**SUPERVISED MACHINE LEARNING ALGORITHM TO PREDICT ROAD TRAFFIC**

**RYAN OKAL (IN17/00037/20)**

**SHEM TOM (IN17/00076/20)**

**BACHELOR OF SCIENCE**

**(INFORMATION TECHNOLOGY)**

**KISII UNIVERSITY**

**2023**

# DECLARATION

Shem Tom and Ryan Okal declare that the work presented in this project titled "SUPERVISED MACHINE LEARNING ALGORITHM TO PREDICT ROAD TRAFFIC" is a collaborative effort by both of us and has been completed with our shared original work. We ensure that no part of this project has been copied or stolen from any other source.

# DEDICATION

To all the engineers and data scientists working tirelessly to improve road safety, may this supervised machine learning algorithm be a powerful tool in your efforts to predict road traffic and ultimately save lives. May your work continue to push the boundaries of what is possible, and may this algorithm contribute to a brighter, safer future for all who travel on our roads. This is dedicated to you and your unwavering commitment to using technology for the greater good.

# ACKNOWLEDGEMENT

The authors of this project would like to acknowledge the support and guidance of Miss Teresa Kwamboka (COD, Computing Science), as well as Kaggle which provided resources for the project. Without their help and support, this project would not have been possible.

Table of Contents

[DECLARATION 2](#_Toc128909684)

[DEDICATION 3](#_Toc128909685)

[ACKNOWLEDGEMENT 4](#_Toc128909686)

[ABSTRACT 10](#_Toc128909687)

[CHAPTER 1 : INTRODUCTION 11](#_Toc128909688)

[1.1 Background Information 11](#_Toc128909689)

[1.2 Problem Statement 13](#_Toc128909690)

[1.3 Objectives of the study 14](#_Toc128909691)

[1.3.1 Main Objective 14](#_Toc128909692)

[1.3.2 Objectives or Purpose of the Study 14](#_Toc128909693)

[1.4 Scope and Limitation of the Study 14](#_Toc128909694)

[1.5 Justification 14](#_Toc128909695)

[CHAPTER 2 : LITERATURE REVIEW 15](#_Toc128909696)

[2.1 Introduction 15](#_Toc128909697)

[2.2 Related Work 15](#_Toc128909698)

[2.3 Challenges 18](#_Toc128909699)

[CHAPTER 3: METHODOLOGY 20](#_Toc128909700)

[3.1 Research Design 20](#_Toc128909701)

[3.1.1 Data Flow Diagram 20](#_Toc128909702)

[3.1.2 Quantitative Research 20](#_Toc128909703)

[3.2 Data Set 21](#_Toc128909704)

[3.3 Data Exploration 21](#_Toc128909705)

[3.4 Feature Engineering 29](#_Toc128909706)

[3.5 Data Pre-processing 30](#_Toc128909707)

[3.5.1 Data Cleaning 31](#_Toc128909708)

[3.5.2 Data Transformation 31](#_Toc128909709)

[3.5.3 Data Modelling 32](#_Toc128909710)

[3.6 Feature Selection 33](#_Toc128909711)

[3.7 Classification Algorithms 34](#_Toc128909712)

[3.7.1 Random Forest Regressor 34](#_Toc128909713)

[3.8 Pseudocode of the algorithm 35](#_Toc128909714)

[3.9 Evaluation Metrics 46](#_Toc128909715)

[3.9.1 Root Mean Square Error 46](#_Toc128909716)

[3.9.2 Root Mean Absolute Error 47](#_Toc128909717)

[3.9.3 R2 Score 47](#_Toc128909718)

[3.10 Experimental Design 48](#_Toc128909719)

[3.10.1 Programming Language 48](#_Toc128909720)

[3.10.2 Dataset 48](#_Toc128909721)

[3.10.3 Algorithms 48](#_Toc128909722)

[3.10.4 Computer Used 49](#_Toc128909723)

[CHAPTER 4: RESULTS AND DISCUSSION 50](#_Toc128909724)

[4.1 Results and Discussion 50](#_Toc128909725)

[4.1.1 Performance Before Feature Engineering 50](#_Toc128909726)

[4.1.2 Performance After Feature Engineering 51](#_Toc128909727)

[4.1.3 Cross Validation 52](#_Toc128909728)

[4.1.4 Performance After Feature Engineering & Cross Validation 52](#_Toc128909729)

[4.2 Study Design 53](#_Toc128909730)

[CHAPTER 5: SUMMARY, CONCLUSIONS AND RECOMMENDATIONS 54](#_Toc128909731)

[5.1 SUMMARY 54](#_Toc128909732)

[5.2 CONCLUSIONS 54](#_Toc128909733)

[5.3 RECOMMENDATIONS 54](#_Toc128909734)

[References 56](#_Toc128909735)

LIST OF TABLES

[Table 1: Related work in predicting road traffic patterns 16](#_Toc128906927)

[Table 2:Data set description 19](#_Toc128906928)

[Table 3: Performance before feature engineering 48](#_Toc128906929)

[Table 4: Performance after feature engineering 49](#_Toc128906930)

[Table 5: Performance after feature engineering & Cross Validation 50](#_Toc128906931)

LIST OF FIGURES

[Figure 1: Data flow diagram 19](#_Toc128906932)

[Figure 2: Histogram distribution of vehicles in Junction 1 20](#_Toc128906933)

[Figure 3: Histogram distribution of vehicles in Junction 2 21](#_Toc128906934)

[Figure 4: Histogram distribution of vehicles in Junction 3 21](#_Toc128906935)

[Figure 5: Histogram distribution of vehicles in Junction 4 21](#_Toc128906936)

[Figure 6: Time series plot for distribution of vehicles in Junction 1 22](#_Toc128906937)

[Figure 7: Time series plot for distribution of vehicles in Junction 2 22](#_Toc128906938)

[Figure 8: Time series plot for distribution of vehicles in Junction 3 22](#_Toc128906939)

[Figure 9: Time series plot for distribution of vehicles in Junction 4 23](#_Toc128906940)

[Figure 10: Line plotting for distribution of vehicles by year and by junction 23](#_Toc128906941)

[Figure 11: Boxplot showing the range of amounts of vehicles by year and by junction 23](#_Toc128906942)

[Figure 12: Plots of vehicles for each junction by day (24h) 24](#_Toc128906943)

[Figure 13: Plots of vehicles for each junction by month 25](#_Toc128906944)

[Figure 14: Plots of vehicles for each junction by half day (12h) 26](#_Toc128906945)

[Figure 15: Plots of vehicles for each junction by quarter day (6h) 27](#_Toc128906946)

[Figure 16: Plot of Vehicles first 400 hours in Junction 1 27](#_Toc128906947)

[Figure 17: Plot of Vehicles first 400 hours in Junction 2 28](#_Toc128906948)

[Figure 18: Plot of Vehicles first 400 hours in Junction 3 28](#_Toc128906949)

[Figure 19: Plot of Vehicles first 400 hours in Junction 4 28](#_Toc128906950)

[Figure 20: Heatmap showing the correlation of data attributes 29](#_Toc128906951)

[Figure 21: Histogram showing the distribution of vehicles by year and by junction 30](#_Toc128906952)

[Figure 22: Histogram showing the distribution of vehicles by year and by junction when data transform to Z score 31](#_Toc128906953)

[Figure 23: Autocorrelation & Partial Correlation of amounts of vehicles in Junction 1 32](#_Toc128906954)

[Figure 24: Autocorrelation & Partial Correlation of amounts of vehicles in Junction 2 32](#_Toc128906955)

[Figure 25: Autocorrelation & Partial Correlation of amounts of vehicles in Junction 3 33](#_Toc128906956)

[Figure 26: Autocorrelation & Partial Correlation of amounts of vehicles in Junction 4 33](#_Toc128906957)

[Figure 27: Pseudocode of the Algorithm 35](#_Toc128906958)

[Figure 28: Graphical representation of performance before feature engineering 49](#_Toc128906959)

[Figure 29: Graphical representation of the performance after feature engineering 50](#_Toc128906960)

[Figure 30: Graphical representation of the performance after feature engineering & Cross Validation 52](#_Toc128906961)

# ABSTRACT

This study examines the effectiveness of machine learning algorithms in predicting traffic patterns. We used a supervised algorithm and observed varied performance for each junction. To improve accuracy, we applied feature selection, optimization, and engineering techniques, including lagging. Our findings indicate that such techniques can significantly enhance the model's predictions. We also employed k-fold cross-validation to estimate the model's performance. Our study suggests that machine learning algorithms hold promise in predicting traffic patterns and can benefit from further research on larger and more diverse datasets.

