, the traffic situation description model accurately captures traffic conditions.

Aldwyish, A et al.utilize a hierarchical architecture and multi-task learning to predict traffic at different resolutions, providing a comprehensive understanding of traffic dynamics. Their approach achieves competitive results with simpler baselines in terms of complexity while maintaining computational efficiency.

Nigam, A et al. focus on the accurate prediction of macroscopic traffic stream variables such as speed and flow, essential for intelligent transportation systems. They employ datadriven approaches and consider the influence of exogenous factors, including weather variables such as precipitation. Various deep learning models, including RNN and LSTM, are compared for traffic flow parameter prediction. The training and testing sets are constructed using traffic sensor data and rainfall data from various weather stations in San Diego. Experiments demonstrate that incorporating rainfall data improves prediction accuracy for recurrent learning models, with LSTM delivering the best performance among deep learning methods. Tedjopurnomo et al. address traffic congestion as a global issue and emphasize the importance of forecasting future traffic to prevent it. They provide an extensive review of Deep Neural Network (DNN) models for traffic prediction, explaining their architectures, classifying the literature, highlighting similarities and differences, and addressing challenges and future research directions.

Kundu, S et al. underscore the significance of traffic forecasting for smart cities to reduce congestion and CO2 emissions, despite the complex and nonlinear nature of traffic patterns. Their research explores how deep learning models can outperform traditional time series and regression models in predicting traffic flow, particularly in capturing intricate, nonlinear traffic patterns.

The paper contrasts the performance of state-of-the-art deep learning models with classical time series and regression models on two traffic flow datasets.

Rahman, F. I. et al. present an approach to enhance traffic flow prediction in cities by integrating short-term traffic flow prediction with machine learning classifiers (KNN, SVM, ANN) and meteorological data. They examine the impact of feature selection on prediction accuracy, with KNN outperforming SVM and ANN when considering weather data.

Evaluation metrics, including MAPE and R-square, measure prediction accuracy. Results indicate that KNN performs better with the addition of weather information, achieving a MAPE of 14.384% and an R-square of 0.948.

Valova, I et al. introduce a novel method called Convolutional Long Short Term Memory (CLSTM), which integrates spatial and temporal information for prediction. They compare CLSTM with other deep learning architectures (GRU, LSTM) and baseline methods (VAR and historical average). Experiments encompass variations in parameters such as the number of units per layer, number of layers, optimizers, learning rate, and sequence input lengths. Tables and graphs in the results section demonstrate the efficiency of the proposed CLSTM method. Ma, D et al. address the challenge of reflecting intra-day and interday traffic patterns and their correlation with weather in short-term traffic flow prediction. They propose a new deeplearning-based approach that integrates contextual information and traffic patterns using a combination of CNN and LSTM units. Historical context is embedded to enhance forecasting. Using real-data case studies, the research demonstrates that the proposed method achieves over 90% prediction accuracy, surpassing competitive baselines and providing robust forecasting performance across various scenarios.

Azzedine, Boukerche et al. conduct a study that surveys state-ofthe-art machine learning (ML) models for traffic flow prediction. They classify these models based on the ML theories they employ and provide insights into their strengths, weaknesses, and suitability for different prediction tasks. Nonparametric ML methods, including regression and kernel-based models, show promise for traffic flow prediction by effectively handling the nonlinearities of traffic data and requiring less a priori knowledge. They propose a new deeplearning-based approach that integrates contextual information and traffic patterns using a combination of CNN and LSTM units. Historical context is embedded to enhance forecasting. . They propose a new deeplearning-based approach that integrates contextual information and traffic patterns using a combination of CNN and LSTM units. Historical context is embedded to enhance forecasting.

LINKAGE TO AIMS:

Our research journey was guided by a thorough literature review, which illuminated significant gaps in traffic flow prediction. While some studies leaned towards machine learning (ML) approaches and others explored deep learning (DL) techniques, none systematically compared the two. Additionally, the crucial influence of weather on traffic often went unconsidered. These observations framed our primary project aim: to conduct a comprehensive comparative analysis of ML and DL algorithms for traffic congestion prediction, while also incorporating weather variables as a critical factor. The literature review served as a compass, directing us toward this challenging yet critical endeavour.

By systematically evaluating the strengths and weaknesses of ML and DL in the context of traffic prediction and recognizing the impact of weather, we aim to enhance the accuracy and effectiveness of traffic flow models, offering valuable insights for urban planning and transportation management. While some studies leaned towards machine learning (ML) approaches and others explored deep learning (DL) techniques, none systematically compared the two. Additionally, the crucial influence of weather on traffic often went unconsidered.

3. SYSTEM ANALYSIS

3.1 INTRODUCTION:

System analysis begins with the collection of various data sources such as traffic volume, vehicle speeds, road conditions, weather conditions, and special events affecting traffic flow. These data can be gathered from sensors, cameras, GPS devices, and other monitoring systems.

3.2 EXISTED SYSTEM:

Integrated traffic management systems integrate traffic flow prediction with real-time traffic monitoring and control capabilities. These systems continuously collect data from various sources such as traffic sensors, surveillance cameras, and GPS devices, and use predictive models to anticipate congestion and optimize traffic flow in real time. Integrated systems can also incorporate dynamic route guidance, adaptive traffic signal control, and incident management strategies to mitigate congestion and improve overall transportation efficiency. The existed system does not take the weather data as historical data.

3.3 PROPOSED SYSTEM:

Incorporating weather data into traffic flow prediction systems can significantly improve the accuracy and robustness of predictions, as weather conditions have a profound impact on traffic patterns and congestion. Here's a proposed system that integrates weather data into traffic flow prediction. In proposed system also solve complex proplems.

The proposed system aims to address the limitations of the existing system by integrating machine learning and deep learning techniques with weather information for more accurate traffic flow prediction. Specifically, the system will leverage Long Short-Term Memory (LSTM) neural networks, AdaBoost, and gradient descent algorithms to enhance prediction accuracy. Additionally, the system will incorporate a diverse set of weather variables, such as clear sky index, cloud cover, relative humidity, atmospheric pressure, precipitation, temperature, and wind speed, to improve forecasting capabilities. By combining advanced prediction techniques with comprehensive weather data, the proposed system aims to significantly enhance urban transportation management, reduce congestion, improve road safety, and contribute to a more sustainable urban environment. Here's a proposed system that integrates weather data into traffic flow prediction. In proposed system also solves the complex problems.

SOFTWARE TOOLS USED:

we used Anaconda for Jupyter Notebook as my primary development environment for the project. Jupyter Notebook is a Powerful Tool for Data Science and Research. It is an opensource web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. Originally developed as a part of the IPython project, Jupyter has evolved into a versatile platform used by researchers, data scientists, educators, and professionals across various domains. we will explore the features, benefits, and applications of Jupyter Notebook in research projects.

Applications in Research:

1. **Data Exploration:** Jupyter Notebook simplifies the process of exploring and visualizing datasets, enabling researchers to gain insights quickly.
2. **Machine Learning and Deep Learning:** It is widely used for developing, training, and evaluating machine learning and deep learning models due to its code interactivity and visualization capabilities.
3. **Statistical Analysis:** Researchers can perform complex statistical analyses and hypothesis testing, with the ability to document each step for transparency and reproducibility.
4. **Collaboration:** Jupyter Notebooks can be shared with colleagues and collaborators, fostering collaborative research efforts and enabling real-time collaboration.

3.4 FEASIBILITY STUDY:

A feasibility study for traffic prediction involves assessing the viability and potential impact of implementing a system to forecast traffic conditions. Here are the key steps to conduct such a study. Feasibility plays a vital role in calculating the technical feasibility and operational feasibility.

3.4.1 Technical Feasibility:

Evaluate the technical feasibility of implementing a traffic prediction system. Assess the availability and quality of data sources, such as traffic sensors, GPS data, weather data, and historical traffic patterns. Determine the computational requirements and feasibility of the predictive algorithms.

3.4.2 Operational Feasibility:

An operational feasibility study for traffic flow prediction aims to assess whether implementing a predictive system is practical and viable within the operational context of transportation management. Here's how you can structure such a study.

3.5 ANALYSIS:

Data collection:

The first step is to gather relevant data. This includes historical traffic flow data, which can be obtained from various sources such as traffic sensors, GPS devices, and traffic management systems. Other data sources like weather conditions, road infrastructure, events, and holidays can also be important for accurate predictions.

Data Preprocessing:

Once the data is collected, it needs to be preprocessed to clean and prepare it for analysis. This may involve handling missing values, smoothing noisy data, normalizing data across different sources, and encoding categorical variables.

Feature Selection and Engineering:

Identifying the most relevant features (or variables) that influence traffic flow is crucial. This could include factors like time of day, day of the week, weather conditions, road type, historical traffic patterns, and special events. Feature engineering may involve creating new features or transforming existing ones to better capture patterns in the data.

Model Selection:

Choosing an appropriate predictive model is critical for accurate traffic flow prediction. This could range from simple statistical models like ARIMA or linear regression to more complex machine learning algorithms like neural networks or support vector machines. The choice of model depends on factors such as the complexity of the data, the forecasting horizon, and computational resources available.

Model Training:

Once a model is selected, it needs to be trained on the historical data. This involves fitting the model to the training data to learn the underlying patterns and relationships between input features and traffic flow.

Model Evaluation:

After training, the model's performance needs to be evaluated using validation data that the model hasn't seen before. Common evaluation metrics for traffic flow prediction include mean absolute error (MAE), mean squared error (MSE), root mean squared error (RMSE), and mean absolute percentage error (MAPE).

Deployment and Monitoring:

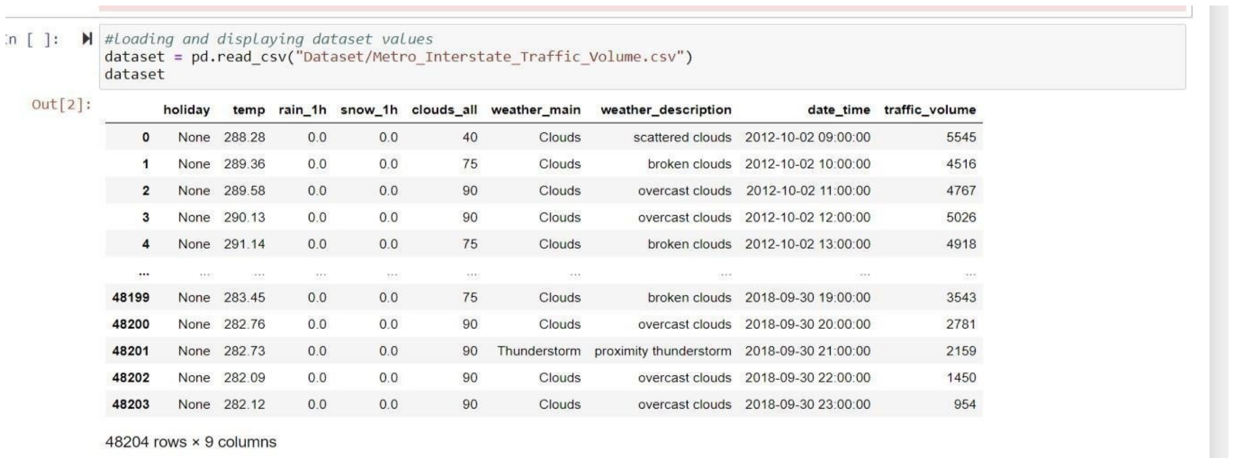
Once the model is trained and evaluated, it can be deployed to make real-time predictions. Continuous monitoring is essential to ensure that the model remains accurate over time, as traffic patterns may change due to various factors such as construction, accidents, or changes in urban development.

3.6 SYSTEMATIC APPROACH FOR PROJECT IMPLEMENTATION:

3.6.1 Dataset:

The most critical stage in the development of an ML/DL model is the creation of data sets. Datasets are available in a variety of forms, including csv files, HTML files, and.xlsx files. We chose CSV format for our model, which means comma-separated values files.

We used the Metro City Road Traffic dataset to train and test each system which we obtained from Kaggle.



```
In [ ]:  #loading and displaying dataset values
         dataset = pd.read_csv("Dataset/Metro_Interstate_Traffic_Volume.csv")
         dataset
```

Out[2]:

	holiday	temp	rain_1h	snow_1h	clouds_all	weather_main	weather_description	date_time	traffic_volume
0	None	288.28	0.0	0.0	40	Clouds	scattered clouds	2012-10-02 09:00:00	5545
1	None	289.36	0.0	0.0	75	Clouds	broken clouds	2012-10-02 10:00:00	4516
2	None	289.58	0.0	0.0	90	Clouds	overcast clouds	2012-10-02 11:00:00	4767
3	None	290.13	0.0	0.0	90	Clouds	overcast clouds	2012-10-02 12:00:00	5026
4	None	291.14	0.0	0.0	75	Clouds	broken clouds	2012-10-02 13:00:00	4918
...	...	...	...	...	...	...	...	...	...
48199	None	283.45	0.0	0.0	75	Clouds	broken clouds	2018-09-30 19:00:00	3543
48200	None	282.76	0.0	0.0	90	Clouds	overcast clouds	2018-09-30 20:00:00	2781
48201	None	282.73	0.0	0.0	90	Thunderstorm	proximity thunderstorm	2018-09-30 21:00:00	2159
48202	None	282.09	0.0	0.0	90	Clouds	overcast clouds	2018-09-30 22:00:00	1450
48203	None	282.12	0.0	0.0	90	Clouds	overcast clouds	2018-09-30 23:00:00	954

48204 rows × 9 columns

Figure Dataset

The above fig. shows the dataset. The data set consists of 48204 rows and 9columns. The first row shows the column names.

3.6.2 Data Visualization:

Data visualization is a fundamental part of EDA. It helps in representing data graphically to gain insights.

Common visualization techniques include histograms, bar charts, scatter plots, box plots, and heatmaps. Visualization helps in understanding the distribution, relationships, and potential anomalies in the data.

Pre-processing:

The pre-processing stage is used to prepare raw data for the ML/DL algorithm. It is a critical stage in the ML/DL model. Real-world data is mostly made up of noise, unsuitable formats, and missing values. Processing aids in cleaning the dataset and preparing for use with ML/DL.

1.Data Acquisition: This initial step involves obtaining the necessary data from various sources, gathering it for analysis and model development.

2. Handling Missing Data: Real-world data often contains gaps or missing values. In this substage, we employ strategies to address missing data, ensuring our dataset is complete and ready for analysis. Missing data is very important to complete the data because gaps or missing values are the issue now a days we are facing so, here we fill the missing data by analyse and ensuring the dataset complete.

3. Categorical Data Encoding: Many datasets include categorical variables that require transformation into numerical form for ML/DL algorithms to process effectively. Encoding techniques facilitate this conversion.

4. Dataset Splitting: To assess the model's performance, it is essential to divide the dataset into training and testing subsets. This allows us to train the model on one portion and evaluate its performance on another, ensuring an unbiased assessment.

5. Feature Scaling: Variability in the magnitude of features can affect the performance of certain ML/DL algorithms. Feature scaling standardizes feature values to ensure consistent treatment during modelling.

In sum, the data pre-processing phase acts as a crucial foundation, shaping the raw data into a format conducive to the successful application of ML/DL techniques while improving the overall accuracy and efficiency of the model.

To do pre-processing using python we have used libraries like NumPy, pandas, Matplotlib .Each serves unique purpose.

NUMPY: NumPy, short for "Numerical Python," is a fundamental Python library for numerical and scientific computing. It provides support for large, multi-dimensional arrays and matrices, as well as a wide range of mathematical functions to operate on these arrays.

PANDAS: Pandas is a popular Python library for data manipulation and analysis. It provides data structures and functions for working with structured data, such as tables and time series, making it an essential tool for data scientists and analysts. Pandas introduces two primary data structures: the Data Frame, which resembles a spreadsheet with rows and columns.

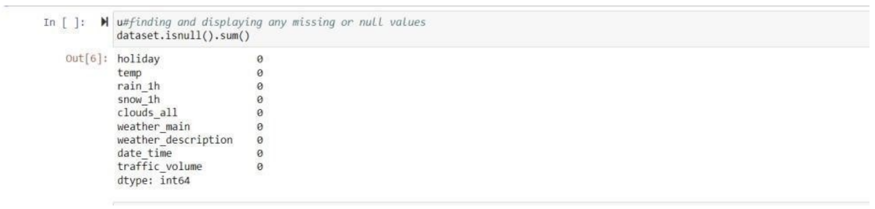
MATPLOTLIB: Matplotlib is a Python library for creating static, animated, and interactive data visualizations, such as line charts, scatter plots, and bar graphs. It provides a flexible and customizable way to display data, making it a valuable tool for data analysis, scientific research.

Handling Missing Data:

After dataset creation next step is handling missing data. If the dataset has missing values, it will create a problem for our ML/Dl model. There are two ways to handle missing data:

1. By deletion of rows: This approach deals with null value by deleting a row or column with invalid values.
2. By calculating mean: -This approach calculates mean of column or row, and then missing values are replaced by that same mean value To handle missing values, we used the scikit-learn library in our model. The class is created for sky learn, pre-processing library to take missing values using technique mean calculation.

Using jupyter notebook, we have found that our dataset has no missing values .So it will be easy for accurate prediction.



```
In [ ]:   u#finding and displaying any missing or null values
          dataset.isnull().sum()

Out[6]:   holiday                0
          temp                   0
          rain_1h                0
          snow_1h                0
          clouds_all             0
          weather_main           0
          weather_description    0
          date_time              0
          traffic_volume         0
          dtype: int64
```

Figure 3.2 Missing value handling

Categorical data encoding:

Categorical encoding in machine learning (ML) and deep learning (DL) models refers to the process of converting categorical or non-numeric data into a numerical format so that these data can be used as inputs for training and making predictions with the model.

Categorical encoding is a crucial pre-processing step because it allows ML and DL models to work with a wider range of data and make meaningful predictions or classifications based on categorical information. The choice of encoding method depends on the specific characteristics of the data and the requirements of the model being used.

Splitting of dataset into training and testing:

This process involves dividing a given dataset into two separate subsets – a training set and a testing set. The training set is used to train and build a machine learning model, while the testing set is used to evaluate the model's performance and assess its ability to make predictions or classifications on new, unseen data. This division helps ensure that the model's effectiveness.

3.7 REQUIREMENT SPECIFICATION:

3.7.1 Hardware Requirements:

System	: Intel Core i5
Hard Disk	: 500GB
Monitor	: 15'' LED
RAM	: 32GB

Computer Power: For more complex machine learning models such as neural networks or ensemble methods, high-performance computing resources may be required. This could include multi-core CPUs, GPUs (Graphics Processing Units), or even TPUs (Tensor Processing Units) for deep learning models. Cloud computing services like AWS, Azure, or Google Cloud Platform can provide scalable computing resources on-demand, allowing you to scale up or down based on computational needs.

Memory(RAM): The amount of RAM required depends on the size of the dataset and the complexity of the model. As a general guideline, several gigabytes (GB) to tens of gigabytes of RAM may be necessary for training large neural networks or processing extensive spatiotemporal data.

Storage: Adequate storage space is needed to store historical traffic data, model parameters, and intermediate results.

Solid-state drives (SSDs) are recommended for faster data access and processing compared to traditional hard disk drives (HDDs), especially when dealing with large datasets.

Networking: Fast and reliable internet connectivity may be necessary for accessing real-time traffic data sources, cloud computing resources, or remote servers.

Dedicated servers or clusters: For large-scale traffic flow prediction projects, dedicated servers or clusters may be required to handle the computational load.

3.7.2 Software Requirements:

Operating system	:	windows 10
Coding Language	:	Python
Tool	:	Jupyter Notebook

MODEL SELECTION:

This section describes about models we have chosen for traffic prediction. They are:

1.SVM

2.Decision Tree (DT)

3.Random Forest

4.Baysian Ridge Algorithm

5. CNN (Convolutional Neural Networks)

6.LSTM

Support Vector Machine:

The Support Vector Machine (SVM) is a supervised machine learning algorithm known for its effectiveness in classification and regression tasks. It is particularly well-suited for scenarios with complex data distributions and high-dimensional feature spaces. The central concept behind SVM is finding an optimal hyperplane that best separates data points into different classes or predicts continuous values in regression tasks.

Advantages of SVM in Traffic Congestion Prediction:

1. **Effective in High-dimensional Spaces**: SVM performs well in highdimensional feature spaces, making it suitable for traffic data that often involves numerous variables and complex interactions.
2. **Robust to Noise**: SVM is robust to noisy data and outliers. It can filter out irrelevant information and focus on the most important data points for accurate predictions.
3. **Versatility:** SVM can be used for both classification and regression tasks, making it versatile in addressing various aspects of traffic congestion prediction.

4. Non-linear Separation

4. Margin Maximization: Support Vector Machine (SVM) algorithm has emerged as powerful tool in traffic congestion prediction, offering accuracy and versatility in modelling complex traffic patterns. Its ability to handle highdimensional data, non-linear relationships, and noisy inputs makes it a choice for our research. By using the margin maximization a large margin can avoid the effect of random noise and reduce overfitting . A larger margin will lead to smaller VC dimension, reduce the number of potential classifiers, and, therefore, reduce the possibility of generalization error. Margin Maximization is plays crucial role in traffic prediction, offering accuracy in complex traffic patterns.

Decision Tree:

Decision trees

Decision trees are versatile and widely used machine learning algorithms known for their interpretability and effectiveness in both classification and regression tasks. They have found applications in various domains, from healthcare to finance and beyond.
Using a Decision Tree algorithm for traffic congestion prediction has several advantages:

a. Interpretability: Decision Trees are easy to understand and interpret. The model's decisionmaking process is represented as a tree-like structure, which can be visualized and explained to non-technical stakeholders, such as city planners or transportation authorities.
b. Handling Non-linearity: Traffic congestion is often influenced by non-linear relationships between various factors like time of day, weather conditions, and road types. Decision Trees can capture these non-linearities effectively.

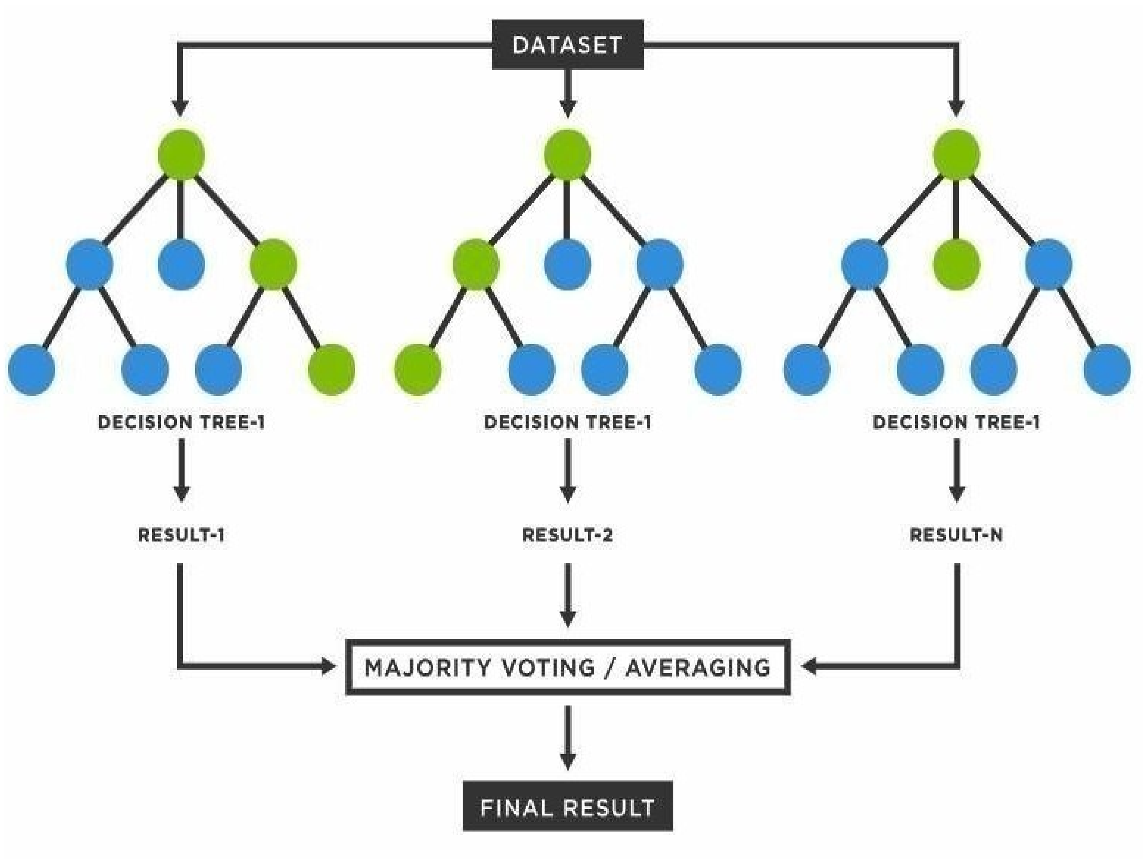
c. Feature Importance: Decision Trees provide a feature importance score, which helps you identify the most critical features contributing to traffic congestion. This information can guide traffic management decisions and help prioritize interventions.
d. No Assumption of Linearity: Unlike linear regression, Decision Trees do not assume a linear relationship between predictors and the target variable. This makes them more suitable for modelling complex and non-linear traffic patterns.
e. Handling Mixed Data Types
f. Robustness to Outliers
g. Ease of Handling Missing Data
h. Ensemble Methods: We can further improve the performance of Decision Trees by using ensemble methods like Random Forests or Gradient Boosting. These methods combine multiple decision trees to reduce overfitting and increase predictive accuracy.
i. Scalability: Decision Trees can handle large datasets reasonably and can be parallelized for faster training on multi-core processors or distributed computing environments.
j. Versatility: Decision Trees can be used for both classification and regression tasks. If the project evolves to include other trafficrelated predictions or classifications, we can continue to use the same algorithm.

However, it's important to note that Decision Trees also have some limitations, such as their susceptibility to overfitting, which can be mitigated with appropriate techniques like pruning or using ensemble methods.

Random Forest:

Machine learning has seen significant advancements over the years, and one of the standout algorithms is the Random Forest. Random Forest is a versatile and powerful ensemble learning method that has found applications in various domains. In this comprehensive overview, we will delve into the fundamentals of Random Forest, its construction, advantages, practical applications.

Random Forest is an ensemble learning technique used for both classification and regression tasks. It's an ensemble of decision trees, each trained on a different subset of the data. The name "Random Forest" reflects the idea of introducing randomness into the construction process, which helps mitigate overfitting and improve model performance.



Random forest is a commonly-used machine learning algorithm, trademarked by Leo Breiman and Adele Cutler, that combines the output of multiple decision trees to reach a single result. Its ease of use and flexibility have fueled its adoption, as it handles both classification and regression problems. The random forest classifier is widely used in different fields due to its accuracy and robustness. Since its invention, the random forest algorithm is naturally developed for multi-dimensional vectorial data since features can be directly sampled during the decision tree construction procedure.

Advantages of Random Forest:

Random Forest offers several advantages that make it a suitable choice for traffic congestion prediction:

1. High Accuracy: Random Forest generally produces accurate predictions due to its ensemble of decision trees. The combination of multiple trees helps reduce bias and variance, leading to robust models. By aggregating the outputs of multiple trees, it reduces the risk of overfitting and provides robust results.
2. Robustness to Noise: Random Forest is less sensitive to noisy data and outliers compared to individual decision trees. The aggregation of multiple trees helps smooth out the impact of outliers.
3. Handling of Large Datasets

4. Feature Importance
5. Non-linearity
6. Scalability
7. Handling Missing Data:

Random Forest can handle missing data gracefully, imputing values based on the information present in the dataset.

Its ability to offer accurate predictions, handle large datasets, and discover essential features makes it a valuable tool, and RF's extensive capabilities make it an excellent fit for our project.

BAYESIAN RIDGE REGRESSION:

The Bayesian Ridge regression algorithm is a statistical technique used in machine learning and regression analysis. It's a variation of linear regression, designed to address some of the shortcomings of traditional linear regression models. Here's an explanation of how Bayesian Ridge regression works:

1. **Linear Regression Basis**: Like traditional linear regression, Bayesian Ridge regression aims to model the relationship between a dependent variable (the target) and one or more independent variables (the features) by fitting a linear equation. However, it introduces a Bayesian framework to estimate the model parameters.
2. **Bayesian Approach**: Bayesian methods incorporate prior knowledge or beliefs about the model parameters into the modelling process. In Bayesian Ridge, this means that we start with prior assumptions about the values of the model parameters, such as the coefficients of the features. These prior assumptions are typically represented by probability distributions.
3. **Parameter Estimation**: Bayesian Ridge uses the observed data and the prior beliefs to estimate the posterior probability distribution of the model parameters. This posterior distribution represents our updated knowledge about the parameters after observing the data.
4. **Regularization**: One of the significant advantages of Bayesian Ridge is that it automatically incorporates regularization into the model. Regularization helps prevent overfitting by adding a penalty term to the likelihood function. This penalty discourages overly complex models by shrinking the parameter estimates towards zero. Bayesian Ridge uses a specific type of regularization called ridge regression.
5. **Uncertainty Estimation**: Bayesian Ridge provides not only point estimates for the model parameters but also estimates of uncertainty. It quantifies how confident the model is in its parameter estimates. This is particularly valuable when making predictions because it allows you to assess the uncertainty associated with each prediction.

6. Prediction: To make predictions using Bayesian Ridge, not only get a single predicted value but also a prediction interval that captures the range within which the actual value is likely to fall with a certain probability.

Advantages of the Bayesian Ridge Algorithm in Traffic Congestion Prediction:

1. Probabilistic Modelling: The Bayesian Ridge Algorithm incorporates Bayesian principles, allowing it to provide probabilistic predictions. Instead of just estimating congestion levels, it provides uncertainty estimates, which are crucial for making informed decisions in traffic management.
2. Regularization: Bayesian Ridge includes automatic parameter regularization, which helps prevent overfitting. This is vital when dealing with noisy and complex traffic data, ensuring that the model generalizes well to unseen situations.
3. Robustness to Outliers: Bayesian Ridge is robust to outliers in the data, making it suitable for real-world traffic datasets that may have occasional anomalies.
4. Incorporating Prior Information: This algorithm allows the incorporation of prior information or expert knowledge into the model through the Bayesian framework. This is valuable for enhancing the model's accuracy and reliability.
5. Scalability

In summary, the Bayesian Ridge Algorithm is a valuable tool for traffic congestion prediction due to its regularization capabilities, robustness to outliers, and the ability to incorporate prior knowledge.

3.8 DEEP LEARNING MODELS:

In the realm of traffic congestion prediction, Convolutional Neural Networks (CNNs) and Long Short-Term Memory networks (LSTMs) are powerful deep learning models. CNNs excel at extracting spatial patterns from visual data like traffic camera images, while LSTMs specialize in capturing temporal dependencies in traffic time series data. By combining the strengths of CNNs' spatial feature extraction with LSTMs' ability to model sequences and time-evolving traffic conditions, these models offer a holistic approach to and predicting traffic congestion.

Convolutional Neural Networks (CNNs):

Convolutional Neural Networks (CNNs) can be effectively used for traffic congestion prediction and analysis. CNN is a type of deep learning model primarily designed for processing and analysing structured grid data, such as images and videos. CNNs have become a fundamental tool in computer vision tasks, revolutionizing fields like image classification, object detection, and image segmentation.

While CNNs are traditionally associated with processing image data, they can also be adapted for non-image data, such as tabular traffic data or sensor data, by utilizing a technique called 1D or 2D convolution depending on the data's structure. Here's how you can apply CNNs to traffic data sets with no images:

1. **Data Representation**:

- We convert our non-image traffic data into a structured grid format, which can be considered as a 1D or 2D grid. For example, we can organize data into a matrix where rows represent time steps, and columns represent different features or sensors.

2. **1D or 2D Convolution**:

- We use 1D convolution if our data is one-dimensional (e.g., time series data) or 2D convolution for more complex gridlike data (e.g., sensor grid data).
- We define convolutional filters to slide over the data grid, capturing local patterns and relationships between features.

3. **Feature Extraction**:

- As the convolutional layers progress, they automatically learn and extract relevant features from our traffic data. These features could represent temporal patterns, spatial correlations, or any other relationships present in our data. Feature extraction enables machine learning models to focus on essential information most relevant to the task, providing improved model generalization, data interpretation.

4. Pooling Layers

- After convolution, we can still use pooling layers, similar to how they are used in traditional CNNs for images, to reduce the spatial dimensions of our feature maps and capture higherlevel patterns.

5. **Flattening and Fully Connected Layers**:

- We flatten the output from the convolutional and pooling layers into a 1D vector.
- We connect this vector to one or more fully connected (dense) layers for further processing and making predictions.

- A flatten layer reshapes the input data, while a fully connected layer performs computations on the reshaped data.

6. **Training and Evaluation**:

- We train the CNN on our traffic data, just like we would with image data. We define a suitable loss function and optimization method.
- We evaluate the model's performance using appropriate metrics, such as Mean Absolute Error (MAE) or Root Mean Squared Error (RMSE), depending on the nature of our traffic congestion prediction task.

By applying CNNs to non-image traffic data in this way, we can leverage their ability to automatically learn spatial and temporal patterns, which may exist in the data even when visual information is absent. This approach can be particularly useful for traffic congestion prediction tasks involving sensor data, weather data, time series, or other structured grid-like data sources.

LSTM:

A Long Short-Term Memory network, or LSTM for short, is a type of recurrent neural network (RNN) architecture that is designed to handle and model sequential data. LSTMs are particularly effective in capturing and learning patterns, dependencies, and relationships within sequences of data. They are widely used in various applications, including natural language processing, time series analysis, speech recognition, and more.

What sets LSTMs apart from traditional RNNs is their ability to maintain long-term dependencies and mitigate the vanishing gradient problem. LSTMs achieve this through the use of specialized memory cells and gating mechanisms that allow them to selectively store, update, and retrieve information over extended sequences. This makes LSTMs well-suited for tasks that involve time series data, where understanding context and long-range dependencies is crucial for accurate predictions and modelling.

In the context of traffic congestion prediction, a Long Short-Term Memory network (LSTM) is a specialized deep learning model used to analyse and forecast traffic conditions over time.

Handling Temporal Dependencies: LSTMs excel at modeling and capturing the complex temporal dependencies present in traffic data. They can capture both shortterm fluctuations and long-term trends, crucial for accurate congestion predictions.

1. **Sequence Modelling**: LSTMs process traffic data as sequences, allowing them to consider historical patterns and trends, making them particularly suitable for time-series traffic data.
2. **Long-Term Memory**: LSTMs mitigate the vanishing gradient problem seen in traditional RNNs, enabling them to effectively retain and use information from distant past time steps.

3. **Feature Extraction**: LSTMs automatically extract relevant temporal features from data, eliminating the need for manual feature engineering.
4. **Flexibility**: They can handle various data types, such as traffic flow rates, sensor readings, and historical data, making them adaptable to different congestion prediction scenarios.
5. **Real-Time Predictions**: LSTMs can provide real-time predictions, allowing for immediate responses to changing traffic conditions.
6. **Multimodal Data Fusion**: They can be part of a larger multimodal system that combines information from various sources, including traffic sensors, weather data, and social media feeds.

LSTM models are ideal for traffic congestion prediction due to their ability to capture temporal dependencies, handle sequences, retain long-term memory, extract features automatically, and adapt to diverse data sources. Their ability to model and predict traffic patterns over time makes them an excellent fit for our project.

PERFORMANCE EVALUATION:

Performance evaluation is a crucial step in assessing the effectiveness of model. Performance metrics are essential tools in evaluating the effectiveness and accuracy of our congestion prediction models. In this context, we employ two key metrics: Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). MSE measures the average of the squared differences between predicted and actual values, providing insights into the overall prediction accuracy. RMSE, derived from MSE, offers a more interpretable measure by taking the square root of MSE, aligning the error scale with the original data. These metrics serve as critical indicators of our model's predictive capabilities, helping us assess the quality of our traffic congestion predictions following the model selection process.

Through the chosen methodology, we achieved a structured and rigorous approach to address the project's research question. Data pre-processing ensured that our models could handle realworld traffic and weather data effectively, enhancing their predictive capabilities.

Model selection involved careful consideration of various machine learning and deep learning algorithms, allowing us to choose the most suitable model for accurate traffic flow prediction while considering environmental factors.

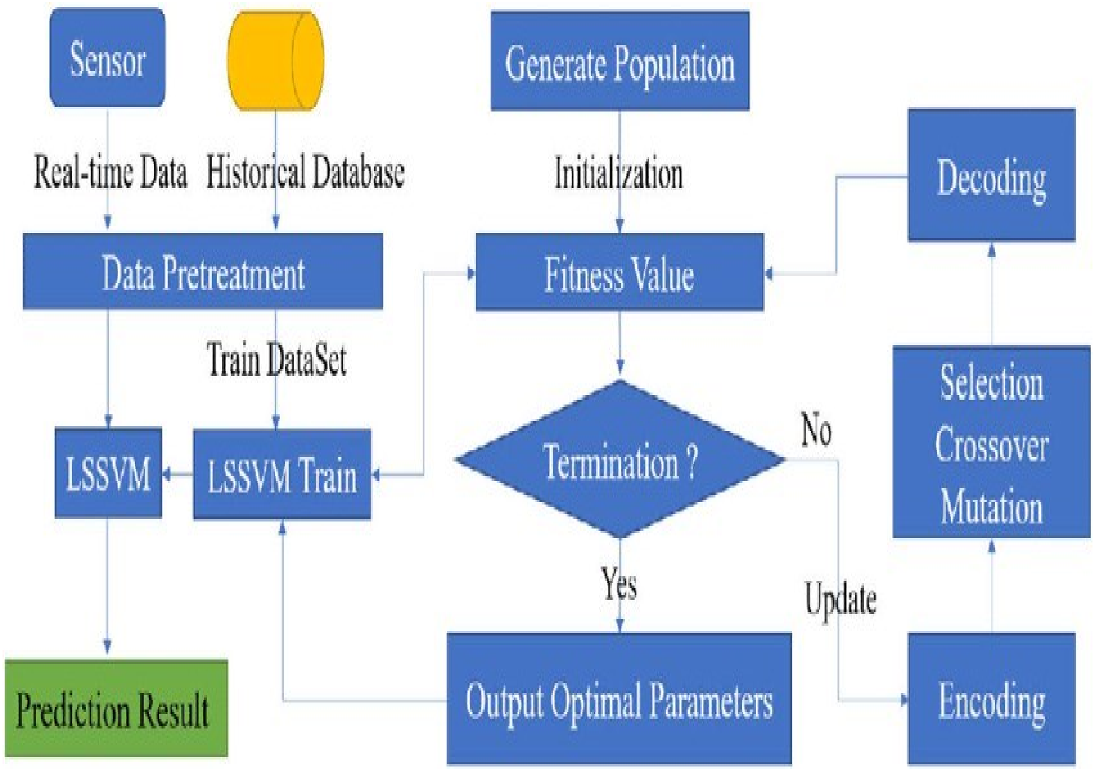
These methodologies were instrumental in setting the project on a path toward success, aligning our approach with the project's objectives, and contributing to the achievement of meaningful results. This chapter played a pivotal role in shaping the project's workflow. The groundwork laid here will soon yield insights and performance evaluations in the following chapters.

4. DESIGN

4.1 UML DIAGRAMS:

Unified Modeling Language (UML) diagrams are invaluable tools for visualizing, designing, and communicating various aspects of a system, including traffic flow prediction systems. Here's how UML diagrams can be utilized in the context of designing such a system.

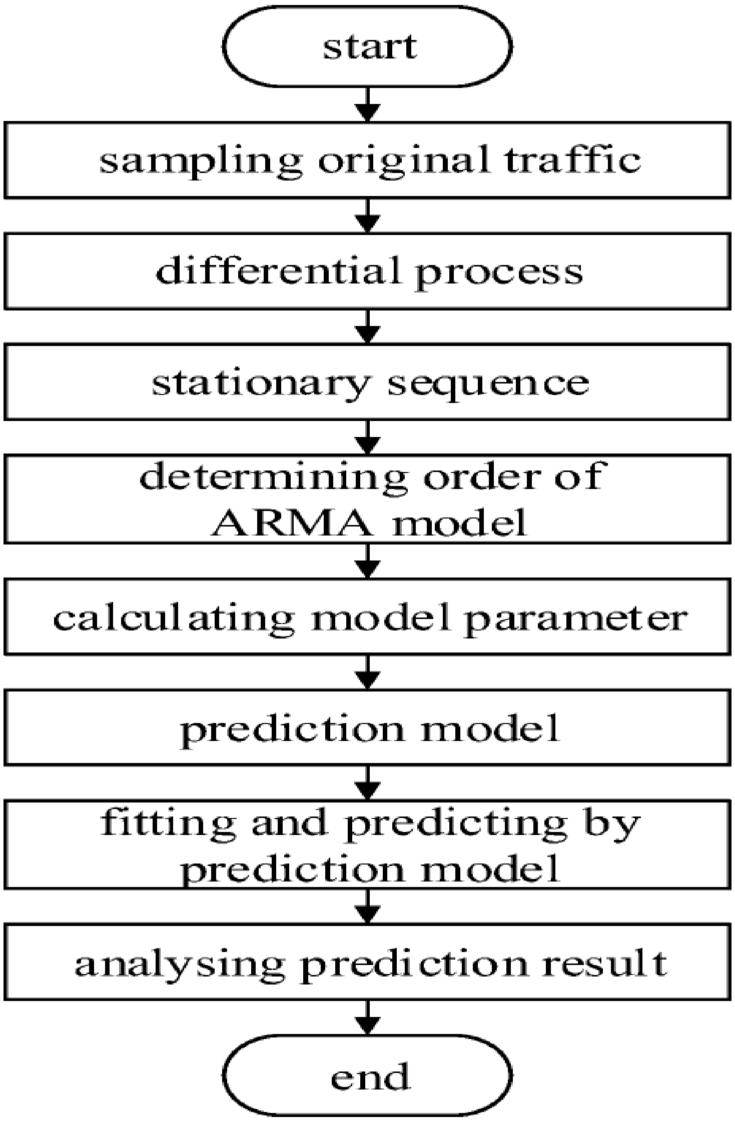
UML diagrams contains class diagrams, sequence diagrams, activity diagrams, usecase diagrams, component diagrams and deployment diagrams. These diagrams very important to understand the problem and these diagrams gives the best explanation of matter.



4.1.1 Class Diagrams

A class diagram is a type of UML (Unified Modeling Language) diagram that represents the static structure of a system by showing the classes in the system, their attributes, methods, and the relationships between classes. Here's a breakdown of its key components and uses.

A Class represents a blueprint for creating objects. It consists of a name and a list of attributes and methods. Class diagrams can serve as a basis for generating code in object-oriented programming languages, as they provide a clear specification of classes and their relationships.

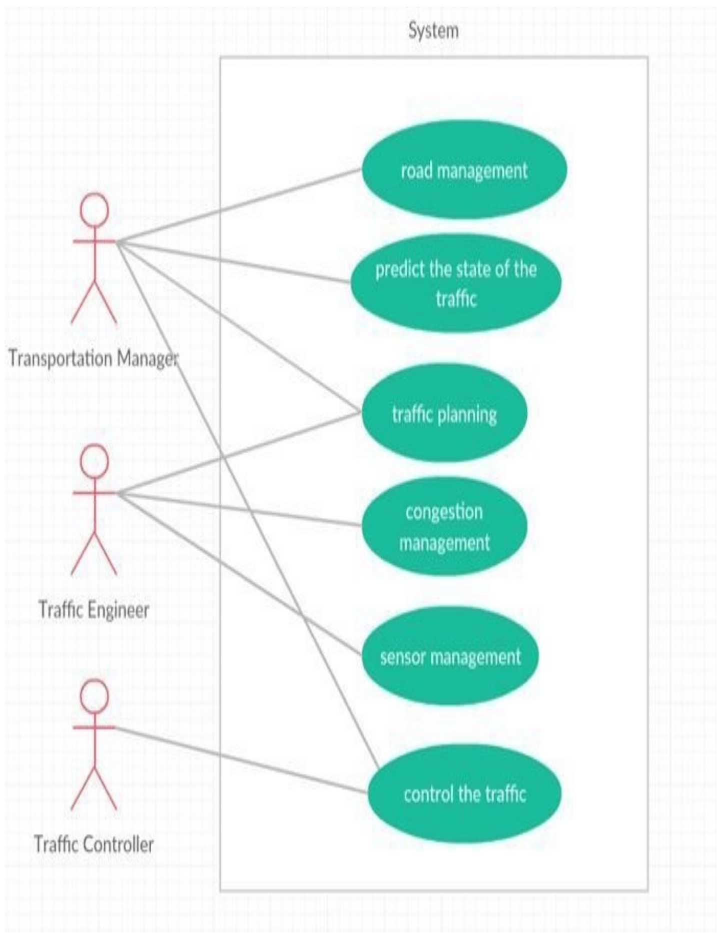

start
sampling original traffic
differential process
stationary sequence
determining order of ARMA model
calculating model parameter
prediction model
fitting and predicting by prediction model
analysing prediction result
end

4.1.2 Use case Diagram

Use case diagrams are part of the Unified Modeling Language (UML) and are used to visually represent the interactions between users (actors) and a system to achieve specific goals. Here's a brief overview of use case diagrams and their uses.

Actors Represent entities outside the system that interact with the system to accomplish specific goals. Actors can be users, other systems, or external entities.

Usecases Represent specific functionalities or services provided by the system. Each use case describes a sequence of interactions between the system and actors to achieve a particular goal.