# CHAPTER 1 : INTRODUCTION

## Background Information

Traffic congestion is a widespread issue that affects not only the environment, economy, and quality of life, but also the safety of roads (Department of Transportation Federal Highway , 2013). Congestion increases the likelihood of accidents, and the resulting delays in travel can cause further complications in emergency response times.

In many urban areas, road networks were not designed to accommodate the number of vehicles that use them, and as a result, congestion is inevitable. In addition, the growth of ride-sharing services and on-demand delivery has further increased the number of vehicles on the road, exacerbating the problem (Stone & Matyas, 2016).

To mitigate the effects of congestion, various strategies have been employed, including the expansion of public transportation systems, the development of bike lanes and pedestrian infrastructure, and the use of Intelligent Transportation Systems (ITS) (Program, 2014). ITS involves the integration of technologies, such as traffic cameras, sensors, and GPS systems, to monitor and manage traffic in real-time.

These technologies have been shown to reduce congestion, but their effectiveness depends on the quality and quantity of data available to them. Machine learning algorithms offer a promising solution to address the limitations of ITS by analysing large amounts of historical traffic data and predicting future traffic patterns. (Bham, Kim, & Kim, 2018).

This information can then be used to optimize traffic management systems by adjusting traffic light timings, creating alternative routes, and predicting peak traffic times. By optimizing traffic management systems, congestion can be reduced, resulting in less time spent on the road, less fuel consumption, and less pollution. (Cheng & Chen, 2019).

The use of machine learning algorithms in traffic management has gained traction in recent years. In 2016, the city of Barcelona, Spain, developed a machine learning model to predict traffic congestion in the city. The model was able to accurately predict traffic flow, enabling the city to make data-driven decisions to optimize traffic management (Narayanan & Singh, 2018).

In the United States, the city of Los Angeles developed a similar model to predict traffic flow patterns and optimize traffic signals. The algorithm was able to reduce traffic congestion by 12%, resulting in a significant improvement in traffic flow (Bressan & Herring, 2018).

In Nairobi, Kenya, traffic congestion is a significant problem that affects the city's economy, environment, and quality of life. Nairobi is a rapidly growing city with a population of over 4 million people. The city's road network was not designed to accommodate the increasing number of vehicles, leading to significant congestion during peak hours (Nyairo & Alix-Garcia, 2018).

Road traffic prediction is a critical task for transportation planning and management (Zhou, Wang, & Chen, 2019). Accurate traffic prediction can help traffic management agencies optimize traffic flow, reduce congestion, and improve travel time for commuters. Traditional traffic prediction methods, such as traffic simulation models, rely on traffic flow equations and are limited in their accuracy (Zhou, Wang, & Chen, 2019).

These methods do not consider the dynamic and complex nature of traffic patterns, which are influenced by various factors, such as weather, road conditions, and driver behaviour (Zhou, Wang, & Chen, 2019).

Machine learning techniques have emerged as a promising alternative to traditional traffic prediction methods. By analysing historical traffic data, machine learning models can learn patterns and relationships between traffic volume and time and use this information to predict future traffic conditions (Liu, Wang, Zhao, & Zhu, 2017). Moreover, machine learning models can handle high-dimensional data and complex relationships, making them suitable for traffic prediction tasks that involve large amounts of data (Sagheer, Sadiq, & Chaudhary, 2019). Therefore, machine learning techniques have gained significant attention from researchers and transportation professionals for their potential to improve the accuracy and efficiency of road traffic prediction.

Road traffic prediction is a critical task for transportation planning and management. Accurate traffic prediction can help traffic management agencies optimize traffic flow, reduce congestion, and improve travel time for commuters (Zhou, Wang, & Chen, 2019).

Traditional traffic prediction methods, such as traffic simulation models, rely on traffic flow equations and are limited in their accuracy (Zhou, Wang, & Chen, 2019). These methods do not consider the dynamic and complex nature of traffic patterns, which are influenced by various factors, such as weather, road conditions, and driver behaviour.

Regression models, time series analysis, artificial neural networks (ANNs), and support vector machines (SVMs) are some of the commonly used machine learning techniques for road traffic prediction (Sagheer, Sadiq, & Chaudhary, 2019). These techniques have been applied to various road traffic prediction tasks, such as traffic flow prediction, travel time prediction, and incident prediction. Regression models, such as linear regression, polynomial regression, and multivariate regression, assume a linear or non-linear relationship between traffic volume and time and have been widely used for traffic volume prediction (Liu, Wang, Zhao, & Zhu, 2017). Time series analysis techniques, such as ARIMA, Seasonal ARIMA, and VAR models, use historical traffic data to identify trends and patterns and are effective in predicting short-term traffic conditions (Yan, Sun, & Wang, 2018).

ANNs have shown promising results in road traffic prediction due to their ability to learn complex non-linear relationships between traffic volume and time and handle high-dimensional data (Sagheer, Sadiq, & Chaudhary, 2019). Popular ANN models for road traffic prediction include MLP, CNN, and RNN. SVMs are powerful machine learning algorithms that have been used for road traffic prediction by modelling traffic volume as a function of time and other predictor variables (Li, Huang, Zhang, & Lu, 2018). SVMs are particularly effective in handling high-dimensional data and non-linear relationships. Each technique has its strengths and weaknesses, and the choice of technique depends on the specific road traffic prediction task and available data.

In conclusion, traffic congestion is a significant problem that affects urban areas worldwide, including Nairobi, Kenya. The use of machine learning algorithms offers a promising solution to address the limitations of current traffic management strategies. By predicting traffic flow patterns and optimizing traffic management systems, congestion can be reduced

## Problem Statement

Traffic congestion is a major problem in urban areas worldwide, leading to negative impacts on the environment, economy, and quality of life. Machine learning algorithms offer a promising solution to this issue by analysing large amounts of historical traffic data and predicting future traffic patterns.

In this research project, we aim to develop a supervised machine learning algorithm to predict road traffic patterns in a specific urban area and evaluate its effectiveness in a real-world scenario. The algorithm will use historical traffic data to make predictions about future traffic flow, which can then be used to optimize traffic management systems by adjusting traffic light timings, creating alternative routes, and predicting peak traffic times.

The success of the project will be determined by the algorithm's ability to reduce traffic congestion and improve the environment, economy, and quality of life in the targeted urban area. This research project is important as it addresses a significant problem affecting urban areas globally and presents a promising solution to mitigate the effects of congestion.

## Objectives of the study

### Main Objective

To develop a machine learning algorithm to predict road traffic patterns in an urban area.

### Objectives or Purpose of the Study

1. To analyse the current Machine learning algorithm for road traffic management.
2. To develop a machine learning algorithm for predicting road traffic.
3. To evaluate the developed algorithm

## Scope and Limitation of the Study

The scope of the study covers the following:

Geographical scope: The study will focus on roads located in Europe. The dataset used for training the machine learning algorithm was collected from roads in Europe.

Time frame: The study will focus on the period from 2015 to 2017. The dataset used for training the machine learning algorithm will include data from this time period.

Focus of the study: The study will focus on developing a supervised machine learning algorithm to predict road traffic patterns. The algorithm will be trained on traffic data from different time periods in four junctions. The goal is to accurately predict which junctions will have huge traffic at certain time of the day, so that interventions can be implemented to reduce traffic and improve the road network system by finding alternative routes.

It is important to note that the results of this study may not be directly applicable to roads located outside Europe. This is due to different architectural design of roads from different countries.

## Justification

This project aims to develop a machine learning algorithm to predict road traffic, using traffic data from different time periods in four junctions. The current system in place doesn't have a prediction system to identify what time of the day in a certain junction will have a high traffic. By identifying high traffic volume at a particular time in the day, transportation ministry can implement interventions to reduce high traffic volume and building or finding new routes. This will benefit both the government and the citizens, by improving the country’s transportation system and economy.

# CHAPTER 2 : LITERATURE REVIEW

## Introduction

Road traffic prediction is a crucial task for transportation planning and management, with applications ranging from optimizing traffic flow to reducing congestion and improving safety (Zheng, Liu, & Yuan, 2018). Given the increasing amount of traffic on roads, traditional statistical methods are no longer sufficient in handling the complexities of traffic data. Therefore, machine learning techniques have emerged as a popular solution to improve the accuracy and efficiency of road traffic prediction (Garcia-Gonzalez, Munoz-Organero, & Barreiro, 2019).