5. IMPLEMENTATION

5.1 INTRODUCTION:

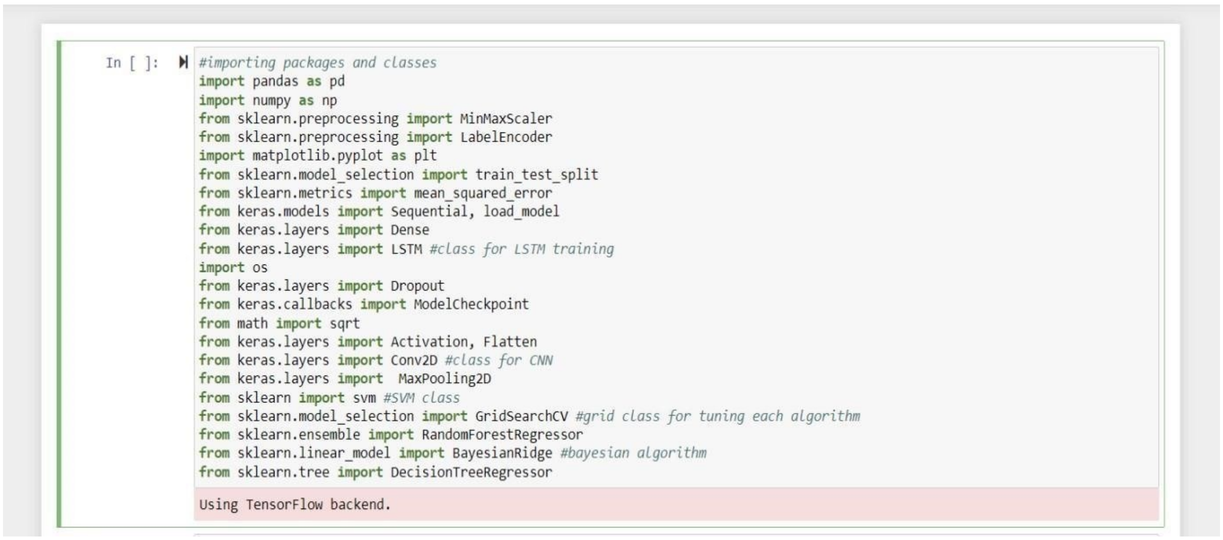
In this chapter, we transition from theory to practice, diving into the real-world application of our research. Using Jupyter Notebook as our canvas, we'll walk through the practical steps that bring our methodologies and models to life. This is where data transforms into insights, algorithms take shape, and our research vision becomes reality.

Welcome to the hands-on journey of implementation**.**

5.2 OUTPUT SCREENS:

Importing packages and libraries:

The extensive set of imported classes and packages that support our research methods is shown in Figure



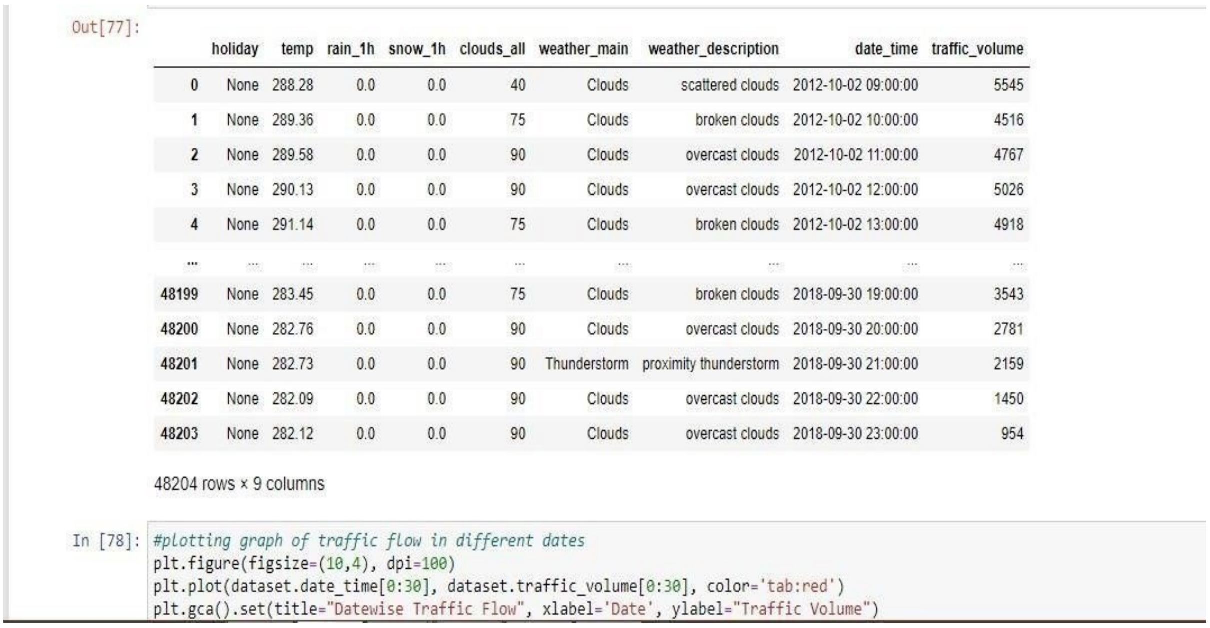
```
In [ ]:  #importing packages and classes
         import pandas as pd
         import numpy as np
         from sklearn.preprocessing import MinMaxScaler
         from sklearn.preprocessing import LabelEncoder
         import matplotlib.pyplot as plt
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import mean_squared_error
         from keras.models import Sequential, load_model
         from keras.layers import Dense
         from keras.layers import LSTM #class for LSTM training
         import os
         from keras.layers import Dropout
         from keras.callbacks import ModelCheckpoint
         from math import sqrt
         from keras.layers import Activation, Flatten
         from keras.layers import Conv2D #class for CNN
         from keras.layers import  MaxPooling2D
         from sklearn import svm #SVM class
         from sklearn.model_selection import GridSearchCV #grid class for tuning each algorithm
         from sklearn.ensemble import RandomForestRegressor
         from sklearn.linear_model import BayesianRidge #bayesian algorithm
         from sklearn.tree import DecisionTreeRegressor

         Using TensorFlow backend.
```

Figure :Importing Packages and libraries

This code imports necessary libraries and programs for machine learning and data manipulation operations. It consists of machine learning algorithms including Support Vector Machines, Decision Tree, Random Forests, and Bayesian Ridge, as well as data handling libraries like Pandas and NumPy, pre-processing tools for data scaling and encoding, visualization tools using Matplotlib, and visualization tools. Convolutional Neural
Network (CNN) layers and other deep learning components from Kera's, such as the Dense and LSTM layers of neural networks, are also imported.

Exploratory Data analysis:

We conducted Exploratory Data Analysis (EDA) to gain insights into the characteristics of our dataset before pre-processing. This involved plotting and visualizing the raw data, creating various types of graphs, and performing initial observations. The primary objectives of this EDA were to identify data distributions, detect outliers, explore correlations between variables, and observe any notable trends or patterns. The findings from this exploratory analysis guided our decisions on data preprocessing techniques and model selection, ensuring a comprehensive understanding of the data and enhancing the quality of our subsequent analysis."



Out[77]:

	holiday	temp	rain_1h	snow_1h	clouds_all	weather_main	weather_description	date_time	traffic_volume
0	None	288.28	0.0	0.0	40	Clouds	scattered clouds	2012-10-02 09:00:00	5545
1	None	289.36	0.0	0.0	75	Clouds	broken clouds	2012-10-02 10:00:00	4516
2	None	289.58	0.0	0.0	90	Clouds	overcast clouds	2012-10-02 11:00:00	4767
3	None	290.13	0.0	0.0	90	Clouds	overcast clouds	2012-10-02 12:00:00	5026
4	None	291.14	0.0	0.0	75	Clouds	broken clouds	2012-10-02 13:00:00	4918
...	...	...	...	...	...	...	...	...	...
48199	None	283.45	0.0	0.0	75	Clouds	broken clouds	2018-09-30 19:00:00	3543
48200	None	282.76	0.0	0.0	90	Clouds	overcast clouds	2018-09-30 20:00:00	2781
48201	None	282.73	0.0	0.0	90	Thunderstorm	proximity thunderstorm	2018-09-30 21:00:00	2159
48202	None	282.09	0.0	0.0	90	Clouds	overcast clouds	2018-09-30 22:00:00	1450
48203	None	282.12	0.0	0.0	90	Clouds	overcast clouds	2018-09-30 23:00:00	954

48204 rows × 9 columns

```
In [78]: #plotting graph of traffic flow in different dates
         plt.figure(figsize=(10,4), dpi=100)
         plt.plot(dataset.date_time[0:30], dataset.traffic_volume[0:30], color='tab:red')
         plt.gca().set(title="Datewise Traffic Flow", xlabel='Date', ylabel="Traffic Volume")
```

Figure : Traffic Dataset

The dataset is depicted in the diagram above. It should be noted, however, that some of the entries in the dataset are not numeric. Because machine learning algorithms only work with numeric input, we must solve this by using the Label Encoder class to convert these nonnumeric values into a compatible numeric format." Exploratory Data Analysis refers to the method of studying and exploring record sets to apprehend their predominant traits, discover patterns, locate outliers, and identify relationships between variables. EDA is normally carried out as a preliminary step before undertaking extra formal statistical analyses or modelling. It involves the examining the information for errors, lacking values, and inconsistencies. It includes techniques including records imputation, managing missing statistics, and figuring out and getting rid of outliers.

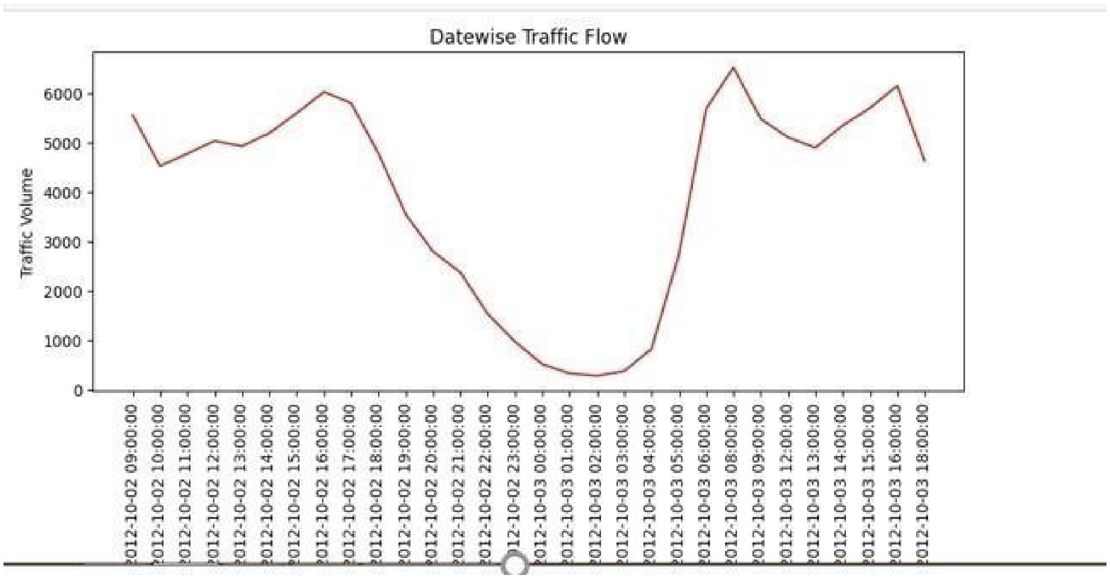
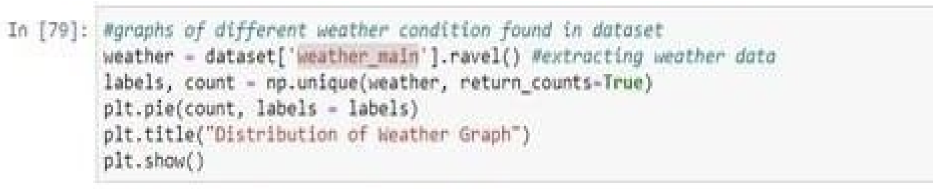
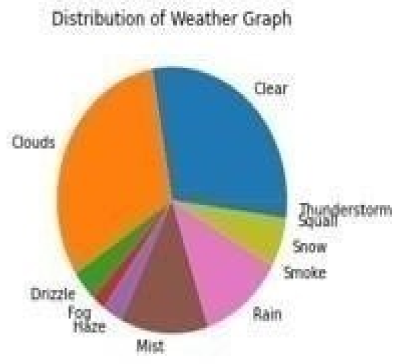


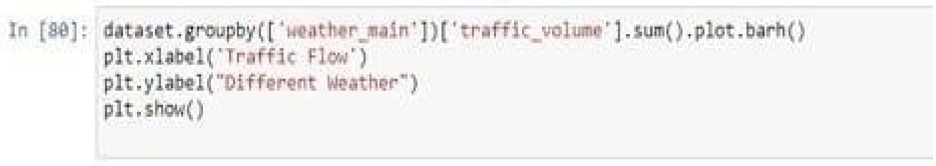
Figure : Datewise Traffic Flow

Finding and drawing a graph of traffic volumes on various dates, where the x-axis is the date and the y-axis is the number of vehicles, as shown in the screen above.



```
In [79]: #graphs of different weather condition found in dataset
         weather = dataset['weather_main'].ravel() #extracting weather data
         labels, count = np.unique(weather, return_counts=True)
         plt.pie(count, labels = labels)
         plt.title("Distribution of Weather Graph")
         plt.show()
```



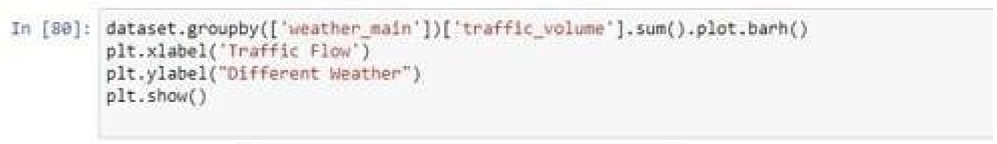


```
In [80]: dataset.groupby(['weather_main'])['traffic_volume'].sum().plot.barh()
         plt.xlabel('Traffic Flow')
         plt.ylabel("Different Weather")
         plt.show()
```

Figure : Traffic under different weather conditions

The graph above shows the percentage of traffic under different weather conditions. It involves examining the information for errors, lacking values, and inconsistencies.

It includes techniques including records imputation, managing missing statistics, and figuring out and getting rid of outliers.



```
In [80]: dataset.groupby(['weather_main'])['traffic_volume'].sum().plot.barh()
         plt.xlabel('Traffic Flow')
         plt.ylabel("Different Weather")
         plt.show()
```

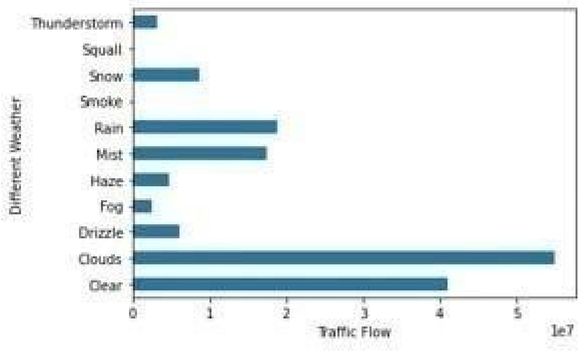
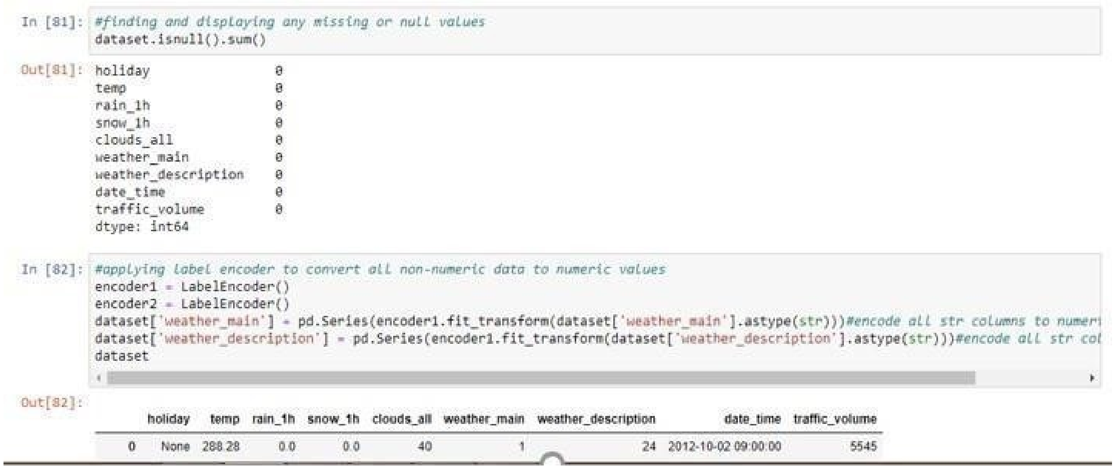


Figure : Traffic count under various weather situations

The x-axis in the above graph depicts count, and the y-axis the weather, and it shows traffic count under various weather situations.

5.3 DATA PREPROCESSING:

Finding missing values:



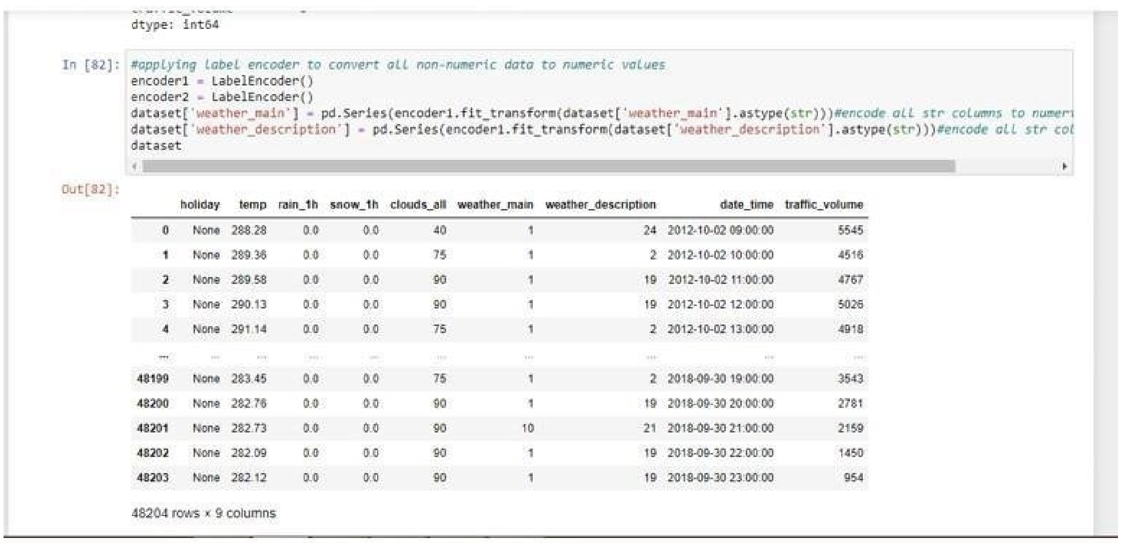
```
In [81]: #finding and displaying any missing or null values
         dataset.isnull().sum()

Out[81]: holiday                0
         temp                   0
         rain_1h                0
         snow_1h                0
         clouds_all             0
         weather_main           0
         weather_description    0
         date_time              0
         traffic_volume         0
         dtype: int64

In [82]: #applying label encoder to convert all non-numeric data to numeric values
         encoder1 = LabelEncoder()
         encoder2 = LabelEncoder()
         dataset['weather_main'] = pd.Series(encoder1.fit_transform(dataset['weather_main'].astype(str)))#encode all str columns to numeri
         dataset['weather_description'] = pd.Series(encoder1.fit_transform(dataset['weather_description'].astype(str)))#encode all str col
         dataset

Out[82]:
```

	holiday	temp	rain_1h	snow_1h	clouds_all	weather_main	weather_description	date_time	traffic_volume
0	None	288.28	0.0	0.0	40	1	24	2012-10-02 09:00:00	5545

Figure :Finding and displaying count of missing values

In above screen finding and displaying count of missing values for each column but this dataset has 0 missing values.

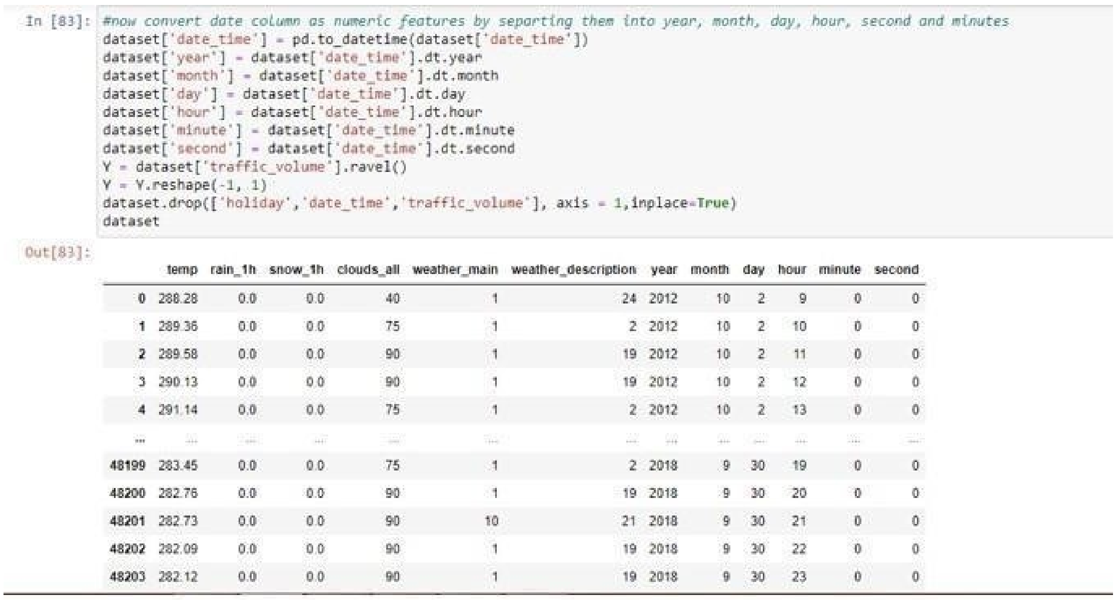
Categorical Encoding:



```
dtype: int64

In [82]: #applying label encoder to convert all non-numeric data to numeric values
         encoder1 = LabelEncoder()
         encoder2 = LabelEncoder()
         dataset['weather_main'] = pd.Series(encoder1.fit_transform(dataset['weather_main'].astype(str)))#encode all str columns to numeri
         dataset['weather_description'] = pd.Series(encoder1.fit_transform(dataset['weather_description'].astype(str)))#encode all str col
         dataset

Out[82]:
       holiday    temp  rain_1h  snow_1h  clouds_all  weather_main  weather_description            date_time  traffic_volume
     0    None  288.28      0.0      0.0          40             1                   24  2012-10-02 09:00:00            5545
     1    None  289.36      0.0      0.0          75             1                    2  2012-10-02 10:00:00            4516
     2    None  289.58      0.0      0.0          90             1                   19  2012-10-02 11:00:00            4767
     3    None  290.13      0.0      0.0          90             1                   19  2012-10-02 12:00:00            5026
     4    None  291.14      0.0      0.0          75             1                    2  2012-10-02 13:00:00            4918
   ...     ...     ...      ...      ...         ...           ...                  ...                  ...             ...
 48199    None  283.45      0.0      0.0          75             1                    2  2018-09-30 19:00:00            3543
 48200    None  282.76      0.0      0.0          90             1                   19  2018-09-30 20:00:00            2781
 48201    None  282.73      0.0      0.0          90            10                   21  2018-09-30 21:00:00            2159
 48202    None  282.09      0.0      0.0          90             1                   19  2018-09-30 22:00:00            1450
 48203    None  282.12      0.0      0.0          90             1                   19  2018-09-30 23:00:00             954

48204 rows × 9 columns
```

Figure :Categorical Encoding

In above screen using label encoder class we have converted all non-numeric values into numeric values and in below screen we are converting DATE into year, month, day and hour format like below screen

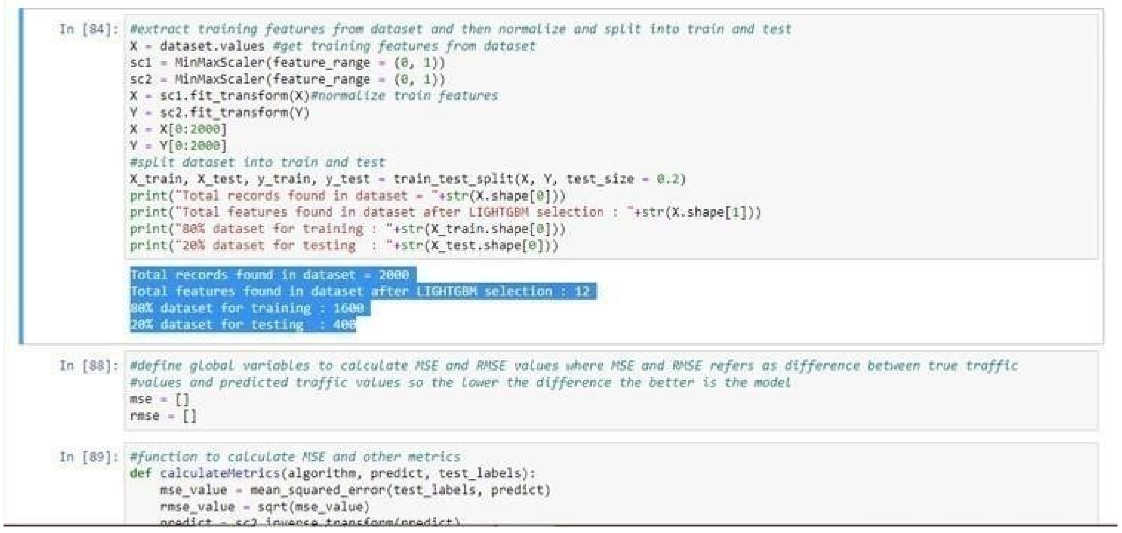


```
In [83]: #now convert date column as numeric features by separting them into year, month, day, hour, second and minutes
         dataset['date_time'] = pd.to_datetime(dataset['date_time'])
         dataset['year'] = dataset['date_time'].dt.year
         dataset['month'] = dataset['date_time'].dt.month
         dataset['day'] = dataset['date_time'].dt.day
         dataset['hour'] = dataset['date_time'].dt.hour
         dataset['minute'] = dataset['date_time'].dt.minute
         dataset['second'] = dataset['date_time'].dt.second
         Y = dataset['traffic_volume'].ravel()
         Y = Y.reshape(-1, 1)
         dataset.drop(['holiday','date_time','traffic_volume'], axis = 1,inplace=True)
         dataset

Out[83]:
          temp  rain_1h  snow_1h  clouds_all  weather_main  weather_description  year  month  day  hour  minute  second
     0  288.28      0.0      0.0          40             1                   24  2012     10    2     9       0       0
     1  289.36      0.0      0.0          75             1                    2  2012     10    2    10       0       0
     2  289.58      0.0      0.0          90             1                   19  2012     10    2    11       0       0
     3  290.13      0.0      0.0          90             1                   19  2012     10    2    12       0       0
     4  291.14      0.0      0.0          75             1                    2  2012     10    2    13       0       0
   ...     ...      ...      ...         ...           ...                  ...   ...    ...  ...   ...     ...     ...
 48199  283.45      0.0      0.0          75             1                    2  2018      9   30    19       0       0
 48200  282.76      0.0      0.0          90             1                   19  2018      9   30    20       0       0
 48201  282.73      0.0      0.0          90            10                   21  2018      9   30    21       0       0
 48202  282.09      0.0      0.0          90             1                   19  2018      9   30    22       0       0
 48203  282.12      0.0      0.0          90             1                   19  2018      9   30    23       0       0
```

Figure : Categorical Encoding

In above screen date converted into year and month formats and now all dataset values are in numeric format and now we will extract training features from above dataset.

Splitting the dataset into train and test data:



```
In [84]: #extract training features from dataset and then normalize and split into train and test
         X = dataset.values #get training features from dataset
         sc1 = MinMaxScaler(feature_range = (0, 1))
         sc2 = MinMaxScaler(feature_range = (0, 1))
         X = sc1.fit_transform(X)#normalize train features
         Y = sc2.fit_transform(Y)
         X = X[0:2000]
         Y = Y[0:2000]
         #split dataset into train and test
         X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2)
         print("Total records found in dataset = "+str(X.shape[0]))
         print("Total features found in dataset after LIGHTGBM selection : "+str(X.shape[1]))
         print("80% dataset for training : "+str(X_train.shape[0]))
         print("20% dataset for testing  : "+str(X_test.shape[0]))

         Total records found in dataset = 2000
         Total features found in dataset after LIGHTGBM selection : 12
         80% dataset for training : 1600
         20% dataset for testing  : 400

In [88]: #define global variables to calculate MSE and RMSE values where MSE and RMSE refers as difference between true traffic
         #values and predicted traffic values so the lower the difference the better is the model
         mse = []
         rmse = []

In [89]: #function to calculate MSE and other metrics
         def calculateMetrics(algorithm, predict, test_labels):
             mse_value = mean_squared_error(test_labels, predict)
             rmse_value = sqrt(mse_value)
             predict = sc2.inverse_transform(predict)
```

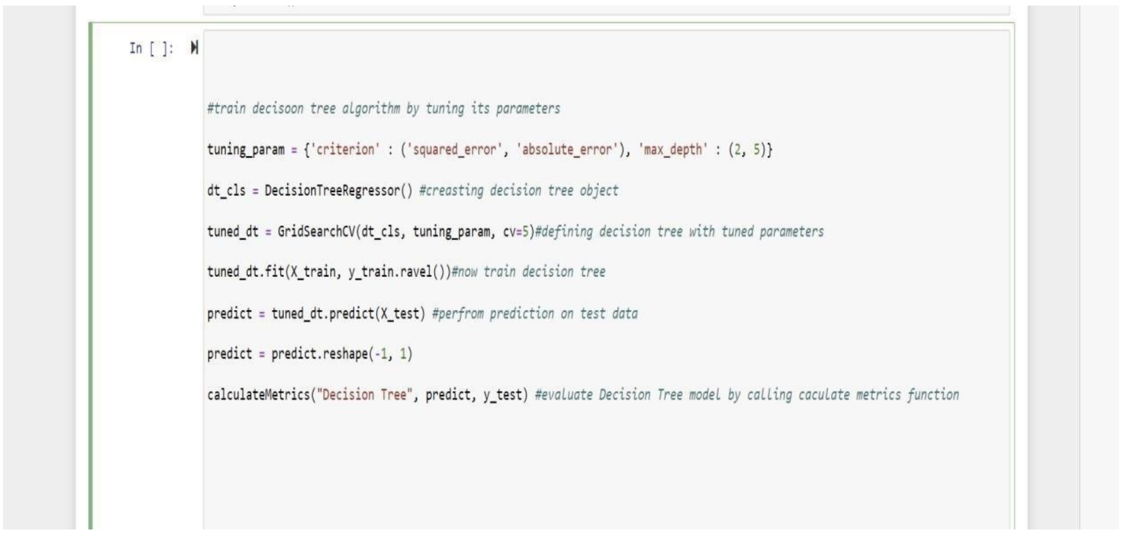
Figure : Splitting the dataset into train and test data

In above screen we are normalizing and splitting dataset into train and test where application using 80% dataset values for training and 20% for testing and in blue colour, we can see train and test values count.

5.4 IMPLEMENTATION OF MODELS IN CODING ENVIRONMENT:

IMPLEMENTATION OF DECISION TREE:

The below figure shows the code used for implementation of DT algorithm.



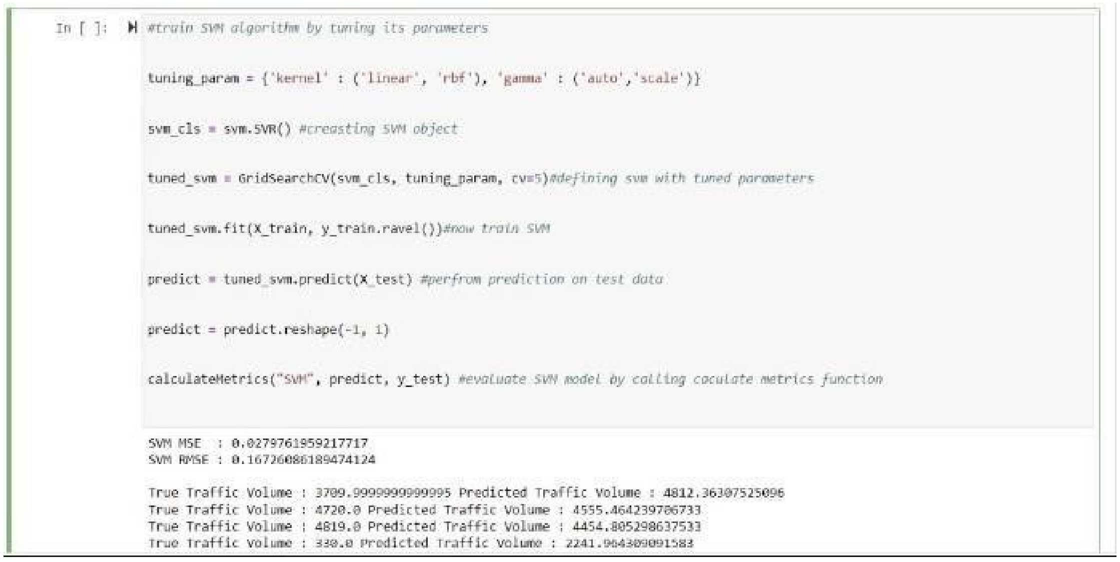
```
In [ ]:

         #train decisoon tree algorithm by tuning its parameters

         tuning_param = {'criterion' : ('squared_error', 'absolute_error'), 'max_depth' : (2, 5)}

         dt_cls = DecisionTreeRegressor() #creasting decision tree object

         tuned_dt = GridSearchCV(dt_cls, tuning_param, cv=5)#defining decision tree with tuned parameters

         tuned_dt.fit(X_train, y_train.ravel())#now train decision tree

         predict = tuned_dt.predict(X_test) #perfrom prediction on test data

         predict = predict.reshape(-1, 1)

         calculateMetrics("Decision Tree", predict, y_test) #evaluate Decision Tree model by calling caculate metrics function
```

Figure 5.10 Implementation of DT algorithm

This code snippet represents the process of tuning and evaluating a Decision Tree Regressor model for predictive performance:

The code begins by defining a dictionary, tuning_param, which lists potential hyperparameters for tuning. Then, an instance of a Decision Tree Regressor model, dt_cls, is created. Next, the GridSearchCV function is employed to systematically explore various hyperparameter combinations from tuning_param and perform a grid search with 5-fold cross-validation to identify the optimal settings for the Decision Tree model. The model is then trained on the training dataset, X_train and y_train. After training, it generates predictions for the test data, X_test, and these predictions are reshaped to ensure compatibility. Finally, a custom function, calculateMetrics, is called to assess the model's performance, typically by calculating metrics such as Mean Absolute Error (MAE) or Mean Squared Error (MSE) to gauge how well the Decision Tree model predicts the target values in y_test.

IMPLEMENTATION OF SVM:

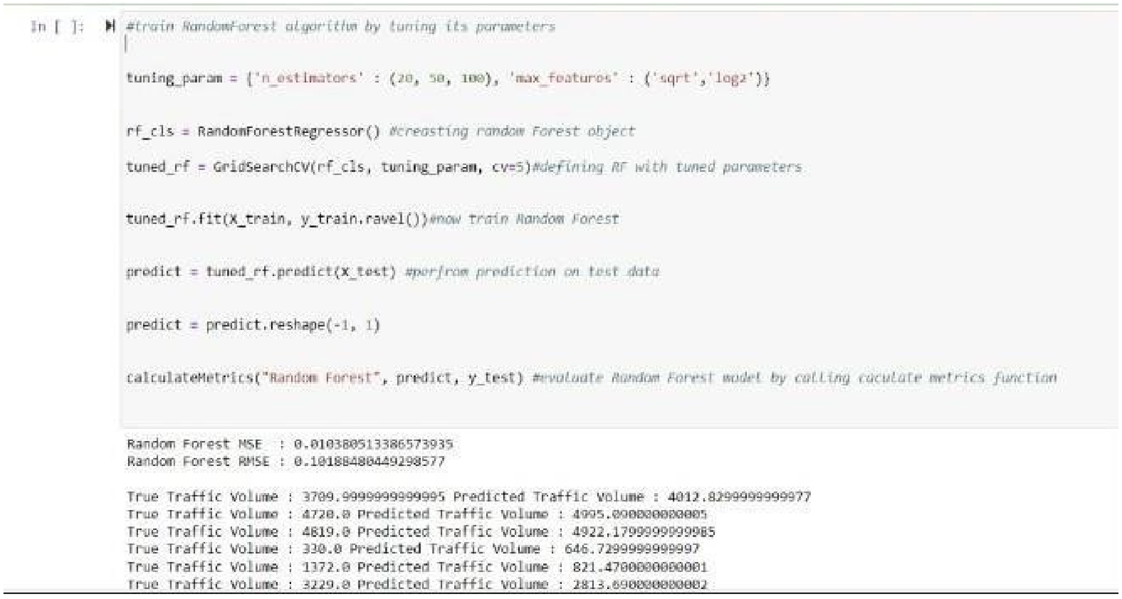


```
In [ ]:  #train SVM algorithm by tuning its parameters

         tuning_param = {'kernel' : ('linear', 'rbf'), 'gamma' : ('auto','scale')}

         svm_cls = svm.SVR() #creasting SVM object

         tuned_svm = GridSearchCV(svm_cls, tuning_param, cv=5)#defining svm with tuned parameters

         tuned_svm.fit(X_train, y_train.ravel())#now train SVM

         predict = tuned_svm.predict(X_test) #perfrom prediction on test data

         predict = predict.reshape(-1, 1)

         calculateMetrics("SVM", predict, y_test) #evaluate SVM model by calling caculate metrics function

         SVM MSE  : 0.0279761959217717
         SVM RMSE : 0.16726086189474124

         True Traffic Volume : 3709.9999999999995 Predicted Traffic Volume : 4812.36307525096
         True Traffic Volume : 4720.0 Predicted Traffic Volume : 4555.464239706733
         True Traffic Volume : 4819.0 Predicted Traffic Volume : 4454.805298637533
         True Traffic Volume : 330.0 Predicted Traffic Volume : 2241.964309091583
```

Figure : Implementation of SVM

The above fig 5.11 shows the implementation of the SVM model which will give us the MSE and RMSE values of models with a graph showing the true and predicted values of the model. Support Vector Machine (SVM) is a powerful machine learning algorithm used for linear or nonlinear classification, regression, and even outlier detection tasks. SVMs can be used for a variety of tasks, such as text classification, image classification, spam detection, handwriting identification, gene expression analysis, face detection.

IMPLEMENTATION OF RANDOM FOREST:



```
In [ ]:  #train RandomForest algorithm by tuning its parameters

         tuning_param = {'n_estimators' : (20, 50, 100), 'max_features' : ('sqrt','log2')}

         rf_cls = RandomForestRegressor() #creasting random Forest object

         tuned_rf = GridSearchCV(rf_cls, tuning_param, cv=5)#defining RF with tuned parameters

         tuned_rf.fit(X_train, y_train.ravel())#now train Random Forest

         predict = tuned_rf.predict(X_test) #perfrom prediction on test data

         predict = predict.reshape(-1, 1)

         calculateMetrics("Random Forest", predict, y_test) #evaluate Random Forest model by calling caculate metrics function
```

```
Random Forest MSE  : 0.010380513386573935
Random Forest RMSE : 0.10188480449298577

True Traffic Volume : 3709.9999999999995 Predicted Traffic Volume : 4012.8299999999977
True Traffic Volume : 4720.0 Predicted Traffic Volume : 4995.090000000005
True Traffic Volume : 4819.0 Predicted Traffic Volume : 4922.1799999999985
True Traffic Volume : 330.0 Predicted Traffic Volume : 646.7299999999997
True Traffic Volume : 1372.0 Predicted Traffic Volume : 821.4700000000001
True Traffic Volume : 3229.0 Predicted Traffic Volume : 2813.6900000000002
```

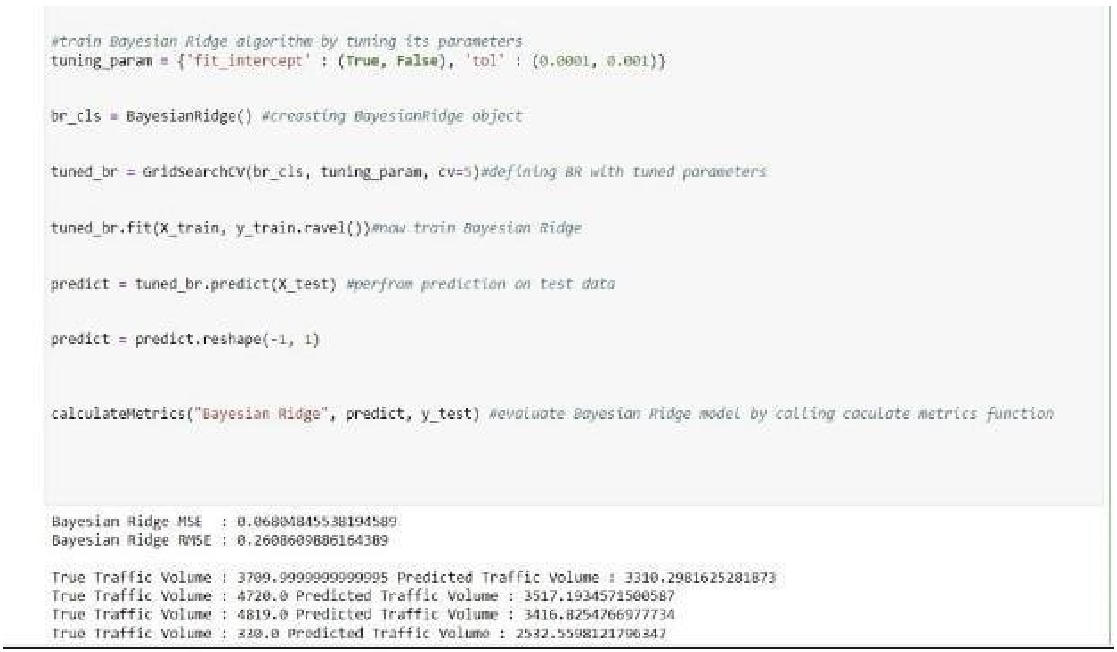
Figure : Implementation of Random Forest Model

This code is all about implementing a Random Forest model. First, we give it some options, like different tools it can use to make predictions. For instance, we tell it how many prediction "trees" to use (either 20, 50, or 100) and which traffic features to consider ('sqrt' or 'log2').Next we create Random forest regressor object using the function RandomForestRegressor().

Then, we want to make our program even smarter, so we use a special technique called "GridSearchCV." This technique helps us try out all the different combinations of tools we mentioned earlier to figure out which combination works the best. We also check if our program is learning well through some tests (5-fold cross-validation).

Once our smart program has learned a lot about traffic from the training data, we ask it to predict what traffic will be like in new situations. It gives us its predictions, which we prepare for further analysis. Finally, a custom function, calculateMetrics, is called to assess the model's performance, typically by calculating metrics such as Mean Absolute Error (MAE) or Mean Squared Error (MSE) to gauge how well the Random Forest model predicts the target values in y_test. To construct decision trees and random forests from discrete sequences, the feature-based strategy is typically employed in which each feature corresponds to a subsequence . Such a feature-based method is composed of two steps: (1) each sequence is first transformed into a feature vector and (2) then standard classification methods for vectorial data are applied. Therefore, the feature extraction step is critical to the success of such feature-based methods.

IMPLEMENTATION OF BAYESIAN RIDGE REGRESSION:



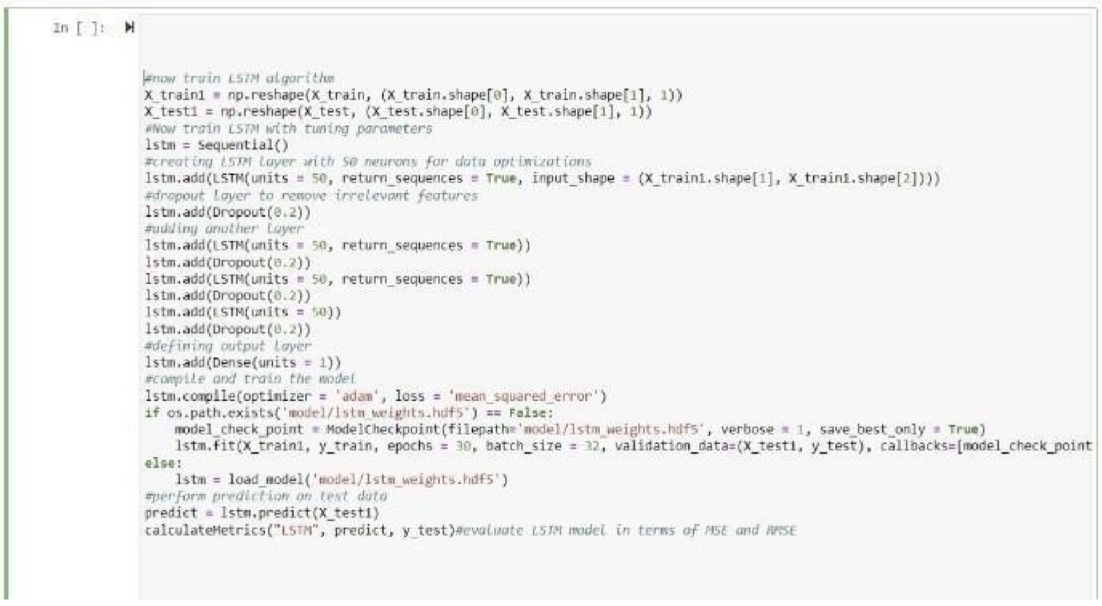
```
#train Bayesian Ridge algorithm by tuning its parameters
tuning_param = {'fit_intercept' : (True, False), 'tol' : (0.0001, 0.001)}

br_cls = BayesianRidge() #creasting BayesianRidge object

tuned_br = GridSearchCV(br_cls, tuning_param, cv=5)#defining BR with tuned parameters

tuned_br.fit(X_train, y_train.ravel())#now train Bayesian Ridge

predict = tuned_br.predict(X_test) #perfrom prediction on test data

predict = predict.reshape(-1, 1)


calculateMetrics("Bayesian Ridge", predict, y_test) #evaluate Bayesian Ridge model by calling caculate metrics function
```

```
Bayesian Ridge MSE  : 0.06804845538194589
Bayesian Ridge RMSE : 0.2608609886164389

True Traffic Volume : 3709.9999999999995 Predicted Traffic Volume : 3310.2981625281873
True Traffic Volume : 4720.0 Predicted Traffic Volume : 3517.1934571500587
True Traffic Volume : 4819.0 Predicted Traffic Volume : 3416.8254766977734
True Traffic Volume : 330.0 Predicted Traffic Volume : 2532.5598121796347
```

Figure : Implementation of Bayesian Ridge Regression model

This code shows the implementation of Bayesian Ridge Regression model. As like the other models this code using the function calculateMetrics() calculates the MSE and RMSE values of the model.