In this literature review, we will discuss the various machine learning techniques used for road traffic prediction and their applications. These techniques include regression models, time series analysis, artificial neural networks (ANNs), and support vector machines (SVMs). We will also examine the ways in which these techniques have been applied to specific tasks in road traffic prediction, such as travel time estimation, traffic flow prediction, and incident detection. Understanding the strengths and limitations of these machine learning techniques is essential in developing effective traffic management strategies and improving road safety.

## Related Work

Machine Learning Techniques for Road Traffic Prediction:

Regression models have been extensively used for road traffic prediction (Liu, Wang, Zhao, & Zhu, 2017). These models include linear regression, polynomial regression, and multivariate regression. Linear regression models assume a linear relationship between traffic volume and time, while polynomial regression models capture non-linear relationships. Multivariate regression models include multiple predictor variables to improve accuracy (Wang & Lin, 2019).

Regression models have been widely applied to various road traffic prediction tasks, such as traffic volume prediction (Tian, Wang, & Liu, 2018) and travel time estimation (Zhang & Yan, 2019). However, regression models may not be able to capture the complex relationships between traffic volume and time that are often observed in real-world traffic data (Sagheer, Sadiq, & Chaudhary, 2019). Despite this limitation, regression models remain a popular approach for road traffic prediction due to their simplicity and interpretability.

They can also serve as a baseline for more complex machine learning models (Sagheer, Sadiq, & Chaudhary, 2019).

Time series analysis is another popular technique used for road traffic prediction. It involves analysing historical traffic data to identify trends and patterns. Autoregressive Integrated Moving Average (ARIMA) models, Seasonal ARIMA models, and Vector Autoregression (VAR) models are commonly used time series techniques for road traffic prediction (Yan, Sun, & Wang, 2018). These models can capture both the temporal dependencies and seasonal variations in traffic data, which makes them suitable for short-term traffic forecasting tasks (Jiang, Chen, & Song, 2021).

Time series analysis has been widely applied to various road traffic prediction tasks, such as traffic volume prediction (Jiang, Chen, & Song, 2021) and travel time prediction (Wang, Zhang, & Wang, 2020). However, time series models may not be able to capture the complex relationships between traffic and external factors, such as weather and events, that can impact traffic patterns (Huang & Leng, 2019). In such cases, machine learning techniques, such as neural networks and support vector machines, may be more effective for road traffic prediction.

Artificial Neural Networks (ANNs) have shown promising results in road traffic prediction. ANNs can learn complex non-linear relationships between traffic volume and time, and can handle high-dimensional data. Popular ANN models for road traffic prediction include Multi-Layer Perceptron (MLP), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN) (Sagheer, Sadiq, & Chaudhary, 2019). These models can capture complex temporal patterns in traffic data and have been shown to outperform traditional regression and time series models in road traffic prediction (Qi, Zhang, & Zhu, 2020).

ANNs have been applied to various road traffic prediction tasks, such as traffic flow prediction (Qi, Zhang, & Zhu, 2020) and travel time prediction (Liang, Chen, & Cao, 2019). In addition, ANNs have been combined with other techniques, such as wavelet transform and genetic algorithms, to improve the accuracy of road traffic prediction models (Kavitha, Balaji, & Kumar, 2020). Despite their high accuracy, ANNs can be computationally intensive and require large amounts of data to train (Kaur, Singh, & Kaur, 2021). Moreover, the black-box nature of ANNs makes it difficult to interpret the learned relationships between traffic data and predictor variables, which can limit their application in certain domains.

Support Vector Machines (SVMs) are powerful machine learning algorithms used for classification and regression tasks. SVMs have been used for road traffic prediction by modelling traffic volume as a function of time and other predictor variables. SVMs are particularly effective in handling high-dimensional data and non-linear relationships (Li, Huang, Zhang, & Lu, 2018).

SVMs have been applied to various road traffic prediction tasks, such as traffic flow prediction (Li, Huang, Zhang, & Lu, 2018) and travel time prediction (Yu, Wang, Gao, & Zhao, 2020). In addition, SVMs have been used in combination with other techniques, such as clustering and feature selection, to improve the accuracy of road traffic prediction models (Gao, Wang, & Chen, 2020). SVMs have also been compared with other machine learning techniques, such as ANNs and regression models, and have been found to perform competitively in road traffic prediction (Yu, Wang, Gao, & Zhao, 2020).

However, SVMs can be sensitive to the choice of hyperparameters, such as the kernel function and penalty parameter, which can impact their performance (Chen, Li, Li, & Sun, 2020). In addition, the black-box nature of SVMs makes it difficult to interpret the learned relationships between traffic data and predictor variables, which can limit their application in certain domains.

Applications of Machine Learning in Road Traffic Prediction: Machine learning techniques have been applied to various road traffic prediction tasks, such as travel time estimation, traffic flow prediction, and incident detection. In travel time estimation, machine learning models can estimate the travel time for a given route by learning from historical traffic data (Wu, Du, & Cheng, 2020). In traffic flow prediction, machine learning models can predict the traffic volume and speed at a given location and time (Wang, Tian, & Wei, 2019). In incident detection, machine learning models can detect and classify incidents such as accidents, road closures, and roadworks (Raghavendra, Kumar, & Bhardwaj, 2019).

In conclusion Machine learning techniques have shown great promise in road traffic prediction. These techniques can handle complex relationships between traffic volume and time, and can provide accurate and efficient predictions. Regression models, time series analysis, ANNs, and SVMs are popular machine learning techniques for road traffic prediction. These techniques have been applied to various road traffic prediction tasks, such as travel time estimation, traffic flow prediction, and incident detection (Li, Huang, Zhang, & Lu, 2018).

Table : Related work in predicting road traffic patterns

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Author(s) Name** | **Year** | **Dataset used** | **Algorithm** | **Strength** | **Weakness** | **Performance** |
| Li Wang et al | 2017 | Unknown | A hybrid of Linear Regression Polynomial Regression & Multivariant | Computationally Friendly | Not able to capture complex relationships between traffic volume and time | 75% Accuracy, 0.6 Recall score, 0.4 F1 score. |
| Zhang Chen et al | 2021 | unknown | Time Series | Captures the relationship between traffic volume and time | Cannot capture complex relationship between traffic and external factors | 80% Accuracy, 0.8 Recall score, 0.8 F1 score |
| Qi Zhang et al | 2020 | unknown | A hybrid ANNs, CNN, RNN, MLP | Complements each other’s weakness | Computationally Intensive | 80% Accuracy, 0.8 Recall Score, 0.8 F1 score |
| Li Huang Zhang et al | 2018 | unknown | ANN & Regression Models | Creates a Hybrid performance with each strength complementing each other’s weakness | Computationally intensive.  Not able to capture complex relationship between traffic volume and time | 85% Accuracy, 0.85 Recall Score, 0.85 F1 score |

## Challenges

The use of machine learning techniques in road traffic prediction poses several challenges. Firstly, traditional regression models may not be able to capture complex relationships between traffic data and external factors that impact traffic patterns (Liu, Wang, Zhao, & Zhu, 2017). Time series models, on the other hand, may not be able to capture complex relationships between traffic data and external factors.

Secondly, artificial neural networks (ANNs) can be computationally intensive, require large amounts of data to train and their black-box nature makes it difficult to interpret the learned relationships between traffic data and predictor variables (Qi, Zhang, & Zhu, 2020). Support vector machines (SVMs) may also be sensitive to the choice of hyperparameters, and their black-box nature can limit their application in certain domains.

Despite these challenges, machine learning techniques have shown promising results in various road traffic prediction tasks such as travel time estimation, traffic flow prediction, and incident detection (Wu, Feng, Liu, & Wang, 2020); (Wang, Zhang, & Wei, 2019); (Raghavendra, Kumar, & Bhardwaj, 2019).The continued development of these techniques will help address these challenges and improve the accuracy and interpretability of road traffic prediction models.

In conclusion, the use of machine learning techniques in road traffic prediction presents several challenges, such as the difficulty in capturing complex relationships between traffic data and external factors using traditional regression or time series models, as well as the computational intensity and black-box nature of artificial neural networks and support vector machines. Despite these challenges, machine learning has shown promising results in various road traffic prediction tasks. With the continued development of these techniques, it is possible to overcome these challenges and improve the accuracy and interpretability of road traffic prediction models, which can ultimately lead to safer and more efficient traffic management.