IMPLEMENTATION OF LSTM.



```
In [ ]:

#now train LSTM algorithm
X_train1 = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test1 = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
#Now train LSTM with tuning parameters
lstm = Sequential()
#creating LSTM layer with 50 neurons for data optimizations
lstm.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train1.shape[1], X_train1.shape[2])))
#dropout layer to remove irrelevant features
lstm.add(Dropout(0.2))
#adding another layer
lstm.add(LSTM(units = 50, return_sequences = True))
lstm.add(Dropout(0.2))
lstm.add(LSTM(units = 50, return_sequences = True))
lstm.add(Dropout(0.2))
lstm.add(LSTM(units = 50))
lstm.add(Dropout(0.2))
#defining output layer
lstm.add(Dense(units = 1))
#compile and train the model
lstm.compile(optimizer = 'adam', loss = 'mean_squared_error')
if os.path.exists('model/lstm_weights.hdf5') == False:
    model_check_point = ModelCheckpoint(filepath='model/lstm_weights.hdf5', verbose = 1, save_best_only = True)
    lstm.fit(X_train1, y_train, epochs = 30, batch_size = 32, validation_data=(X_test1, y_test), callbacks=[model_check_point
else:
    lstm = load_model('model/lstm_weights.hdf5')
#perform prediction on test data
predict = lstm.predict(X_test1)
calculateMetrics("LSTM", predict, y_test)#evaluate LSTM model in terms of MSE and RMSE
```

Figure :Implementation of LSTM

The code is about implementing LSTM model .It involves following steps

1. Data Preparation

2. Creating the LSTM:

- We build the model LSTM.
- We give it 50 "neurons" to understand traffic patterns.
- We add "dropout" layers to keep only the important traffic information and remove the rest.

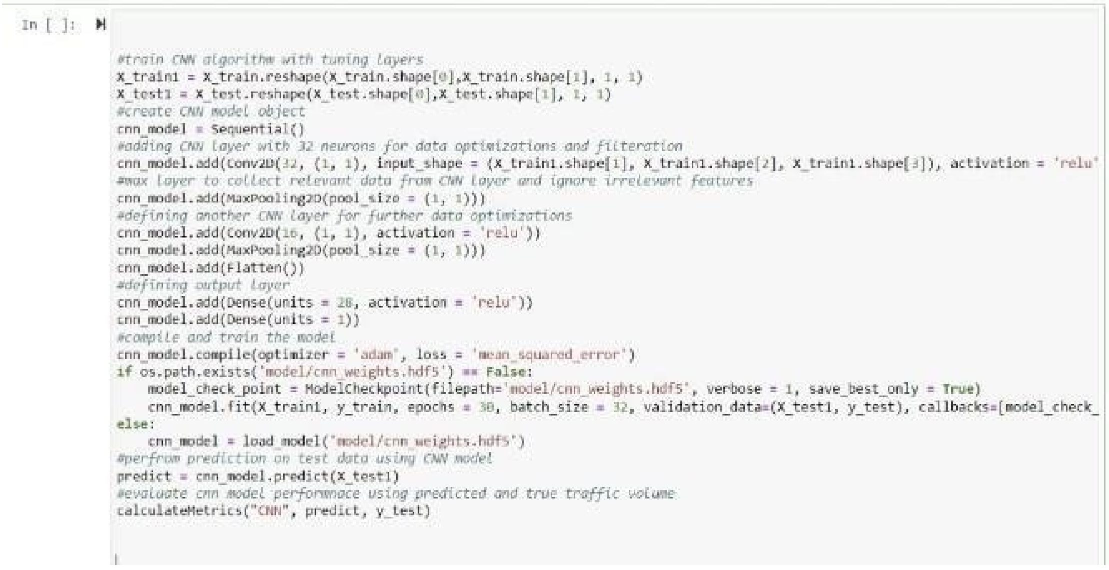
3. Training the Program:

4. Making Predictions:

5. Evaluating Performance**:**

So, in simple terms, this code prepares data, builds a smart traffic predictor (LSTM), trains it to make good traffic predictions, uses it to predict traffic, and then checks how well it's doing by comparing its predictions to real traffic data.

IMPLEMENTATION OF CNN MODEL:



```
In [ ]:

#train CNN algorithm with tuning layers
X_train1 = X_train.reshape(X_train.shape[0],X_train.shape[1], 1, 1)
X_test1 = X_test.reshape(X_test.shape[0],X_test.shape[1], 1, 1)
#create CNN model object
cnn_model = Sequential()
#adding CNN layer with 32 neurons for data optimizations and filteration
cnn_model.add(Conv2D(32, (1, 1), input_shape = (X_train1.shape[1], X_train1.shape[2], X_train1.shape[3]), activation = 'relu'
#max layer to collect relevant data from CNN layer and ignore irrelevant features
cnn_model.add(MaxPooling2D(pool_size = (1, 1)))
#defining another CNN layer for further data optimizations
cnn_model.add(Conv2D(16, (1, 1), activation = 'relu'))
cnn_model.add(MaxPooling2D(pool_size = (1, 1)))
cnn_model.add(Flatten())
#defining output layer
cnn_model.add(Dense(units = 28, activation = 'relu'))
cnn_model.add(Dense(units = 1))
#compile and train the model
cnn_model.compile(optimizer = 'adam', loss = 'mean_squared_error')
if os.path.exists('model/cnn_weights.hdf5') == False:
    model_check_point = ModelCheckpoint(filepath='model/cnn_weights.hdf5', verbose = 1, save_best_only = True)
    cnn_model.fit(X_train1, y_train, epochs = 30, batch_size = 32, validation_data=(X_test1, y_test), callbacks=[model_check_
else:
    cnn_model = load_model('model/cnn_weights.hdf5')
#perfrom prediction on test data using CNN model
predict = cnn_model.predict(X_test1)
#evaluate cnn model performnace using predicted and true traffic volume
calculateMetrics("CNN", predict, y_test)
```

Figure 5.14 Implementation of CNN

The code above involves following steps in implementing CNN :

1. Data Preparation

2. Creating the CNN model:

- We add a "max layer" to collect important information from the CNN layer and ignore the rest.

- We add another layer with 16 "neurons" for further data understanding and optimization.
- We flatten the data to make it ready for the final prediction step.
- We define the output layer that provides the traffic predictions.

3. Training the Program:

4. Making Predictions:

5. Evaluating Performance:

So, in simple terms, this code prepares data, builds a CNN model, trains it to make good traffic predictions, uses it to predict traffic, and then checks how well it's doing by comparing its predictions to real traffic data.

5.5 SUMMARY:

In this chapter, we prepared our data and brought our algorithms to life with practical coding. Our hard work sets the stage for the next chapter, where we'll discuss the results and see how well our models perform. The code prepares data, builds a smart traffic predictor (LSTM), trains it to make good traffic predictions, uses it to predict traffic, and then checks how well it's doing by comparing its predictions to real traffic data. Once our smart program has learned a lot about traffic from the training data, we ask it to predict what traffic will be like in new situations. It gives us its predictions, which we prepare for further analysis. Finally, a custom function, calculateMetrics, is called to assess the model's performance, typically by calculating metrics such as Mean Absolute Error (MAE) or Mean Squared Error (MSE) to gauge how well the Random Forest model predicts the target values in y_test.

6. SYSTEM TESTING

6.1 INTRODUCTION:

The aim of this chapter is to conduct a performance evaluation of traffic congestion prediction approaches used in this research. We want to know how accurate and dependable these methods are in making predictions.

To do this, we use two common measures to evaluate their performance: Root Mean Squared Error (RMSE) and Mean Squared Error (MSE). These measures help us see how close our predictions are to the real data. By using RMSE and MSE, we can objectively figure out which prediction methods are good and which ones might need improvement when it comes to forecasting traffic congestion.

6.2 FUNCTIONAL TESTING:

System testing for a traffic flow prediction system involves verifying and validating the system's functionality, performance, reliability, and usability. Here's an overview of how system testing can be conducted for such a system.

Prediction Accuracy: Verify that the system accurately predicts traffic flow under various conditions, including different times of day, weather conditions, and traffic patterns.

Feature Testing: Ensure that all features of the system, such as data collection, preprocessing, model training, prediction generation, and feedback integration, work as expected.

Boundary Testing: Test the system's behavior at the boundaries of input ranges to ensure robustness and accuracy.

Integration Testing: erify the interaction and integration of different system components, such as data sources, prediction models, and user interfaces.

6.3 RELIABILITY TESTING:

Stability: Verify that the system operates reliably over extended periods without crashing or exhibiting unexpected behavior.

Fault Tolerance: Test the system's ability to recover gracefully from failures, such as data source failures or model errors, without disrupting overall operation.

Error Handling: Validate the system's error handling mechanisms, such as logging, alerting, and graceful degradation, to ensure robustness in handling unexpected situations.

6.4 USABILITY TESTING :

User Interface: Evaluate the user interface for clarity, intuitiveness, and ease of use, ensuring that users can interact with the system effectively.

Feedback Mechanism: Test the system's feedback mechanisms, such as providing predictions, alerts, and recommendations, to ensure they are informative and actionable for users.

Accessibility: Ensure that the system is accessible to users with disabilities and complies with accessibility standards and guidelines.

6.5 INTEGRATION TESTING:

Integration testing is a software testing technique used to verify the interactions between different components or modules of a system when they are integrated together. The primary goal of integration testing is to ensure that the individual components work together as expected and that the integrated system functions correctly as a whole. Here's a more detailed explanation.

Top-Down-Testing: In this approach, testing starts from the top-level modules (usually the main or higher-level modules) and progresses downwards, with lower-level modules being integrated and tested incrementally. Stubs or drivers may be used to simulate the behavior of lower-level modules that are not yet implemented.

Bottom-up-Testing: This approach begins with testing the lower-level modules first and gradually integrates higher-level modules, moving upwards in the system hierarchy. Mock objects or harnesses may be used to simulate the behavior of higher-level modules that are not yet available.

Big Bang Integration Testing: In this approach, all components are integrated simultaneously, and the entire system is tested as a whole. This approach is typically used when the system's components are relatively independent and do not have complex interactions.

Incremental Integration Testing: Incremental integration testing involves integrating and testing small groups of components or modules incrementally. This approach allows for early detection of integration issues and provides more granular control over the testing process.

6.6 PERFORMANCE EVALUATION OF TRAFFIC PREDICTION MODELS.

After completion of any model, it is necessary to evaluate that model. Evaluation metrics are essential in machine learning and deep learning tasks.

Evaluation Parameters:

Evaluation metrics are helpful to check the effectiveness of the prediction model. In this section, two important metrics of prediction models are discussed.

Mean Squared ERROR (MSE):

Mean squared error (MSE) measures the amount of error in statistical models. It assesses the average squared difference between the observed and predicted values. When a model has no error, the MSE equals zero. As model error increases, its value increases. The mean squared error is also known as the mean squared deviation (MSD).

$$MSE = \frac{\sum(y_i - \hat{y}_i)^2}{n}$$

- y_i is the i^{th} observed value.
- $\hat{y}_i$ is the corresponding predicted value.
- n = the number of observations.

Rooted Mean Squared Error (RMSE):

The root mean square error (RMSE) is a crucial metric in predictive modeling, where its value indicates how well a model performs. RMSE measures the average difference between a statistical model's predicted values and the actual values. Mathematically, it is the standard deviation of the residuals. Residuals represent the distance between the regression line and the data points.
RMSE quantifies how dispersed these residuals are, revealing how tightly the observed data clusters around the predicted values. As the data points move closer to the regression line, the model has less error, lowering the RMSE. A model with less error produces more precise predictions.

RMSE values can range from zero to positive infinity and use the same units as the dependent (outcome) variable. A value of 0 means that the predicted values perfectly match the actual values. Low RMSE values indicate that the model fits the data well and has more precise predictions.
Conversely, higher values suggest more error and less precise predictions. Mathematically, it is the standard deviation of the residuals. Residuals represent the distance between the regression line and the data points.

RSME Formula:

The RSME formula for a sample is the following:

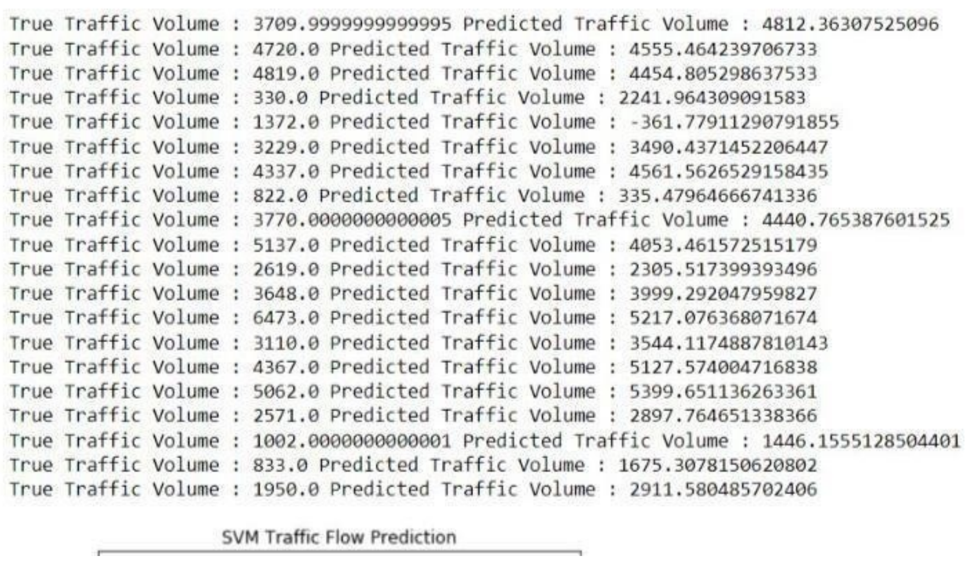
$$RSME = \sqrt{\frac{\sum(y_i - \hat{y}_i)^2}{N - P}}$$

Where:

- y_i is the actual value for the ith observation.
- $\hat{y}_i$ is the predicted value for the ith observation.
- N is the number of observations.
- P is the number of parameter estimates, including the constant.

Experimental Results for SVM:

In this section, we're going to talk about how well the SVM algorithm performed. Figure 5.1 shows a visual comparison between the traffic predictions made by the SVM model and the actual traffic volume. Figure 5.2 gives you a graph that helps you see how close the SVM predictions (green line) are to the real traffic counts (red line) across different test data points. And in Table 5.1, we've put the numbers that tell us how accurate the SVM model is. These numbers, called MSE and RMSE, help us understand how close the model's predictions are to the real traffic counts.



```
True Traffic Volume : 3709.9999999999995 Predicted Traffic Volume : 4812.36307525096
True Traffic Volume : 4720.0 Predicted Traffic Volume : 4555.464239706733
True Traffic Volume : 4819.0 Predicted Traffic Volume : 4454.805298637533
True Traffic Volume : 330.0 Predicted Traffic Volume : 2241.964309091583
True Traffic Volume : 1372.0 Predicted Traffic Volume : -361.77911290791855
True Traffic Volume : 3229.0 Predicted Traffic Volume : 3490.4371452206447
True Traffic Volume : 4337.0 Predicted Traffic Volume : 4561.5626529158435
True Traffic Volume : 822.0 Predicted Traffic Volume : 335.47964666741336
True Traffic Volume : 3770.0000000000005 Predicted Traffic Volume : 4440.765387601525
True Traffic Volume : 5137.0 Predicted Traffic Volume : 4053.461572515179
True Traffic Volume : 2619.0 Predicted Traffic Volume : 2305.517399393496
True Traffic Volume : 3648.0 Predicted Traffic Volume : 3999.292047959827
True Traffic Volume : 6473.0 Predicted Traffic Volume : 5217.076368071674
True Traffic Volume : 3110.0 Predicted Traffic Volume : 3544.1174887810143
True Traffic Volume : 4367.0 Predicted Traffic Volume : 5127.574004716838
True Traffic Volume : 5062.0 Predicted Traffic Volume : 5399.651136263361
True Traffic Volume : 2571.0 Predicted Traffic Volume : 2897.764651338366
True Traffic Volume : 1002.0000000000001 Predicted Traffic Volume : 1446.1555128504401
True Traffic Volume : 833.0 Predicted Traffic Volume : 1675.3078150620802
True Traffic Volume : 1950.0 Predicted Traffic Volume : 2911.580485702406
```

Figure : SVM Model output

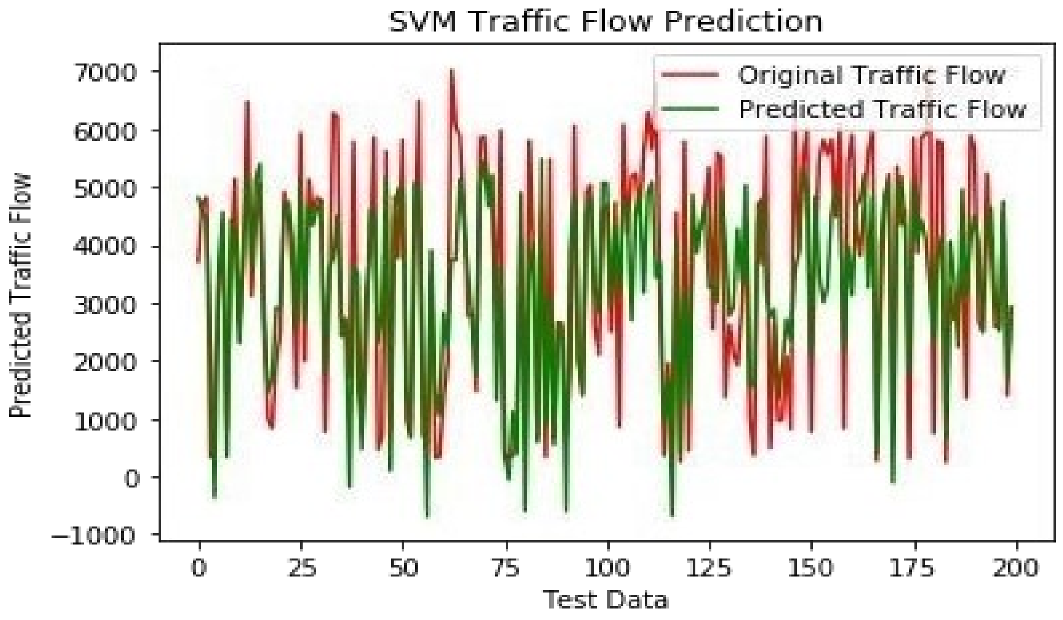


Figure SVM Traffic Flow Prediction

SVM MODEL	
MSE	0.0279761959217717
RMSE	0.16726086189474124

Table 6.1 SVM Model

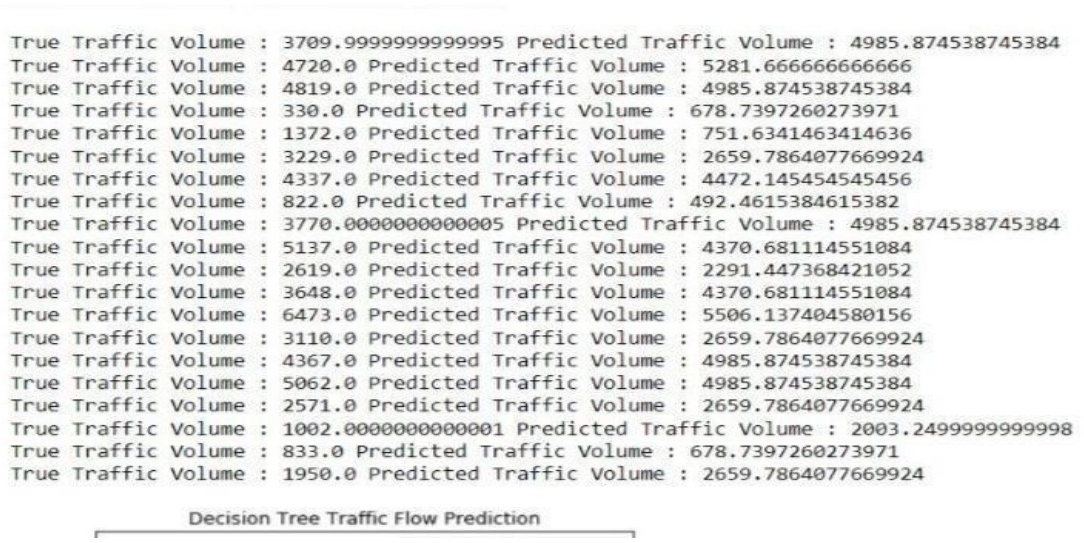
Looking at the graphs in Figure 5.2, you can see that the green line (SVM predictions) and the red line (actual traffic) are really close, almost overlapping. This means the Support Vector Machine (SVM) algorithm is pretty good at predicting traffic.

To put it in numbers, the model gave us an MSE (Mean Squared Error) of 0.0279 and an RMSE (Root Mean Squared Error) of 0.1673. These numbers are small, which is a good sign. It means that, on average, the SVM predictions are very close to the actual traffic counts.

Now, we'll go on to talk about each algorithm we used and see how well they performed one by one.