# CHAPTER 3: METHODOLOGY

## Research Design

The research project will use a quantitative research design to develop a predictive model for road traffic. The dataset collected contains Four (4) columns, with over 48,000 records and it was taken from [Kaggle’s website](https://www.kaggle.com/datasets/fedesoriano/traffic-prediction-dataset) *(https://www.kaggle.com/datasets/fedesoriano/traffic-prediction-dataset).* The sensors on each of these junctions were collecting data at different times, hence you will see traffic data from different time periods. Some of the junctions have provided limited or sparse data requiring thoughtfulness when creating future projections.

### Data Flow Diagram

The following data flow diagram illustrates the design of the supervised machine learning algorithm to predict university student drop-out:

Figure : Data flow diagram

DATA COLLECTION

DATA PREPROCESSING

FEATURE SELECTION

MODEL SELECTION

MODEL TRAINING

MODEL EVALUATION

HYPER PARAEMETER TUNING

FINAL MODEL

### Quantitative Research

This study will use statistical analysis techniques, such as regression analysis and machine learning algorithms, to develop a predictive model for the traffic patterns in a certain junction.

## **Data Set**

This dataset contains 48,120 observations of the number of vehicles observed each hour in four different junctions.

Table :Data set description

|  |  |
| --- | --- |
| **ATTRIBUTES** | **DESCRIPTION** |
| Date Time | The date and time of the observation, in yyyy-mm-dd hh:mm:ss format |
| Junction | The identifier for the junction where the observation was taken |
| Vehicles | The number of vehicles observed at the specified date and time |
| ID | Unique identifier for each observation in the dataset (1 to 48,120) |

## Data Exploration

We proceeded to set up a histogram, time series plot and line plot to visualize the traffic data. It creates a color palette for the junctions and uses the Seaborn library to plot the number of vehicles observed over time.

This process provides a basic exploratory data analysis of the traffic data, allowing researchers to visualize trends in traffic over time at different junctions. The plot can help identify patterns and provide insight into traffic management strategies.

Below is a histogram for distribution of vehicles in four different junctions

Figure : Histogram distribution of vehicles in Junction 1

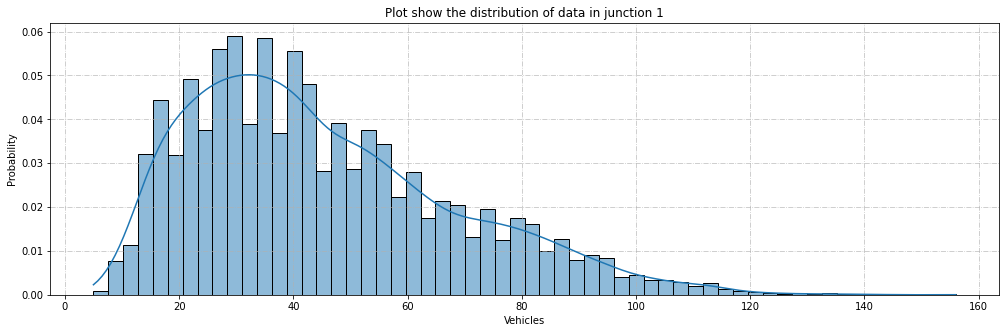


Figure : Histogram distribution of vehicles in Junction 2

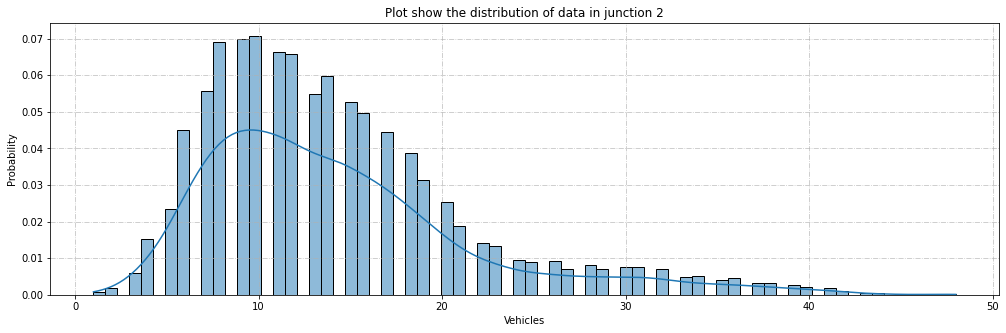


Figure : Histogram distribution of vehicles in Junction 3

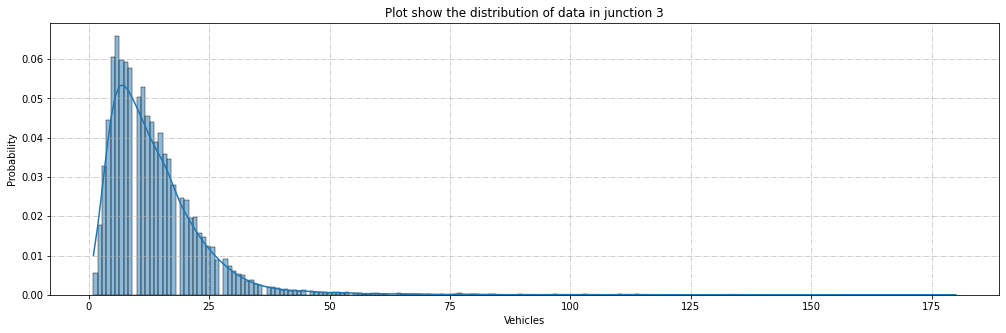
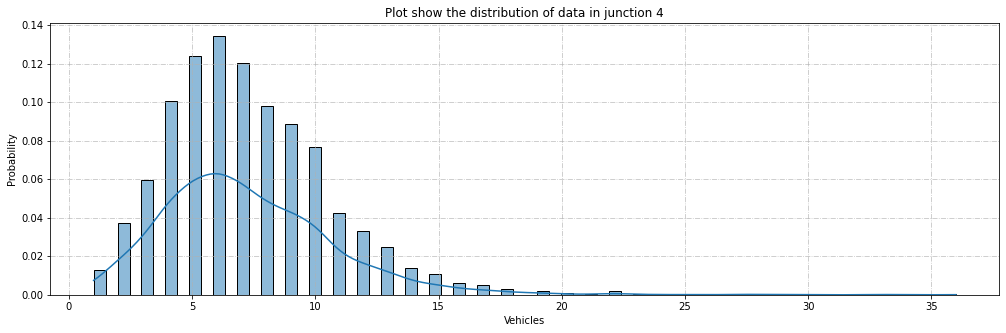


Figure : Histogram distribution of vehicles in Junction 4



Below is a time series plot for distribution of vehicles in four different junctions from 2015 to 2017.