Experimental Results for DT:

In this section, we're going to talk about how well the DT algorithm performed. Figure 5.3 shows a visual comparison between the traffic predictions made by the DT model and the actual traffic volume. Figure 5.4 gives you a graph that helps you see how close the DT predictions (green line) are to the real traffic counts (red line) across different test data points. Table 5.2, shows MSE and RMSE values, help us understand how close the model's predictions are to the real traffic counts.



```
True Traffic Volume : 3709.9999999999995 Predicted Traffic Volume : 4985.874538745384
True Traffic Volume : 4720.0 Predicted Traffic Volume : 5281.666666666666
True Traffic Volume : 4819.0 Predicted Traffic Volume : 4985.874538745384
True Traffic Volume : 330.0 Predicted Traffic Volume : 678.7397260273971
True Traffic Volume : 1372.0 Predicted Traffic Volume : 751.6341463414636
True Traffic Volume : 3229.0 Predicted Traffic Volume : 2659.7864077669924
True Traffic Volume : 4337.0 Predicted Traffic Volume : 4472.145454545456
True Traffic Volume : 822.0 Predicted Traffic Volume : 492.4615384615382
True Traffic Volume : 3770.0000000000005 Predicted Traffic Volume : 4985.874538745384
True Traffic Volume : 5137.0 Predicted Traffic Volume : 4370.681114551084
True Traffic Volume : 2619.0 Predicted Traffic Volume : 2291.447368421052
True Traffic Volume : 3648.0 Predicted Traffic Volume : 4370.681114551084
True Traffic Volume : 6473.0 Predicted Traffic Volume : 5506.137404580156
True Traffic Volume : 3110.0 Predicted Traffic Volume : 2659.7864077669924
True Traffic Volume : 4367.0 Predicted Traffic Volume : 4985.874538745384
True Traffic Volume : 5062.0 Predicted Traffic Volume : 4985.874538745384
True Traffic Volume : 2571.0 Predicted Traffic Volume : 2659.7864077669924
True Traffic Volume : 1002.0000000000001 Predicted Traffic Volume : 2003.2499999999998
True Traffic Volume : 833.0 Predicted Traffic Volume : 678.7397260273971
True Traffic Volume : 1950.0 Predicted Traffic Volume : 2659.7864077669924
```

Figure : Visual comparison of predictions made by the DT model

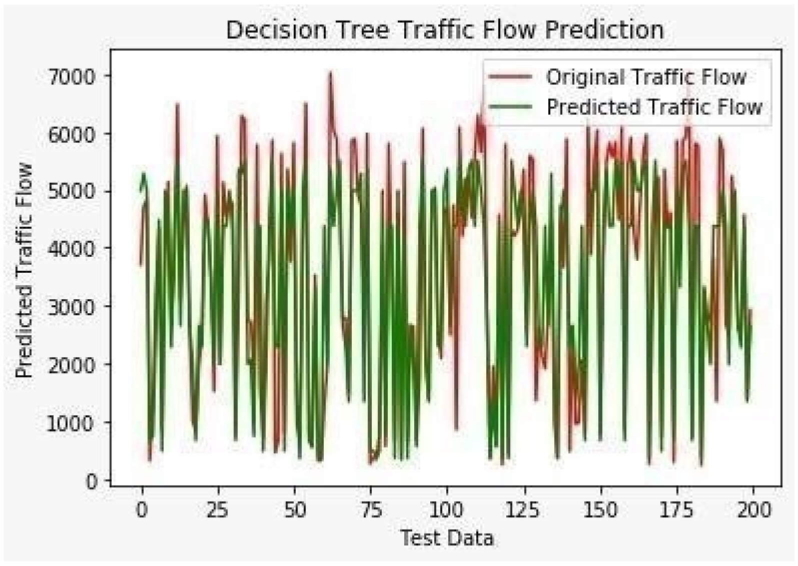


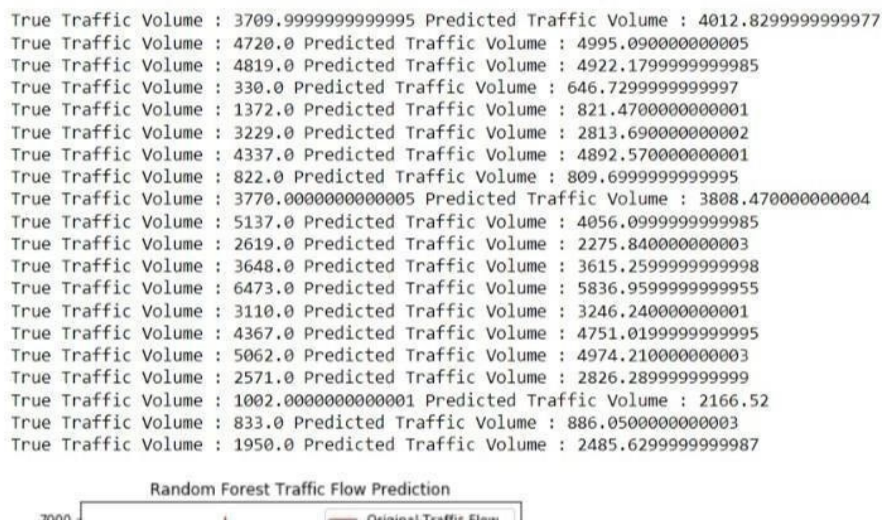
Figure 6.4 DT Traffic Flow Prediction

DT MODEL	
MSE	0.01939865017602931
RMSE	0.13927903710188877

Table 5.2 DT Model

Experimental Results for Random Forest:

In this section, we're going to talk about how well the Random Forest algorithm performed. Figure 5.5 shows a visual comparison between the traffic predictions made by the Random Forest model and the actual traffic volume. Figure 5.6 gives you a graph that helps you see how close the SVM predictions (green line) are to the real traffic counts (red line) across different test data points. And in Table 5.3, we've put the MSE and RMSE values that tell us how accurate the Random Forest model is. Random forest (RF) is a classical machine learning model, and many variants have been proposed to improve the performance or interpretability in recent years. To improve the classification performance and interpretability of RF under the premise of consistency, a novel RF variant named intervention correlation ratio random forest (ICR ^{2} F) is proposed. First, intervention correlation ratio (ICR) is proposed as a novel causality evaluation method by the ratio of pre- and post intervention on features which is used to select features and thresholds to divide a non-leaf node when building a decision tree.



```
True Traffic Volume : 3709.9999999999995 Predicted Traffic Volume : 4012.8299999999977
True Traffic Volume : 4720.0 Predicted Traffic Volume : 4995.090000000005
True Traffic Volume : 4819.0 Predicted Traffic Volume : 4922.1799999999985
True Traffic Volume : 330.0 Predicted Traffic Volume : 646.7299999999997
True Traffic Volume : 1372.0 Predicted Traffic Volume : 821.4700000000001
True Traffic Volume : 3229.0 Predicted Traffic Volume : 2813.690000000002
True Traffic Volume : 4337.0 Predicted Traffic Volume : 4892.570000000001
True Traffic Volume : 822.0 Predicted Traffic Volume : 809.6999999999995
True Traffic Volume : 3770.0000000000005 Predicted Traffic Volume : 3808.470000000004
True Traffic Volume : 5137.0 Predicted Traffic Volume : 4056.0999999999985
True Traffic Volume : 2619.0 Predicted Traffic Volume : 2275.840000000003
True Traffic Volume : 3648.0 Predicted Traffic Volume : 3615.2599999999998
True Traffic Volume : 6473.0 Predicted Traffic Volume : 5836.9599999999955
True Traffic Volume : 3110.0 Predicted Traffic Volume : 3246.240000000001
True Traffic Volume : 4367.0 Predicted Traffic Volume : 4751.0199999999995
True Traffic Volume : 5062.0 Predicted Traffic Volume : 4974.210000000003
True Traffic Volume : 2571.0 Predicted Traffic Volume : 2826.289999999999
True Traffic Volume : 1002.0000000000001 Predicted Traffic Volume : 2166.52
True Traffic Volume : 833.0 Predicted Traffic Volume : 886.0500000000003
True Traffic Volume : 1950.0 Predicted Traffic Volume : 2485.6299999999987
```

Figure 5.5 Traffic predictions made by the Random Forest model

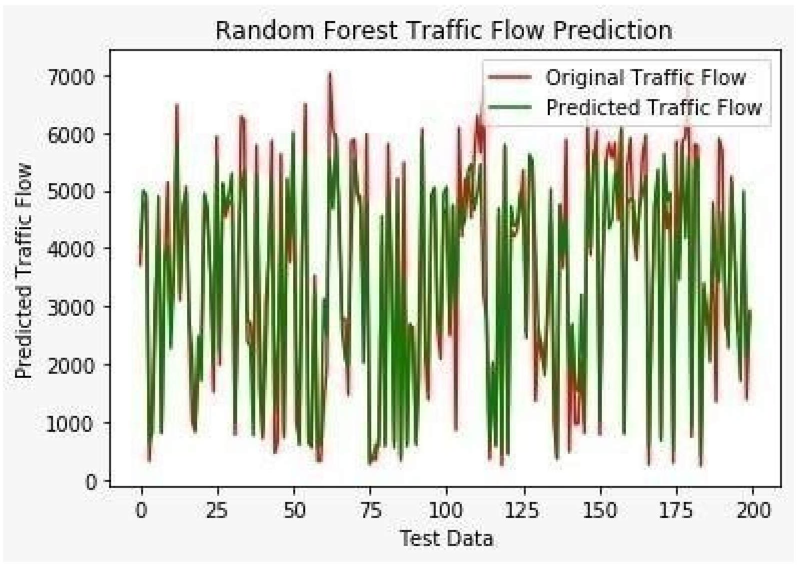


Figure 6.6 Random Forest Traffic Flow Prediction

Random Forest model	
MSE	0.00835110161032001
RMSE	0.0913843619571752

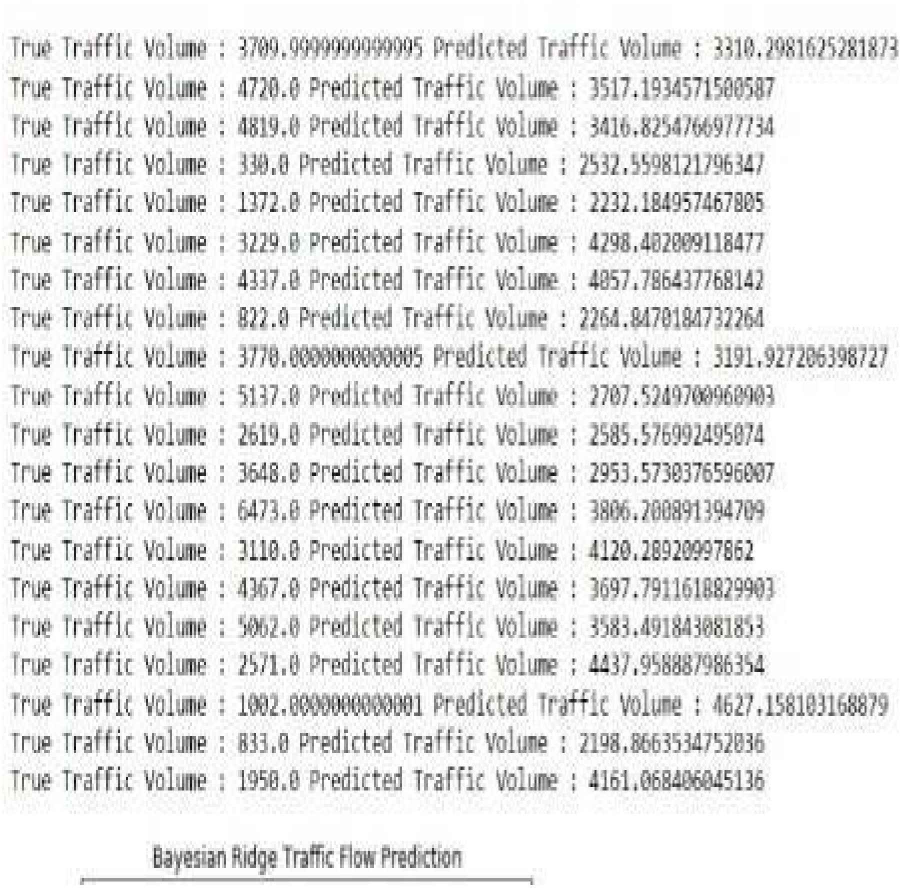
Table: Random Forest Model

The Random Forest algorithm really stood out in our traffic prediction. It made even fewer mistakes compared to the SVM and Decision Tree models. When we looked at the graphs in Figure 5.2, the predicted traffic (the green line) was incredibly close to the actual traffic (the red line), showing how well the model understands traffic patterns.

In numbers, the Random Forest had impressively low error values with an MSE of 0.0084 and an RMSE of 0.0914. These numbers tell us that the model's predictions are super close to the real traffic counts.

Experimental Results for Bayesian algorithm:

In this section, we're going to talk about how well the Bayesian Ridge algorithm performed. Figure 5.7 shows a visual comparison between the traffic predictions made by the model and the actual traffic volume. Figure 5.8 gives you a graph that helps you see how close the BA's predictions (green line) are to the real traffic counts (red line) across different test data points. Table 5.4 shows the MSE and RMSE values.



```
True Traffic Volume : 3709.9999999999995 Predicted Traffic Volume : 3310.2981625281873
True Traffic Volume : 4720.0 Predicted Traffic Volume : 3517.1934571500587
True Traffic Volume : 4819.0 Predicted Traffic Volume : 3416.8254766977734
True Traffic Volume : 330.0 Predicted Traffic Volume : 2532.5598121796347
True Traffic Volume : 1372.0 Predicted Traffic Volume : 2232.184957467805
True Traffic Volume : 3229.0 Predicted Traffic Volume : 4298.402009118477
True Traffic Volume : 4337.0 Predicted Traffic Volume : 4857.786437768142
True Traffic Volume : 822.0 Predicted Traffic Volume : 2264.8470184732264
True Traffic Volume : 3770.0000000000005 Predicted Traffic Volume : 3191.927206398727
True Traffic Volume : 5137.0 Predicted Traffic Volume : 2707.5249700960903
True Traffic Volume : 2619.0 Predicted Traffic Volume : 2585.576992495074
True Traffic Volume : 3648.0 Predicted Traffic Volume : 2953.5730376596007
True Traffic Volume : 6473.0 Predicted Traffic Volume : 3806.200891394709
True Traffic Volume : 3110.0 Predicted Traffic Volume : 4120.28920997862
True Traffic Volume : 4367.0 Predicted Traffic Volume : 3697.7911618829903
True Traffic Volume : 5062.0 Predicted Traffic Volume : 3583.491843081853
True Traffic Volume : 2571.0 Predicted Traffic Volume : 4437.958887986354
True Traffic Volume : 1002.0000000000001 Predicted Traffic Volume : 4627.158103168879
True Traffic Volume : 833.0 Predicted Traffic Volume : 2198.8663534752036
True Traffic Volume : 1950.0 Predicted Traffic Volume : 4161.068406045136
```

Bayesian Ridge Traffic Flow Prediction

Figure: Traffic predictions made by Bayesian Ridge Model

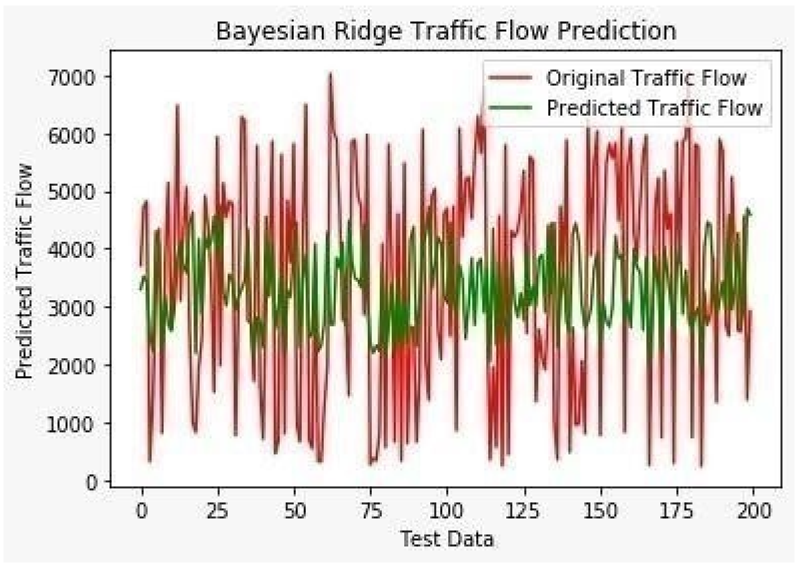


Figure : Bayesian Ridge Traffic Flow Prediction

Bayesian Ridge Model	
MSE	0.68048455381
RMSE	0.26088616

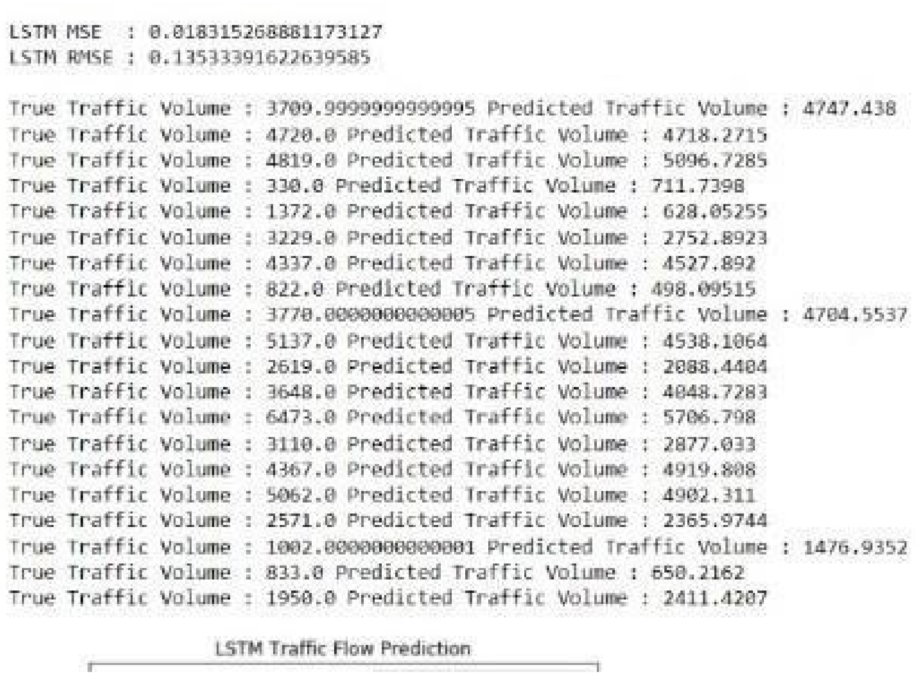
Table:Bayesian Ridge Model

The Bayesian Ridge algorithm did a decent job at predicting traffic, but it had slightly higher errors compared to some other models we used. When we looked at the graphs, the predicted traffic (the green line) was reasonably close to the actual traffic (the red line) in Figure 5.2, showing that it understands traffic patterns.

In numbers, the Bayesian Ridge had an MSE of 0.0680 and an RMSE of 0.2609. These values indicate that its predictions are quite close to the real traffic counts, but there's a bit more room for improvement compared to the other models**.**

6.6.5 Experimental Results for LSTM:

In this part, we'll discuss how well the LSTM algorithm performed. Figure 5.9 depicts a visual comparison between the LSTM model's traffic forecasts and actual traffic volume. Figure 5.10 depicts a graph that shows how similar the LSTM predictions (green line) are to the actual traffic counts (red line) at various test data points. The MSE and RMSE values are shown in Table 5.5.



```
LSTM MSE  : 0.018315268881173127
LSTM RMSE : 0.13533391622639585

True Traffic Volume : 3709.9999999999995 Predicted Traffic Volume : 4747.438
True Traffic Volume : 4720.0 Predicted Traffic Volume : 4718.2715
True Traffic Volume : 4819.0 Predicted Traffic Volume : 5096.7285
True Traffic Volume : 330.0 Predicted Traffic Volume : 711.7398
True Traffic Volume : 1372.0 Predicted Traffic Volume : 628.05255
True Traffic Volume : 3229.0 Predicted Traffic Volume : 2752.8923
True Traffic Volume : 4337.0 Predicted Traffic Volume : 4527.892
True Traffic Volume : 822.0 Predicted Traffic Volume : 498.09515
True Traffic Volume : 3770.0000000000005 Predicted Traffic Volume : 4704.5537
True Traffic Volume : 5137.0 Predicted Traffic Volume : 4538.1064
True Traffic Volume : 2619.0 Predicted Traffic Volume : 2088.4404
True Traffic Volume : 3648.0 Predicted Traffic Volume : 4048.7283
True Traffic Volume : 6473.0 Predicted Traffic Volume : 5706.798
True Traffic Volume : 3110.0 Predicted Traffic Volume : 2877.033
True Traffic Volume : 4367.0 Predicted Traffic Volume : 4919.808
True Traffic Volume : 5062.0 Predicted Traffic Volume : 4902.311
True Traffic Volume : 2571.0 Predicted Traffic Volume : 2365.9744
True Traffic Volume : 1002.0000000000001 Predicted Traffic Volume : 1476.9352
True Traffic Volume : 833.0 Predicted Traffic Volume : 650.2162
True Traffic Volume : 1950.0 Predicted Traffic Volume : 2411.4207
```

Figure 6.9 LSTM model traffic forecasts

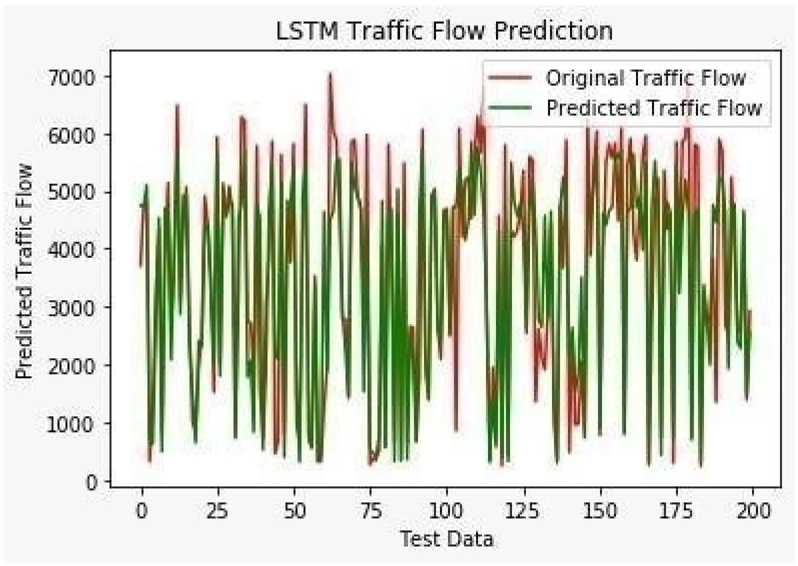


Figure:LSTM Traffic flow Prediction

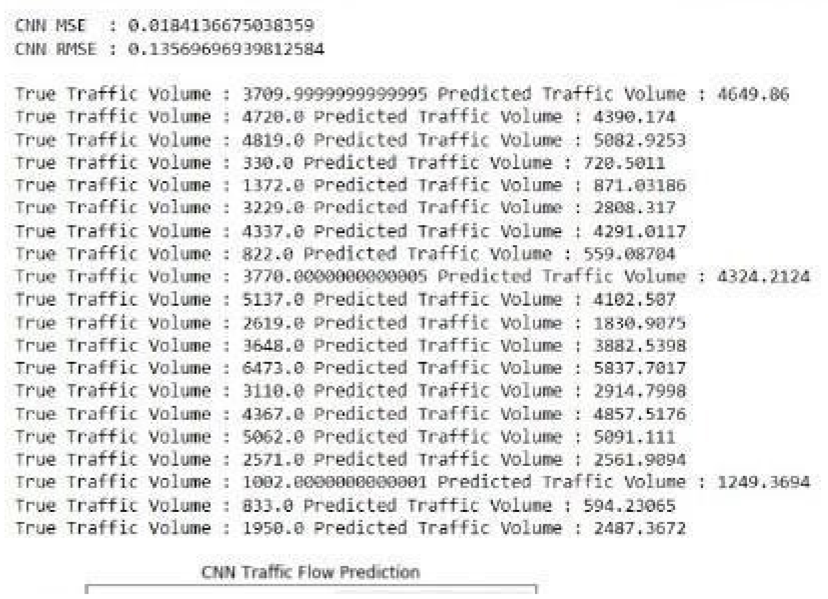
The LSTM algorithm has demonstrated promising performance in our traffic prediction task. In Figure 5.2, we observed a close alignment between the predicted traffic values (green line) and the actual traffic volume (red line), indicating its ability to effectively capture traffic patterns.

Quantitatively, the computed Mean Squared Error (MSE) for LSTM was 0.0183, with a Root Mean Squared Error (RMSE) of 0.1353. These values reflect the model's capability to provide accurate predictions, with low prediction errors. LSTM algorithm has proven its effectiveness in traffic forecasting, achieving noteworthy accuracy in our study.

However, quantitatively, the LSTM model had a somewhat higher Mean Squared Error (MSE) of 0.0183 and Root Mean Squared Error (RMSE) of 0.1353 compared to Random Forest.

Experimental Results for CNN:

In this part, we'll discuss how well the CNN algorithm performed. Figure 5.11 depicts a visual comparison between the CNN model's traffic forecasts and actual traffic volume. Figure 5.12 depicts a graph that shows how similar the LSTM predictions (green line) are to the actual traffic counts (red line) at various test data points. The MSE and RMSE values are shown in Table 5.6.



```
CNN MSE  : 0.0184136675038359
CNN RMSE : 0.13569696939812584

True Traffic Volume : 3709.9999999999995 Predicted Traffic Volume : 4649.86
True Traffic Volume : 4720.0 Predicted Traffic Volume : 4390.174
True Traffic Volume : 4819.0 Predicted Traffic Volume : 5082.9253
True Traffic Volume : 330.0 Predicted Traffic Volume : 720.5011
True Traffic Volume : 1372.0 Predicted Traffic Volume : 871.03186
True Traffic Volume : 3229.0 Predicted Traffic Volume : 2808.317
True Traffic Volume : 4337.0 Predicted Traffic Volume : 4291.0117
True Traffic Volume : 822.0 Predicted Traffic Volume : 559.08704
True Traffic Volume : 3770.0000000000005 Predicted Traffic Volume : 4324.2124
True Traffic Volume : 5137.0 Predicted Traffic Volume : 4102.507
True Traffic Volume : 2619.0 Predicted Traffic Volume : 1830.9075
True Traffic Volume : 3648.0 Predicted Traffic Volume : 3882.5398
True Traffic Volume : 6473.0 Predicted Traffic Volume : 5837.7017
True Traffic Volume : 3110.0 Predicted Traffic Volume : 2914.7998
True Traffic Volume : 4367.0 Predicted Traffic Volume : 4857.5176
True Traffic Volume : 5062.0 Predicted Traffic Volume : 5091.111
True Traffic Volume : 2571.0 Predicted Traffic Volume : 2561.9094
True Traffic Volume : 1002.0000000000001 Predicted Traffic Volume : 1249.3694
True Traffic Volume : 833.0 Predicted Traffic Volume : 594.23065
True Traffic Volume : 1950.0 Predicted Traffic Volume : 2487.3672
```

Figure: CNN model's traffic forecasts

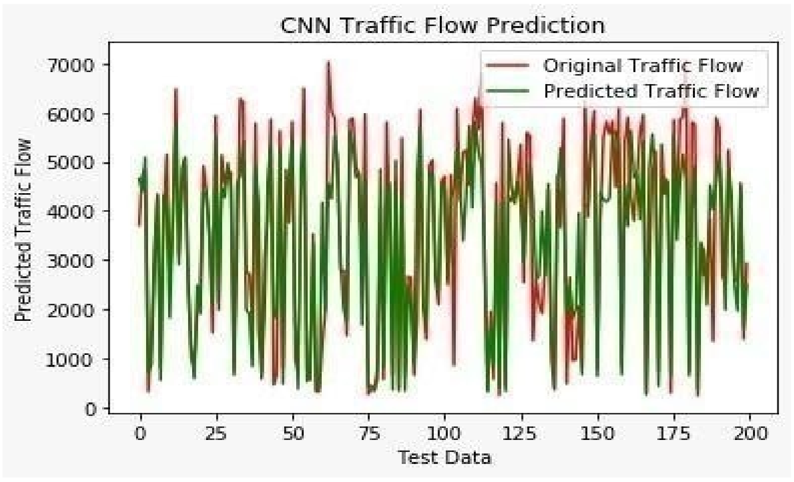


Figure :CNN Traffic Flow Prediction

CNN Model	
MSE	0.0184136675038359
RMSE	0.13569696939812584

Table 6.6 CNN Model

The CNN (Convolutional Neural Network) algorithm has shown promising performance in our traffic prediction task.
Quantitatively, the computed Mean Squared Error (MSE) for CNN was 0.0184, with a Root Mean Squared Error (RMSE) of 0.1357. These values demonstrate the model's capability to provide accurate traffic predictions with relatively low errors.

COMPARISION:

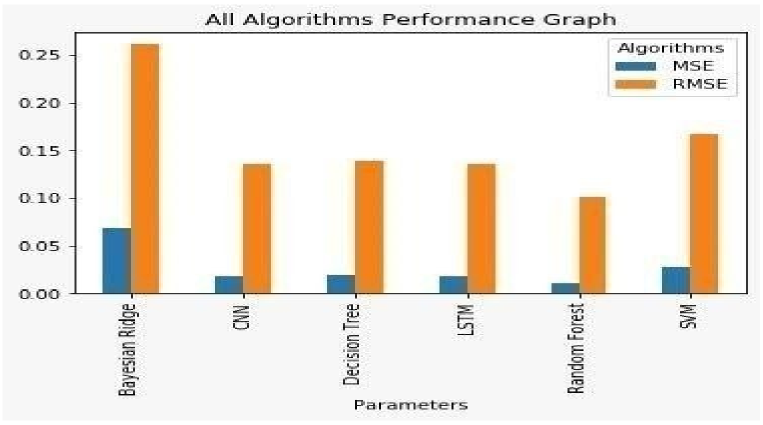


Figure 6.13 All Algorithms Performance Graph

The bar chart (Fig.5.13) presented above illustrates the Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) values for each algorithm. On this chart, the x-axis represents the different algorithms or parameters, while the y-axis indicates the corresponding error values .

Algorithm Name	**MSE**	**RMSE**
Random Forest	0.00835110161032001	0.0913843619571752
LSTM	0.018315268881173127	0.13533391622639585

CNN	0.0184136675038359	0.13569696939812584
Decision Tree	0.01939865017602931	0.13927903710188877
SVM	0.0279761959217717	0.16726086189474124
Bayesian Algorithm	0.06804845538194589	0.2608609886164389

Table 6.7 All Algorithms Performance

The table provided above offers a comprehensive comparison of all the models, presenting their respective Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) values. This tabulated data allows for a concise and clear assessment of the models' performance in terms of predictive accuracy.

7. CONCLUSION

In our comprehensive evaluation of multiple algorithms for traffic prediction, we observed varying degrees of accuracy and effectiveness. The Random Forest algorithm emerged as the standout performer, showcasing the lowest Mean Squared Error (MSE) of 0.0084 and Root Mean Squared Error (RMSE) of 0.0914, indicating exceptional accuracy in predicting traffic patterns. It was closely followed by the LSTM (Long Short-Term Memory) model, with an MSE of 0.0183 and RMSE of 0.1353, demonstrating commendable accuracy.

The Convolutional Neural Network (CNN) also exhibited strong performance, with an MSE of 0.0184 and RMSE of 0.1357, closely trailing the LSTM. The Decision Tree model performed well, with an MSE of 0.0194 and RMSE of 0.1393, showcasing its ability to accurately predict traffic. The Bayesian Ridge algorithm, while proficient, had slightly higher errors with an MSE of 0.0680 and RMSE of 0.2609. Nevertheless, it still provided valuable predictions. Lastly, the Support Vector Machine (SVM) showed reasonable performance with an MSE of 0.0279 and RMSE of 0.1673. The Decision Tree model performed well, with an MSE of 0.0194 and RMSE of 0.1393, showcasing its ability to accurately predict traffic.

In summary, the Random Forest algorithm proved to be the top choice for precise traffic forecasting in our study, closely followed by LSTM and CNN. These models excelled in minimizing prediction errors and demonstrated superior accuracy compared to others.

8.FUTURE WORK

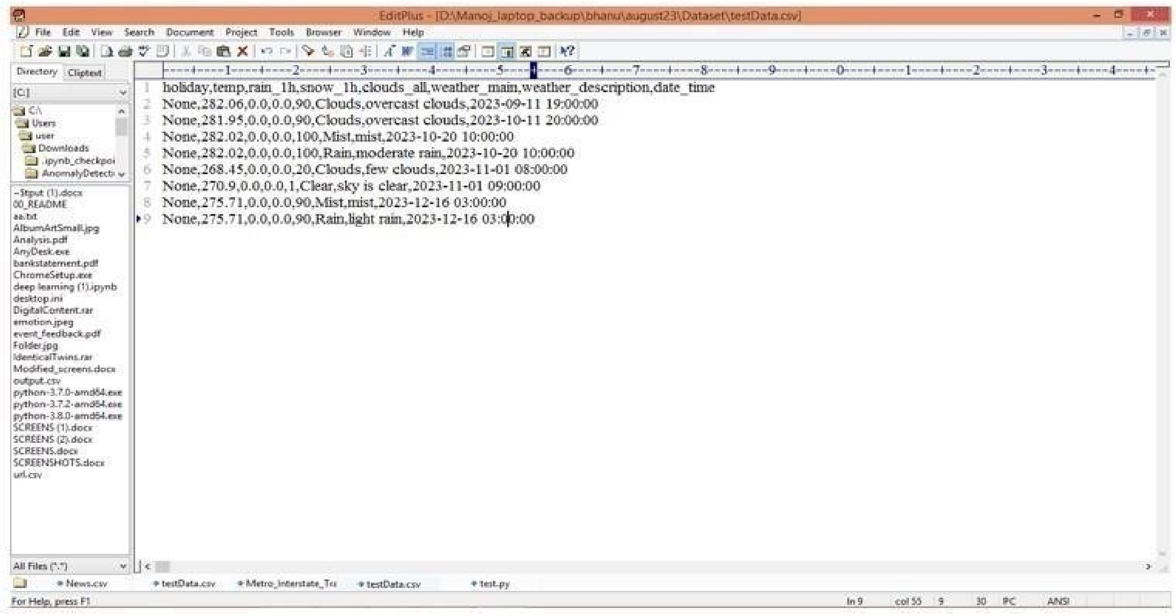
Model Fine-Tuning and Advanced Hybrid Models:

In the ever-evolving field of traffic prediction, continuous improvement of predictive models is essential. Model fine-tuning, a process of optimizing a model's parameters and settings, plays a pivotal role in enhancing prediction accuracy. Fine-tuning allows us to extract the best performance from existing models like Support Vector Machines (SVM), Decision Trees (DT), Bayesian Algorithms, Random Forests, and deep learning models like LSTM and CNN. Additionally, exploring advanced hybrid models that combine the strengths of multiple algorithms is gaining traction. These hybrid approaches can leverage the robustness of traditional machine learning models while harnessing the power of deep learning techniques. By integrating complementary components, hybrid models can potentially provide more accurate and reliable traffic predictions, thus contributing to improved traffic management and enhanced commuter experiences in today's congested urban environments.

Transformer Models (BERT and GPT) in Traffic Prediction:

Transformer models like BERT and GPT, initially designed for language tasks, are now being applied to traffic prediction due to their versatility in capturing complex temporal patterns. These models excel at handling long-range dependencies and contextual information, making them highly effective in forecasting traffic congestion. By training them on historical traffic data, they can discover intricate traffic patterns, resulting in more precise and adaptable predictions. Integrating transformer models into traffic management systems enhances decision-making and responsiveness in dynamic urban environments.

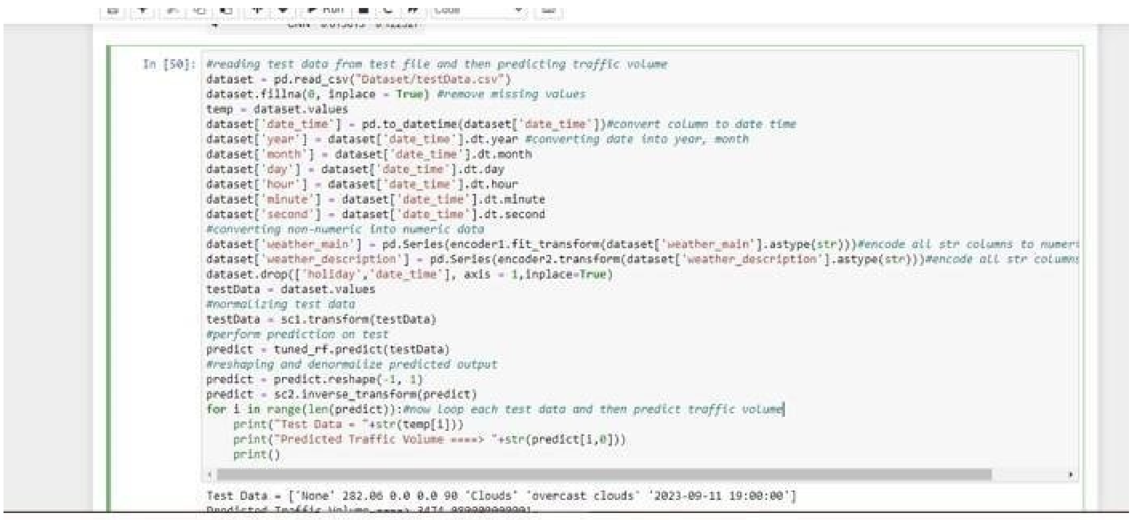
9. APPENDICES



```
holiday,temp,rain_1h,snow_1h,clouds_all,weather_main,weather_description,date_time
None,282.06,0.0,0.0,90,Clouds,overcast clouds,2023-09-11 19:00:00
None,281.95,0.0,0.0,90,Clouds,overcast clouds,2023-10-11 20:00:00
None,282.02,0.0,0.0,100,Mist,mist,2023-10-20 10:00:00
None,282.02,0.0,0.0,100,Rain,moderate rain,2023-10-20 10:00:00
None,268.45,0.0,0.0,20,Clouds,few clouds,2023-11-01 08:00:00
None,270.9,0.0,0.0,1,Clear,sky is clear,2023-11-01 09:00:00
None,275.71,0.0,0.0,90,Mist,mist,2023-12-16 03:00:00
None,275.71,0.0,0.0,90,Rain,light rain,2023-12-16 03:00:00
```

Fig :Test Data CSV

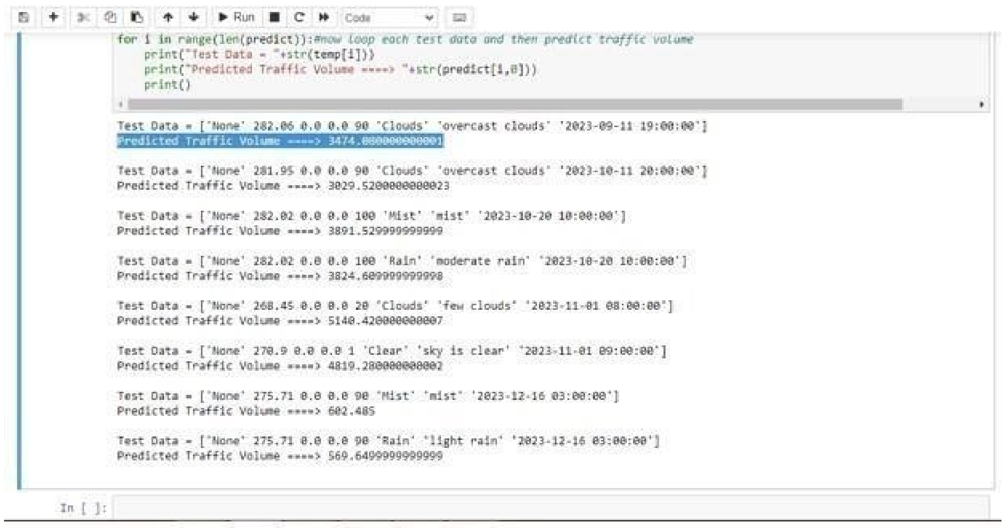
We have created a Manual testing data to perform the traffic prediction .Fig 7.1 shows the data of the csv file consists all the elements needed for prediction including weather conditions .



```
In [50]: #reading test data from test file and then predicting traffic volume
         dataset = pd.read_csv("Dataset/testData.csv")
         dataset.fillna(0, inplace = True) #remove missing values
         temp = dataset.values
         dataset['date_time'] = pd.to_datetime(dataset['date_time'])#convert column to date time
         dataset['year'] = dataset['date_time'].dt.year #converting date into year, month
         dataset['month'] = dataset['date_time'].dt.month
         dataset['day'] = dataset['date_time'].dt.day
         dataset['hour'] = dataset['date_time'].dt.hour
         dataset['minute'] = dataset['date_time'].dt.minute
         dataset['second'] = dataset['date_time'].dt.second
         #converting non-numeric into numeric data
         dataset['weather_main'] = pd.Series(encoder1.fit_transform(dataset['weather_main'].astype(str)))#encode all str columns to numer
         dataset['weather_description'] = pd.Series(encoder2.transform(dataset['weather_description'].astype(str)))#encode all str column
         dataset.drop(['holiday','date_time'], axis = 1,inplace=True)
         testData = dataset.values
         #normalizing test data
         testData = sc1.transform(testData)
         #perform prediction on test
         predict = tuned_rf.predict(testData)
         #reshaping and denormalize predicted output
         predict = predict.reshape(-1, 1)
         predict = sc2.inverse_transform(predict)
         for i in range(len(predict)):#now loop each test data and then predict traffic volume
             print("Test Data = "+str(temp[i]))
             print("Predicted Traffic Volume ====> "+str(predict[i,0]))
             print()

         Test Data = ['None' 282.06 0.0 0.0 90 'Clouds' 'overcast clouds' '2023-09-11 19:00:00']
```

Fig :code used for Random forest model to predict output on manual test data

Fig : shows the code used for prediction . we are giving testData.csv file as input and then preprocessing test data and then calling predict function to predict traffic and after executing above block will get below outpu



```
for i in range(len(predict)):#now loop each test data and then predict traffic volume
    print("Test Data = "+str(temp[i]))
    print("Predicted Traffic Volume ====> "+str(predict[i,0]))
    print()
```

```
Test Data = ['None' 282.06 0.0 0.0 90 'Clouds' 'overcast clouds' '2023-09-11 19:00:00']
Predicted Traffic Volume ====> 3474.0000000000001

Test Data = ['None' 281.95 0.0 0.0 90 'Clouds' 'overcast clouds' '2023-10-11 20:00:00']
Predicted Traffic Volume ====> 3029.5200000000023

Test Data = ['None' 282.02 0.0 0.0 100 'Mist' 'mist' '2023-10-20 10:00:00']
Predicted Traffic Volume ====> 3891.529999999999

Test Data = ['None' 282.02 0.0 0.0 100 'Rain' 'moderate rain' '2023-10-20 10:00:00']
Predicted Traffic Volume ====> 3824.609999999998

Test Data = ['None' 268.45 0.0 0.0 20 'Clouds' 'few clouds' '2023-11-01 08:00:00']
Predicted Traffic Volume ====> 5140.420000000007

Test Data = ['None' 270.9 0.0 0.0 1 'Clear' 'sky is clear' '2023-11-01 09:00:00']
Predicted Traffic Volume ====> 4819.280000000002

Test Data = ['None' 275.71 0.0 0.0 90 'Mist' 'mist' '2023-12-16 03:00:00']
Predicted Traffic Volume ====> 602.485

Test Data = ['None' 275.71 0.0 0.0 90 'Rain' 'light rain' '2023-12-16 03:00:00']
Predicted Traffic Volume ====> 569.6499999999999
```

Fig: Predicted traffic volume for manual test data

In above screen in square bracket we are printing test data values and then in next line after arrow == symbol displaying number of traffic vehicle or traffic volume

10. BIBILIOGRAPHY

[1] Gobezie, Ayele, and Marta Sintayehu Fufa. "Machine learning and deep learning models for traffic flow prediction: A survey." (2020).

[2] Liu, Lingbo, et al. "Dynamic spatial-temporal representation learning for traffic flow prediction." *IEEE Transactions on Intelligent Transportation Systems* 22.11 (2020): 71697183.

[3] Shao, Yanli, et al. "The traffic flow prediction method using the incremental learningbased CNN-LTSM model: the solution of mobile application." *Mobile Information Systems* 2021.

[4] Chen, Zekuan, et al. "Deep reinforcement learning-based LSTM model for traffic flow forecasting in internet of vehicles." *Proceedings of 2021 Chinese Intelligent Automation Conference*. Springer Singapore, 2022.

[5] Aldwyish, A., Tanin, E., Xie, H., Karunasekera, S., & Ramamohanarao, K. "Effective Traffic Forecasting with Multi-Resolution Learning." In *17th International Symposium on Spatial and Temporal Databases* 2021, August). (pp. 44-53).

[6] Nigam, Archana, and Sanjay Srivastava. "Macroscopic Traffic Stream Variable Prediction with Weather Impact using Recurrent Learning Approach." *2020 IEEEHYDCON*. IEEE, 2020.

[7] Tedjopurnomo, David Alexander, et al. "A survey on modern deep neural network for traffic prediction: Trends, methods and challenges." *IEEE Transactions on Knowledge and Data Engineering* 34.4 (2020): 1544-1561.

[8] Kundu, Shounak, Maunendra Sankar Desarkar, and P. K. Srijith. "Traffic forecasting with deep learning." *2020 IEEE Region 10 Symposium (TENSYMP)*. IEEE, 2020.

[9] Rahman, Faysal Ibna. "Short Term Traffic Flow Prediction Using Machine LearningKnn, Svm And Ann With Weather Information." *International Journal for Traffic & Transport Engineering* 10.3 (2020).