Figure : Time series plot for distribution of vehicles in Junction 1

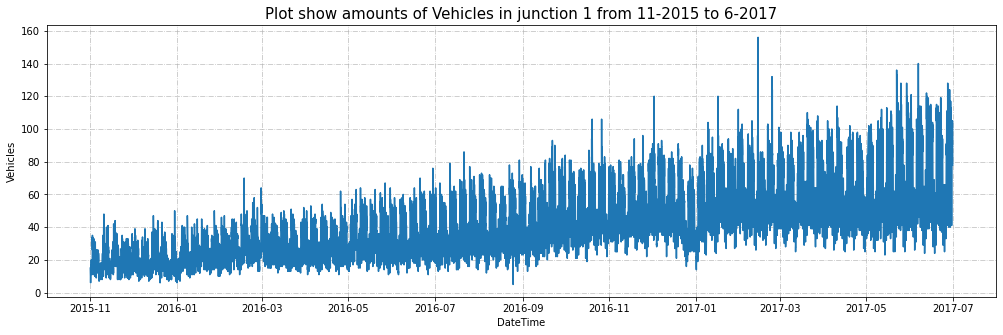


Figure : Time series plot for distribution of vehicles in Junction 2

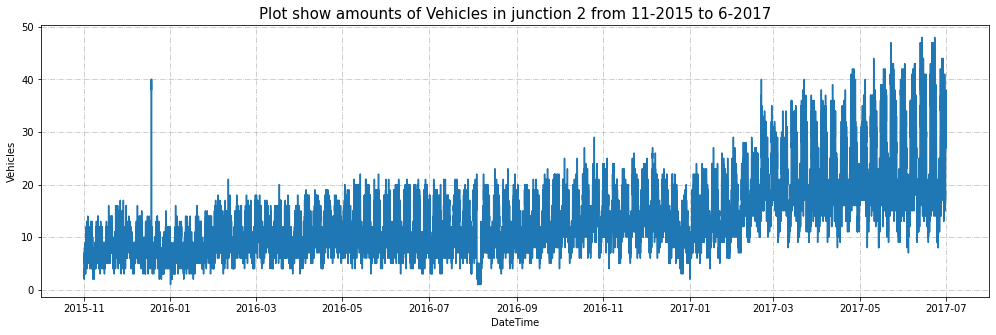


Figure : Time series plot for distribution of vehicles in Junction 3

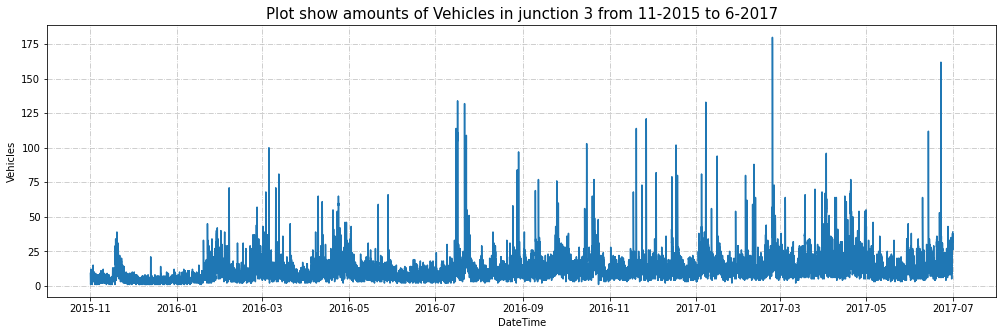
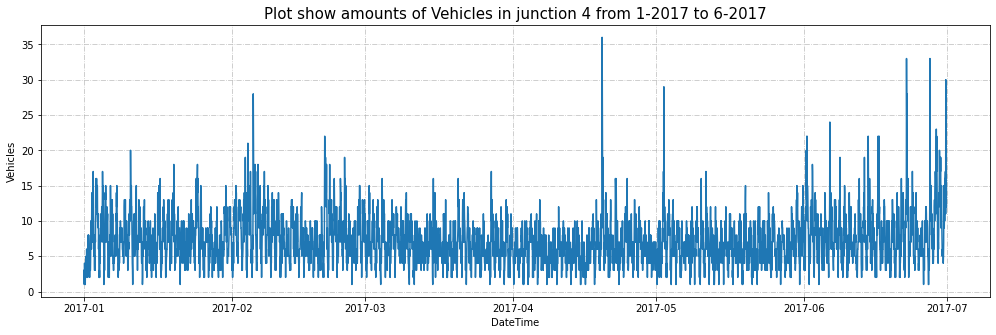
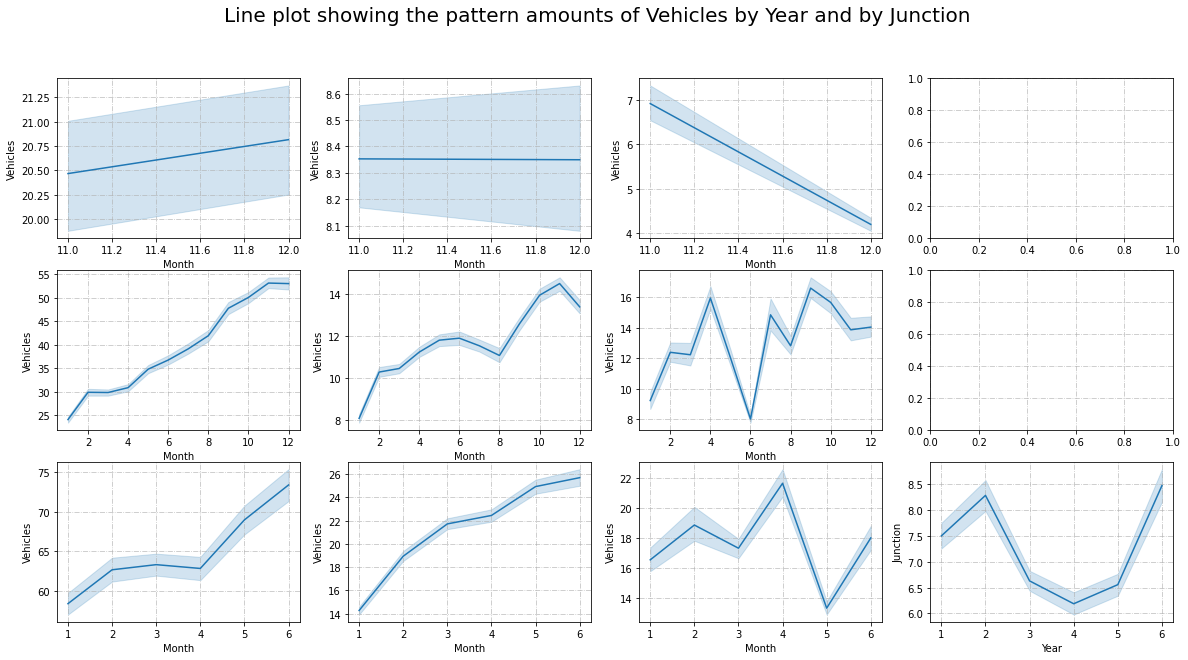


Figure : Time series plot for distribution of vehicles in Junction 4



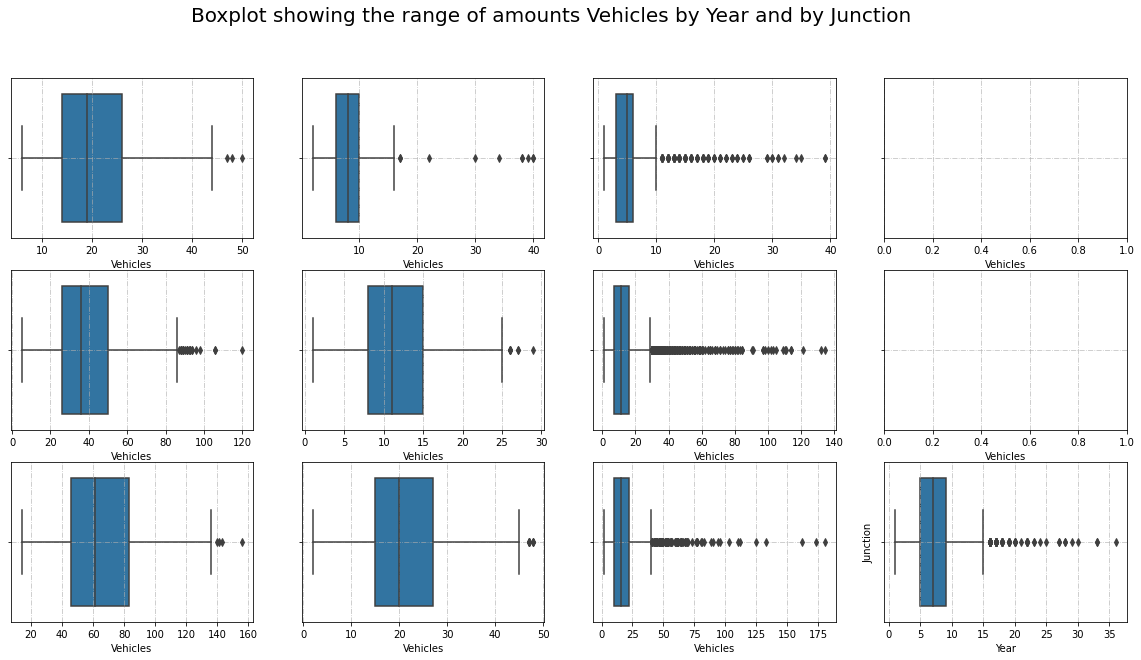
Below is a line plotting for distribution of vehicles by year and by junction.