[10] Valova, Iren, Natacha Gueorguieva, and Sandeep Smudidonga. "Short-Term Traffic Forecasting Using Deep Learning." Avestia Publ., in Proceedings of the 7th World Congress on Electrical Engineering and Computer Systems and Sciences (EECSS'21) Prague, Czech Republic. 2021.

[11] Ma, Dongfang, Xiang Song, and Pu Li. "Daily traffic flow forecasting through a contextual convolutional recurrent neural network modeling inter-and intra-day traffic patterns." *IEEE Transactions on Intelligent Transportation Systems* 22.5 (2020): 26272636.

[12] Boukerche, Azzedine, and Jiahao Wang. "Machine learning- based traffic prediction models for intelligent transportation systems." *Computer Networks* 181 (2020): 107530.

[14] Lu, Wang., Rui, Wu., Weichun, Ma., Weiju, Xu. (2023). Examining the volatility of soybean market in the MIDAS framework: The importance of bagging-based weather information. International Review of Financial Analysis, doi: 10.1016/j.irfa.2023.102720

[15] Jingyi, Lu. (2023). An efficient and intelligent traffic flow prediction method based on LSTM and variational modal decomposition. Measurement: Sensors, doi: 10.1016/j.measen.2023.100843

[16] Bashir, Mohammed., Nandini, Krishnaswamy., Mariam, Kiran. (2019). Multivariate Time-Series Prediction for Traffic in Large WAN Topology. doi: 10.1109/ANCS.2019.8901870

[17] Hao, Peng., Santosh, U., Bobade., Michael, E., Cotterell., John, A., Miller. (2018). Forecasting Traffic Flow: Short Term, Long Term, and When It Rains. doi: 10.1007/978-3- 319-94301-5_5

[18] M., S., Srinath. (2023). Vehicular Traffic Flow Prediction Model Deep Learning. International Journal For Science Technology And Engineering, doi: 10.22214/ijraset.2023.54576

[19] Guowen, Dai., Jinjun, Tang. (2023). Short-term traffic flow prediction: An ensemble machine learning approach. alexandria engineering journal, doi: 10.1016/j.aej.2023.05.015

[20] Aditya, Srivastava., Aryan. (2023). A Survey Paper on Traffic Prediction Using Machine Learning. International Journal For Science Technology And Engineering,doi: 10.22214/ijraset.2023.51390.

TRAFFIC FLOW PREDICTION USING MACHINE LEARNING

K. SAI MANASWINI
Student
Computer Science and Engineering
Tirumala Engineering College, Narasaraopeta
Email: saimanaswinikancharla2002@gmail.com

K. DEEPTHI
Student
Computer Science and Engineering
Tirumala Engineering College, Narasaraopeta
Email: deepu.kattera25@gmail.com

P.V.SAI ESWAR REDDY
Student
Computer Science and Engineering
Tirumala Engineering College, Narasaraopeta
Email: eswarnani3655@gmail.com

M. SOMA SEKHAR
Student
Computer Science and Engineering
Tirumala Engineering College, Narasaraopeta
Email: somusekhar949@gmail.com

Dr. Lalu Naik. Ph.D
ASST PROFESSOR, DEPARTMENT OF CSE
Tirumala Engineering College, Narasaraopeta
Email: lalunaik12321@gmail.com

***Abstract* – As a result of the ongoing increase in the number of vehicles, there are increasingly severe traffic bottlenecks. The amount of time and money that users of transportation spend traveling will immediately change as a result. To some extent, this issue can be mitigated by projecting future traffic patterns. Three machine learning algorithms—Linear Regression, Decision Tree, and Support Vector Machine—are used in this work to predict the traffic flow after the data has been preprocessed using Selenium, OSS, and Message Queue. Next, we examine real-time Beijing traffic data as it shows up on the Baidu map. According to this study, Random Forest is the most accurate of all three techniques, with an accuracy of 0.719. Second are logistic regression and SVM.**

Keywords - *Traffic FlowPrediction; Decision Tree; Random Forest; SVM; Baysian Ridge; CNN; LSTM*

I. INTRODUCTION

Due to its potential to improve transportation management and lessen urban traffic congestion, the application of deep learning and machine learning to traffic prediction has attracted a lot of attention lately. To close the gaps and lay the groundwork for our project, which aims to close those gaps, we identify key points that each study article may have missed by taking insights from a large number of them.

Weather-based traffic flow prediction is one area of research for intelligent transportation systems. The effectiveness of standard prediction methodologies in estimating short-term traffic flow is limited due to the complexity of the affecting components. Lu Wang et al.'s short-term traffic flow prediction model, which makes use of variational modal decomposition and short- and long-term memory networks, has shown encouraging results in producing accurate forecasts [1].
Jingyi Lu and others Furthermore, it has been discovered that adding meteorological data to traffic flow forecasting models increases forecast accuracy. Research has demonstrated that the accuracy of power price estimates in day-ahead markets can be greatly enhanced by utilizing next-day weather forecasts, indicating that weather forecasts may also be useful in anticipating traffic patterns [2]. Mohammed Bashir et al. Furthermore, it has been discovered that the use of weather variables in soybean volatility forecasting models—such as clear sky index, cloud cover, relative humidity, atmospheric pressure, precipitation, temperature, and wind speed—outperforms models without weather data, demonstrating the weather's predictive power in this area as well [3].

As per the authors' discourse, Wang et al. devised an innovative approach that enhances the precision of traffic flow forecasts through the employment of deep learning methodologies, specifically Long Short-Term Memory (LSTM) neural networks, AdaBoost, and gradient descent.
As per the authors' discourse, Wang et al. devised an innovative approach that enhances the precision of traffic flow forecasts through the employment of deep learning methodologies, specifically Long Short-Term Memory (LSTM) neural networks, AdaBoost, and gradient descent.[4]

Aditya with every one of them The Support Vector Regression approach is a useful tool for traffic flow

prediction, according to a number of studies. Traffic flow forecasts on an hourly, daily, monthly, or even annual basis can be made using it.[5]
Our research aims to advance traffic prediction by providing a thorough methodology that considers several machine learning and deep learning techniques in addition to meteorological factors. In order to accomplish this, we will examine these study publications to identify any gaps in knowledge. We want to fill in the following details in order to develop a trustworthy traffic forecast model that might significantly enhance urban transportation management, reduce traffic, boost road safety, and support a more sustainable urban environment, these gaps in the literature.

II. METHOD

To ascertain the traffic state, we employ datamining as the prediction model. Data mining is a technique that uses specific algorithms to sift through massive amounts of data in search of hidden information. Combining databases, machine learning, artificial intelligence, statistics, and many other topics is known as data mining.
Three steps are involved in data mining:

1) Rata preparing herself. We first define the question and the main problem in this part. After that, the database is set up and the necessary data is collected. We then preprocess the data after that.
2) acquiring information.We acquire pertinent information from the internet and other sources and accurately evaluate the data.
3) interpreting and analyzing data. Using certain techniques, we first select and construct the appropriate model in this section. After that, evaluate the model and report the results.

Four ways are usually used in data mining.

1) Neural network methodology. It draws inspiration from the human brain's neural network, which is trained to operate as a nonlinear prediction model.By mimicking the action of human brain neurons, neural network algorithms may carry out numerous data mining tasks, such as grouping, classification, feature mining, and so forth. Associative memory, non-linear learning, and anti-interference are a few of its many benefits. In complicated circumstances, it also produces precise forecast results. Still, it cannot obtain learning stages in the process and is not suitable for processing high dimensional data. Second, interpreting the results is difficult. Thirdly, making predictions takes a long time[6].

2) The algorithm used by Bayes. This type of classification method is grounded in mathematical statistics and probability. There are two types of Bayesian algorithms. Naive Bayes is the first. Categorization accuracy is higher when each attribute functions independently of the others.If not, it might be lower. The second is the Bayes network with tree augmentation. The ability to recognize the dependencies among the data's intrinsic features and utilize those connections to categorize the information. Large databases can be used with Bayesian. It is easy to use, quick, and accurate.[7].

3) the algorithm for linear regression. The least square function is a statistical technique known as "linear regression" that is used to depict the relationship between one or more independent variables and dependent variables. A linear combination of one or more model parameters makes up this 31 function. When there are multiple independent variables, the condition is referred to as numerous regression. When each attribute in the data is a number and the prediction is a number as well, It is going to be regarded as the finest algorithm[8].

4) Decision Tree Approach. It is a technique for categorizing data according to a collection of guidelines; it resembles the tree structure's flowchart. It is not required to have a protracted building procedure. It is easy to categorize, define, and comprehend. One drawback is that it could be challenging to identify rules based on the fusion of several factors.The approach is especially useful for processing large volumes of data and performs well when handling non-numerical data. A means of illustrating regulations, such as the values that will be earned in what situations, is with a decision tree[9].

In this study, we use the SVM, Decision Tree, and Linear Regression techniques to make the prediction. Next, these three models are compared to decide which prediction model is the best.

A. Web retrieving

1) Selenium
Browser control may be achieved with Python APIs thanks to an application called Selenium.Using Selenium, our team launches a browser, goes to an online map page, and snaps a screenshot [10].

2) WebStorage.
The decision to employ an Object Storage System (Object Storage System) to hold the picture data so that the screenshots can be saved for a later photo and data analysis. When it comes to handling large volumes of data, the Ass system is more scalable than the File Storage System since it is flattened[11].

3) MessageQueue.
We stored the data regarding the examined and unexplored using Message Buffer. photographs

temporarily.Focusing on a tiny area of the map means we can only develop one picture fetching module and one image analyzing module.However, when more road analysis is needed, this strategy is unsatisfactory.Message Queue increases the scalability and resilience of our system.[12]

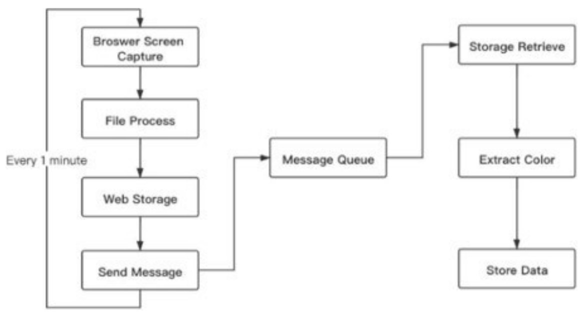


Figure.1 Process of original data retrieving and analysis

B. Machine Learning

1) Support Vector Machine(SVM):

One supervised machine, technique that is well-known for its efficiency in classification and regression applications is the Support Vector Machine (SVM). Situations with high-dimensional feature spaces and complex data distributions are especially well-suited for it. Finding the ideal hyperplane to divide data points into distinct groups or forecast continuous values in regression tasks is the fundamental idea of support vector machines (SVM).

The Support Vector Machine (SVM) algorithm has become a potent tool for predicting traffic congestion because it is accurate and adaptable when modeling intricate traffic patterns. It is a suitable choice for our research because of its capacity to manage noisy inputs, non-linear interactions, and high-dimensional data.

2) Decision Tree:

Widely utilized and adaptable machine learning algorithms, decision trees are renowned for their efficacy and interpretability in both classification and regression problems. They have found use in a number of industries, including banking, healthcare, and more.

It's crucial to remember that Decision Trees do have certain drawbacks, such as their tendency to overfit, which can be lessened by employing ensemble approaches or pruning, among other suitable strategies. They are a useful option in prediction due to their interpretability and simplicity of use.

3) Random Forest:

Over the years, there have been notable developments in machine learning, with the Random Forest algorithm being one of the most notable. Applications for Random Forest's robust and adaptable ensemble learning technique can be found in many different fields. We will explore the foundations of Random Forest, its construction, benefits, and real-world applications in this thorough review.

An ensemble learning method called Random Forest is applied to both regression and classification problems. Each decision tree in the ensemble was trained using a distinct subset of the data. The concept of adding randomization to the building process, which reduces overfitting and enhances model performance, is reflected in the term "Random Forest".

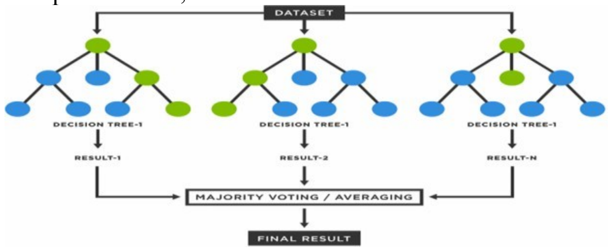


Figure.2 Structure of Random Forest

4) Bayesian Ridge Regression:

A statistical method used in regression analysis and machine learning is the Bayesian Ridge regression algorithm. This particular kind of linear regression was created to address a few issues with more conventional linear regression models.

C. DEEP LEARNING MODELS:

Convolutional neural networks (CNNs) and long short-term memory networks (LSTMs) are two effective deep learning models in the field of traffic congestion prediction. Whereas LSTMs are adept at capturing temporal correlations in traffic time series data, CNNs are particularly good at identifying spatial patterns from visual data, such as traffic camera photos. Through the integration of CNNs' proficiency in extracting spatial features and LSTMs' aptitude in modeling sequences and traffic conditions that change over time, these models provide a comprehensive understanding and forecasting of traffic congestion, thereby enabling more efficient traffic management and congestion mitigation tactics. Let's talk about each algorithm in detail.

1) Convolutional Neural Networks (CNNs):

Convolutional neural networks, or CNNs, are useful for analyzing and forecasting traffic congestion. A particular kind of deep learning model called CNN is mostly intended for handling and analyzing structured grid data, such pictures and movies. CNNs have revolutionized domains such as image classification, object recognition, and image segmentation, and have become an essential tool in computer vision applications.

Although CNNs are typically used to process image data, they may also be modified to handle non-image data by applying a technique known as 1D or 2D convolution, depending on the structure of the data. Examples of this type of data include tabular traffic statistics and sensor data.

2) Long Short-Term Memory (LSTM):

Recurrent neural networks (RNNs) with Long Short-Term Memory (LSTM) architecture are specifically made to handle and model sequential input. When it comes to identifying and comprehending patterns, dependencies, and connections among data sequences, LSTMs are especially good. They are extensively employed in many different applications, such as speech recognition, time series analysis, and natural language processing. LSTMs are different from conventional RNNs in that they can preserve long-term dependencies and address the issue of the vanishing gradient. Specialized memory cells and gating mechanisms enable long-term sequences of selectively storing, updating, and retrieving information, which is how LSTMs accomplish this. Because of this, LSTMs are a good fit for tasks involving time series data where comprehending context and distant relationships is crucial for accurate predictions and modelling.

A Long Short-Term Memory network (LSTM) is a customized deep learning model used to analyze and anticipate traffic conditions over time in the context of traffic congestion prediction. Because they can collect and model the intricate temporal correlations and patterns found in traffic data, LSTMs are a good fit for this purpose.

III. PERFORMANCE EVALUATION

We accomplish this by assessing their performance using the Mean Squared Error (MSE) and Root Mean Squared Error (RMSE), two widely used metrics. These metrics enable us to assess the degree to which our forecasts match the actual facts. When it comes to forecasting traffic congestion, we can objectively determine which prediction techniques are effective and which ones would require improvement by utilizing RMSE and MSE.

Every model must be examined once it has been completed. For activities involving deep learning and machine learning, evaluation measures are crucial.

A. Evaluation Parameters:

Metrics for evaluation are useful in determining how well the prediction model works. Two significant prediction model metrics are covered in this section.

Mean Squared ERROR (MSE):

The statistical model's error level is measured by mean squared error, or MSE. The average squared difference between the observed and anticipated values is evaluated. The MSE is equal to 0 in a model that has no errors. The value of the model increases with the error. Another name for the mean squared error is the mean squared deviation (MSD).

$$MSE = \frac{\sum(y_i - \hat{y})^2}{n}$$

Rooted Mean Squared Error (RMSE):

A key indicator in predictive modeling is the root mean square error (RMSE), whose value represents a model's performance. The average difference between the anticipated and actual values of a statistical model is measured by RMSE. It is the standard 33 deviation of the residuals in mathematics. The distance between the data points and the regression line is represented by residuals.

The RMSE measures the degree to which these residuals have been dispersed, providing insight into how well the observed data adheres to the expected values. The RMSE decreases as the data points approach the regression line because the model has less error. Predictions made by a model with lower error are more accurate.

RMSE values are expressed in the same units as the dependent (outcome) variable and can span from zero to positive infinity. When the predicted and actual numbers are exactly the same, the result is 0. Low RMSE values show that the model has more accurate predictions and matches the data well. Higher levels, on the other hand, indicate greater mistake and less accurate predictions.
The RSME formula for a sample is the following:

$$MSE = \sqrt{\frac{\sum(y_i - \hat{y})^2}{N - P}}$$

IV. COMPARISION AND DISCUSSION

A. COMPARISION

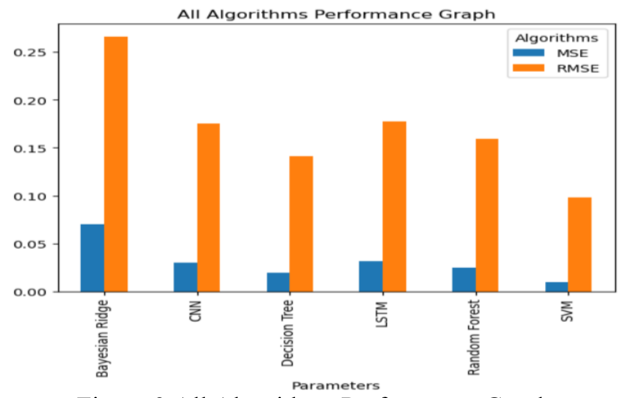


Figure.3 All Algorithms Performance Graph

The Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) values for each method are displayed in the bar chart (Figure.3) that was previously mentioned. The y-axis on this chart shows the associated error numbers, and the x-axis reflects the various algorithms or parameters.

Algorithm Name	**Mean Squared Error (MSE)**
Random Forest	0.00965160562044798
LSTM	0.030584202891979474
CNN	0.018035024129894092
Decision Tree	0.019936270731283737
SVM	0.025308814693551094

Baysian Algorithm	0.07062080416605596

Table1. All Algorithms Mean Squared Error (MSE) Performance

Algorithm Name	**Rooted Mean Squared Error (RMSE)**
Random Forest	0.09824258557493273
LSTM	0.17488339798842964
CNN	0.13429454244269978
Decision Tree	0.14119585946933336
SVM	0.15908744354458365
Baysian Algorithm	0.26574575098401093

Table2. All Algorithms Rooted Mean Squared Error (RMSE) Performance

A thorough comparison of all the models is given in Tables.1 and 2. These tables display the Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) values for each model. A succinct and understandable evaluation of the model's prediction accuracy performance is made possible by the tabular data.

V. CONCLUSION

During our thorough analysis of several traffic prediction algorithms, we found that their efficacy and accuracy varied. With the lowest Mean Squared Error (MSE) of 0.00965 and Root Mean Squared Error (RMSE) of 0.098242, the Random Forest algorithm stood out as the most accurate in predicting traffic patterns. The LSTM (Long Short-Term Memory) model, which showed excellent accuracy with an MSE of 0.03058 and an RMSE of 0.17488, came in close second.

Closely lagging the LSTM, the CNN demonstrated robust performance as well, with an MSE of 0.01803 and an RMSE of 0.13429. With an MSE of 0.01993 and an RMSE of 0.14119, the Decision Tree model demonstrated its performance in properly predicting traffic.

Despite its proficiency, the Bayesian Ridge algorithm exhibited slightly more mistakes, with an RMSE of 0.26574 and an MSE of 0.07062. It nevertheless continued to offer insightful forecasts. Lastly, with an MSE of 0.025308 and an RMSE of 0.15908, the Support Vector Machine (SVM) demonstrated respectable performances.

In conclusion, our analysis showed that the Random Forest algorithm was the best option for accurate traffic predictions, with LSTM and CNN coming in close second. These models outperformed others in terms of accuracy and prediction error minimization.

VI. REFERENCES

[1] Jingyi, Lu. (2023). An efficient and intelligent traffic flow prediction method based on LSTM and variational modal decomposition. Measurement: Sensors, doi: 10.1016/j.measen.2023.100843

[2] Bashir, Mohammed., Nandini, Krishnaswamy., Mariam, Kiran. (2019). Multivariate Time-Series Prediction 10.1109/ANCS.2019.8901870 for Traffic in Large WAN Topology. doi:

[3] Lu, Wang., Rui, Wu., Weichun, Ma., Weiju, Xu. (2023). Examining the volatility of soybean market in the MIDAS framework: The importance of bagging-based weather information. International Review of Financial Analysis, doi: 10.1016/j.irfa.2023.102720

[4] Aditya, Srivastava., Aryan. (2023). A Survey Paper on Traffic Prediction Using Machine Learning. International Journal For Science Technology And Engineering, doi: 10.22214/ijraset.2023.51390

[5] Gobezie, Ayele, and Marta Sintayehu Fufa. "Machine learning and deep learning models for traffic flow prediction: A survey." (2020).

[6] Agafonov A. A., Short-Term Traffic Data Fore casting: A Deep Learning Approach, [J]Optical Memory and Neural Networks Volume 30, Issue1. 2021.PP1-10

[7] Banchhor Chitrakant; Srinivasu N., Analysis of Bayesian optimization algorithms for big data classification based on Map Reduce framework, [J]Journal of Big Data Volume 8, Issue1.2021.
[8] Peng Jianan; Liu Wei; Bretz Frank; Hayter A J, Simultaneous confidence tubes for comparing several multivariate linear regression models., [J]Biometrical Journal 2021.

[9] Camilo Gutierrez-Osorio, Cesar Pedraza.Modern data sources and techniques for analysis and forecast of road accidents: A review[J]. Journal of Traffic and Transportation Engineering (English Edition), 2020, 7(04):432-446.

[10] R. S.Chaulagain, S. Pandey, S.R.Basnet and S.Shakya, "CloudBased Web Scraping for Big Data Applications," 2017 IEEE International Conference on Smart Cloud (SmartCloud), 2017, pp. 138-143, doi: 10.1109Cloud.2017.28.

[11] M. Factor, K. Meth, D.Naor, O. Rodehand1. Satran, "Object storage: the future building block for storage systems,"2005 IEEE International Symposium on Mass Storage Systems and Technology, 2005, pp. 119 123,doi: 10.1109/LGDI.2005.1612479.

[12] Johansson, L. and Dossot, D, "RabbitMQ Essentials: Build distributed and scalable applications with message queuing using RabbitMQ, 2nd Edition, "Packt Publishing, 2020.

DOI: 10.55041/IJSREM31650

ISSN: 2582-3930
Impact Factor: 8.448

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT

An Open Access Scholarly Journal || Index in major Databases & Metadata

CERTIFICATE OF PUBLICATION

International Journal of Scientific Research in Engineering & Management is hereby awarding this certificate to

K. DEEPTHI

in recognization to the publication of paper titled

TRAFFIC FLOW PREDICTION USING MACHINE LEARNING

published in IJSREM Journal on Volume 08 Issue 04 April, 2024

Editor-in-Chief
IJSREM Journal

www.ijsrem.com

e-mail: editor@ijsrem.com