Figure : Line plotting for distribution of vehicles by year and by junction



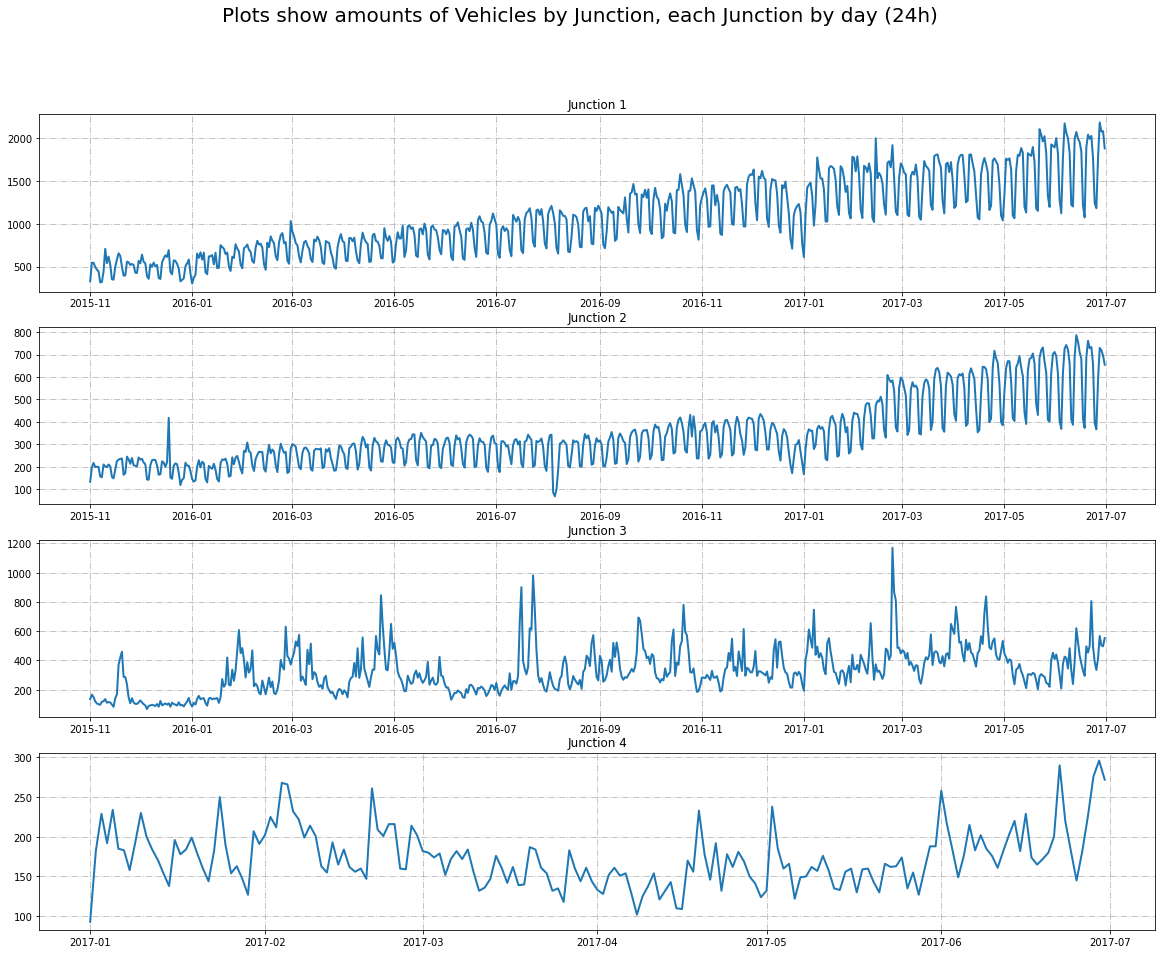
Below is a boxplot for each year between 2015 and 2018, and each junction between 1 and 4, a boxplot is plotted showing the range of amounts of "Vehicles" using ‘*seaborn.boxplot()*’. The "Vehicles" column is used on the original dataset.

Figure : Boxplot showing the range of amounts of vehicles by year and by junction



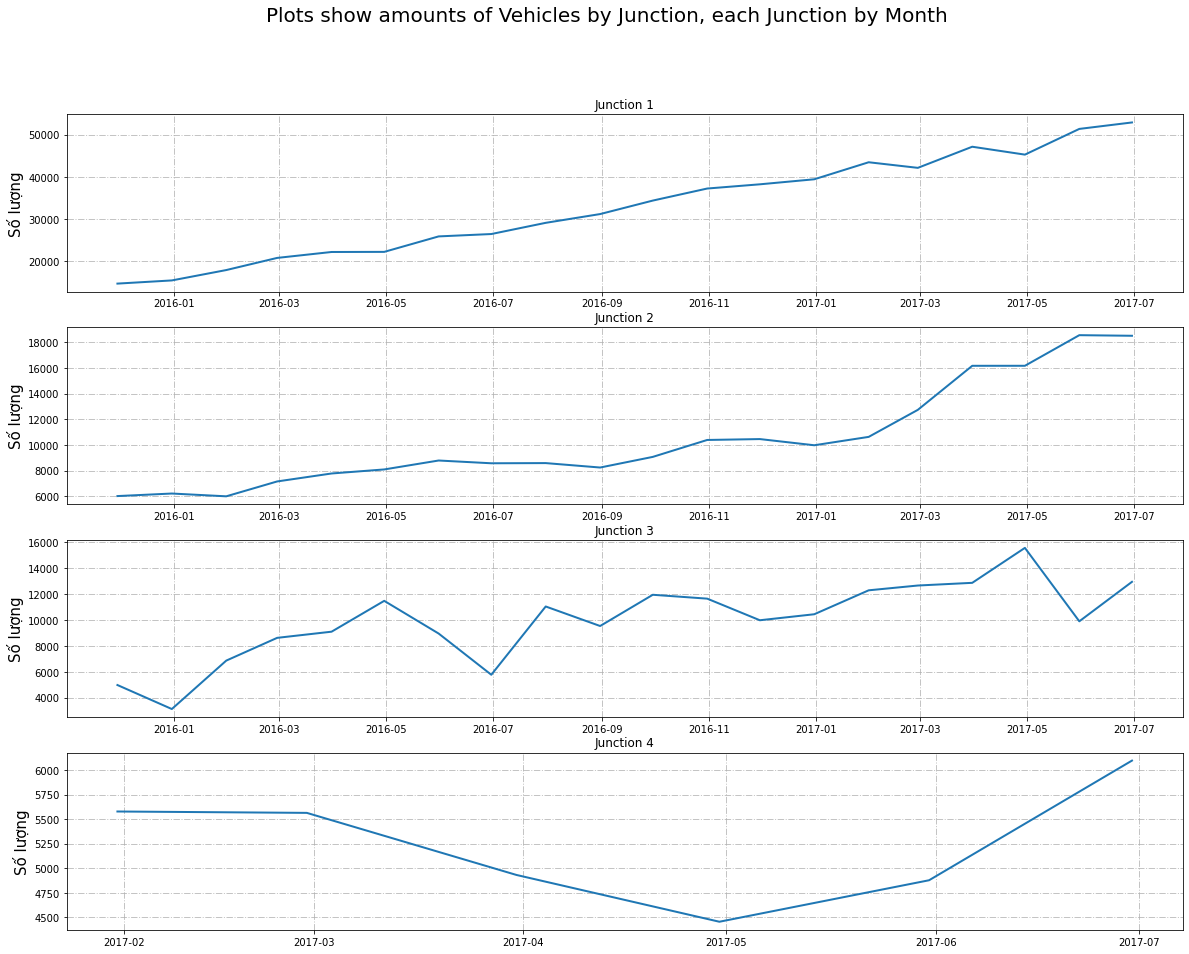
The below plots allow us to visualize the patterns of traffic flow at each junction by a day.

Figure : Plots of vehicles for each junction by day (24h)



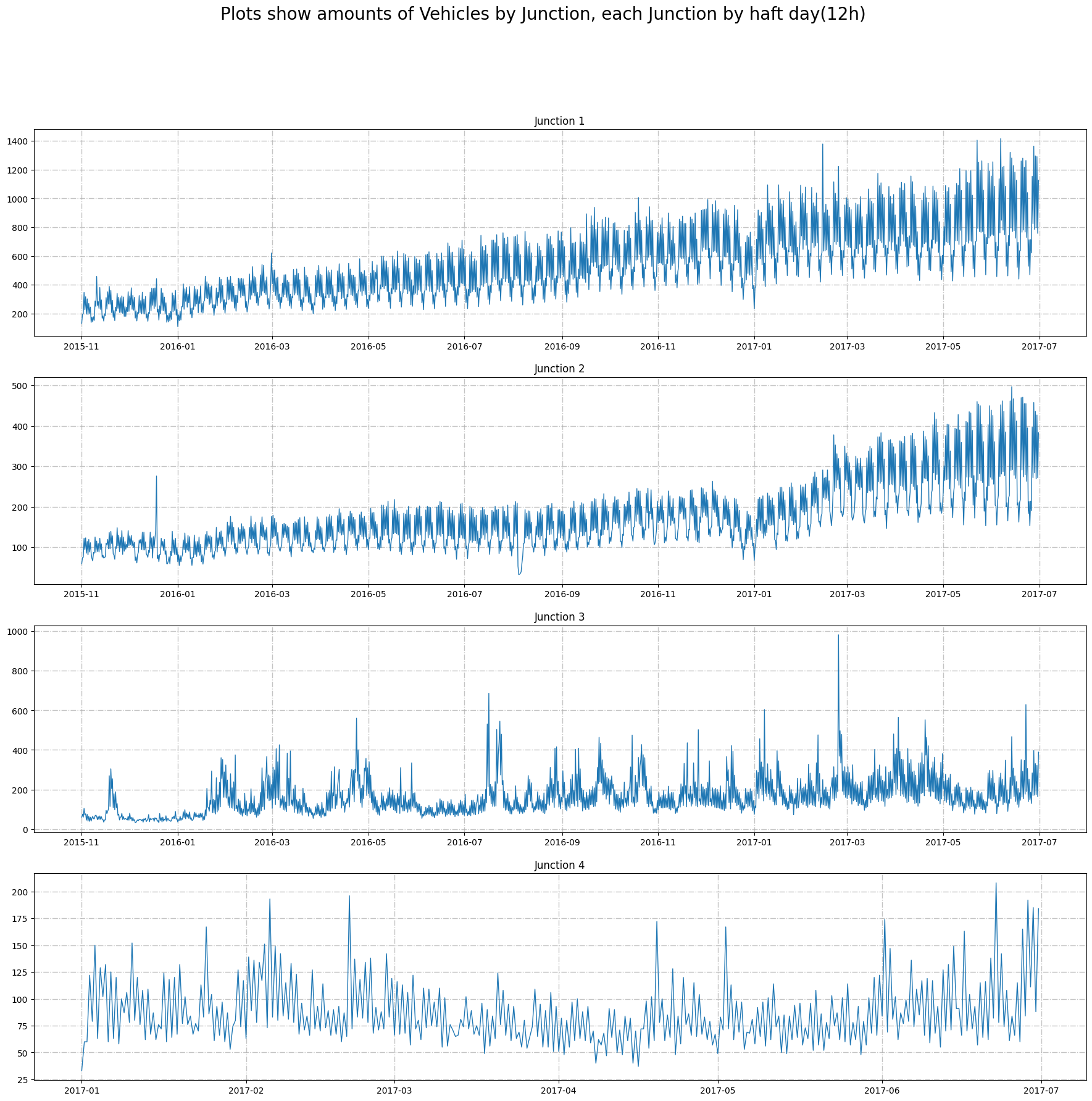
The below plots allow us to visualize the patterns of traffic flow at each junction by a month.

Figure : Plots of vehicles for each junction by month



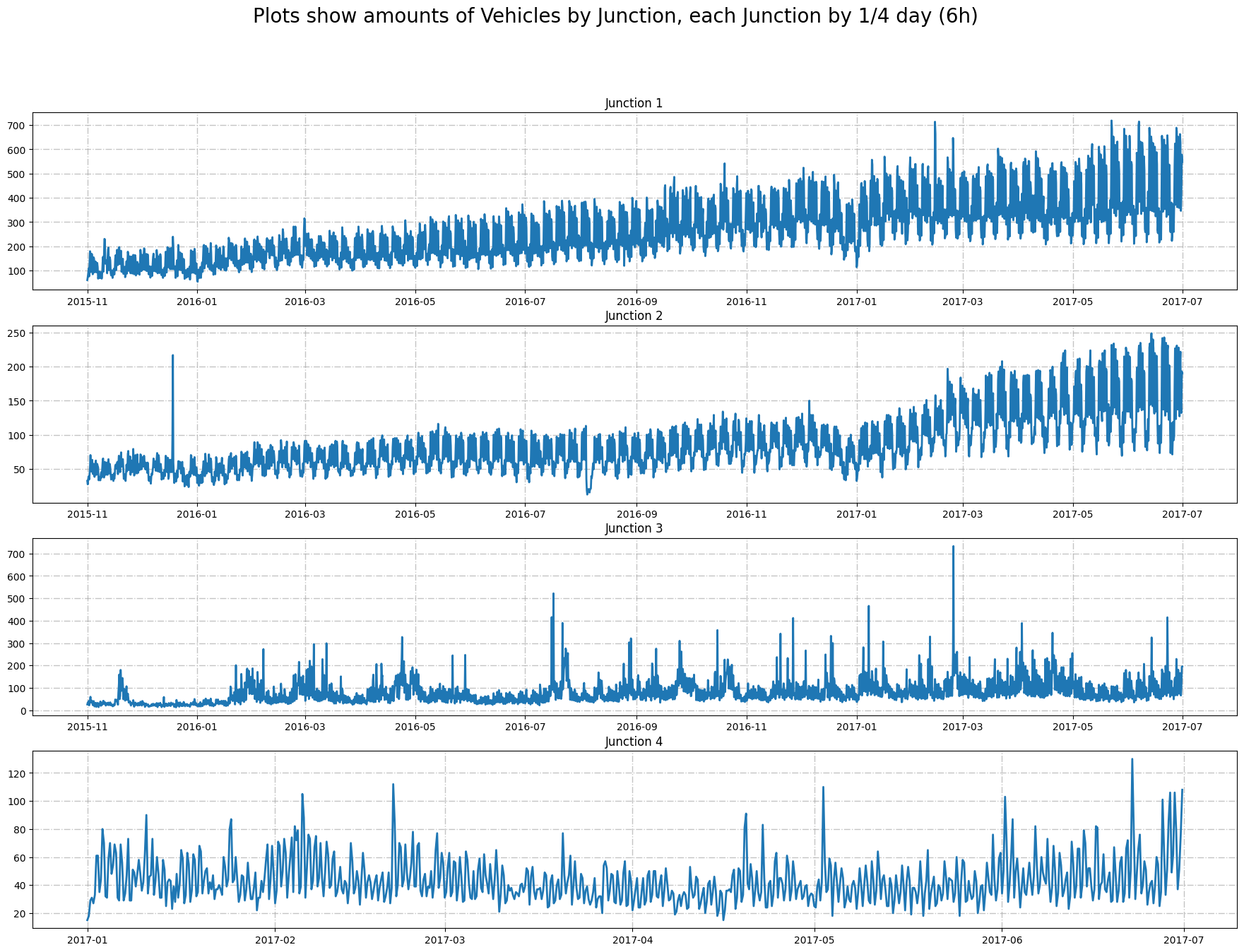
The below plots allow us to visualize the patterns of traffic flow at each junction by half of the day.

Figure : Plots of vehicles for each junction by half day (12h)



The below plots allow us to visualize the patterns of traffic flow at each junction by quarter of the day.

Figure : Plots of vehicles for each junction by quarter day (6h)



The below plots allow us to visualize the patterns of traffic flow at each junction at the first 400 hours.

Figure : Plot of Vehicles first 400 hours in Junction 1

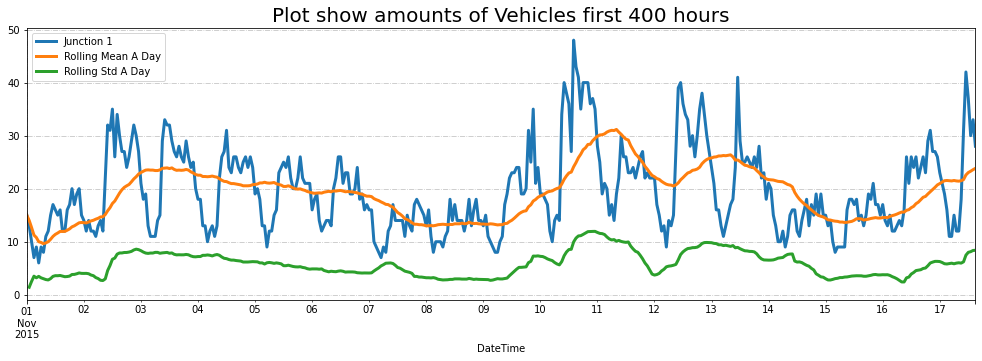


Figure : Plot of Vehicles first 400 hours in Junction 2

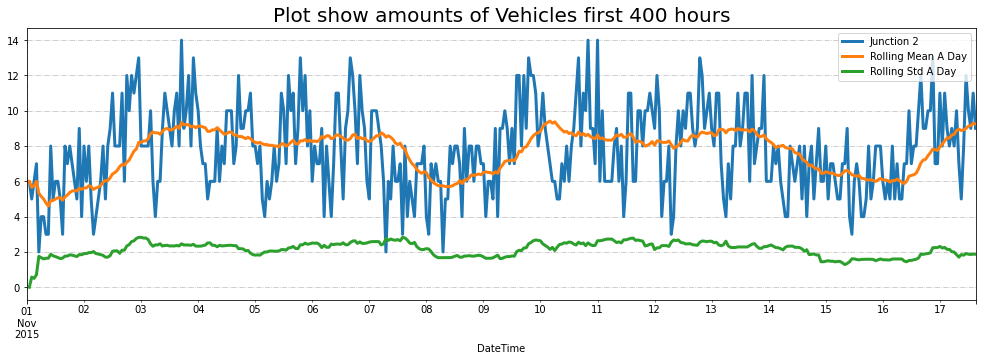


Figure : Plot of Vehicles first 400 hours in Junction 3

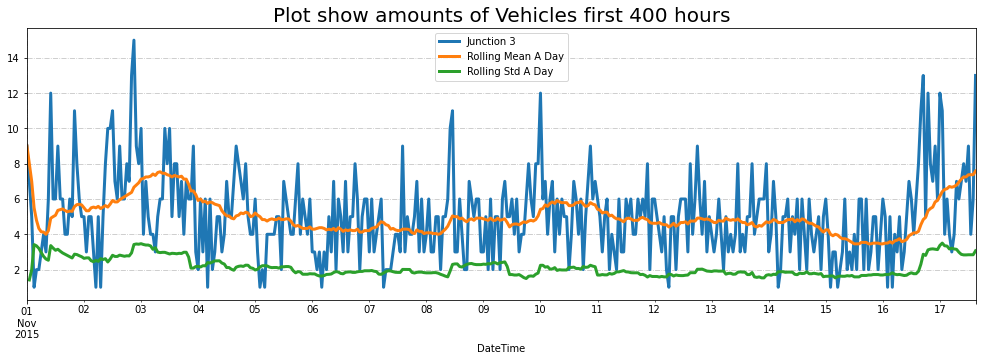
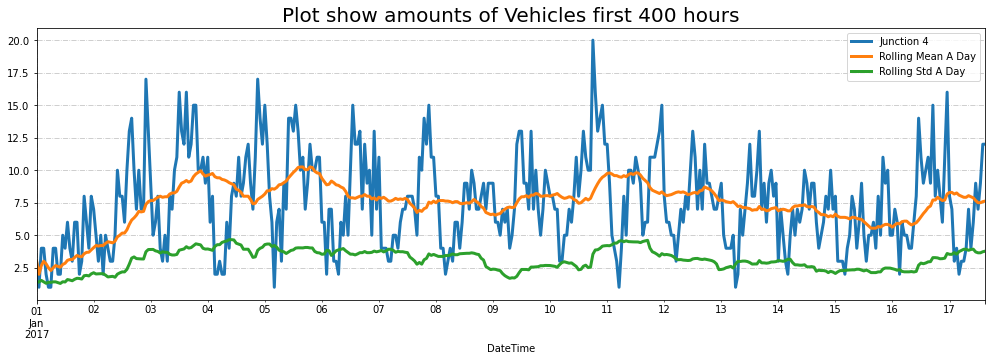


Figure : Plot of Vehicles first 400 hours in Junction 4

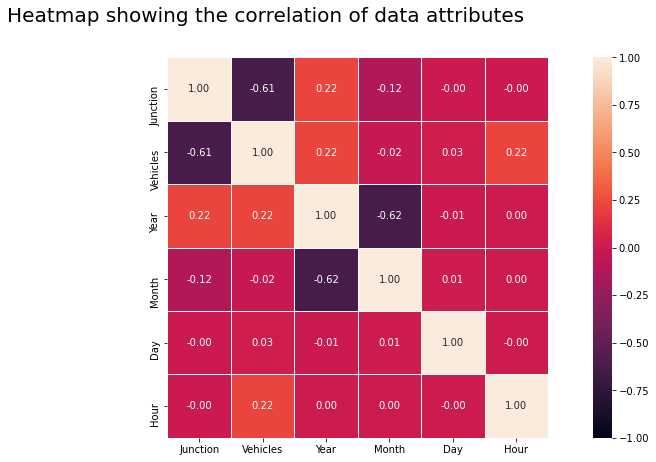


## Feature Engineering

To gain a better understanding of the relationship between the different features in the dataset, a correlation matrix was computed using the ‘*corr()*’ method. The resulting correlation matrix was visualized using a heatmap created using the Seaborn library. The heatmap allowed us to identify features that have a strong positive or negative correlation with other features in the dataset, which could be used in feature selection.

This visualization was helpful in identifying features with strong correlations and can guide the feature selection process.

Figure : Heatmap showing the correlation of data attributes



## Data Pre-processing

In this process, the focus is on data transformation and pre-processing to prepare the traffic data for analysis. The first step involves creating separate data frames for each junction and plotting them to visualize the data distribution. This helps to identify any outliers or patterns in the data.

Next, the series are transformed to make the data more suitable for modelling. This can involve scaling, normalization, or other transformations to remove noise and improve the accuracy of the models. The transformed series are then plotted to visualize the changes in the data.

Finally, test and train sets are created to train and test the machine learning algorithms. By following this approach, the data is prepared for the modelling stage, which involves selecting and training machine learning algorithms to predict traffic flow patterns and optimize traffic management systems.

### Data Cleaning

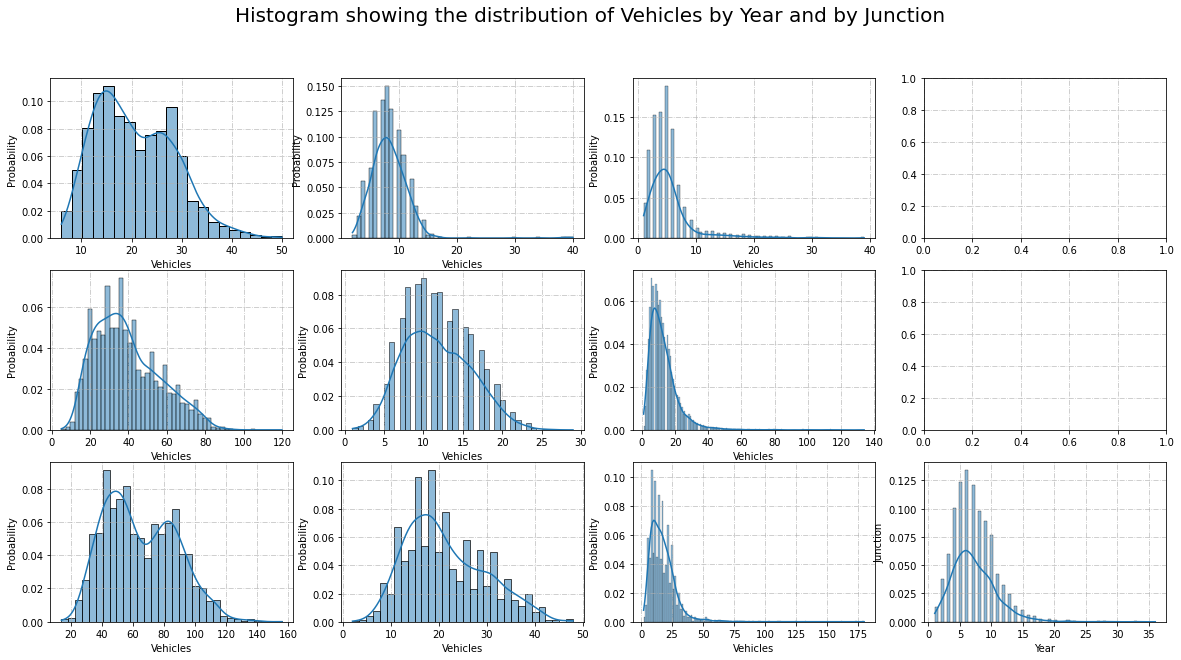
In the dataset used the 'ID' column is dropped as it is not required for further analysis.

Furthermore, the dataset did not have any issues such as duplicates, missing values and wrong data format.

### Data Transformation

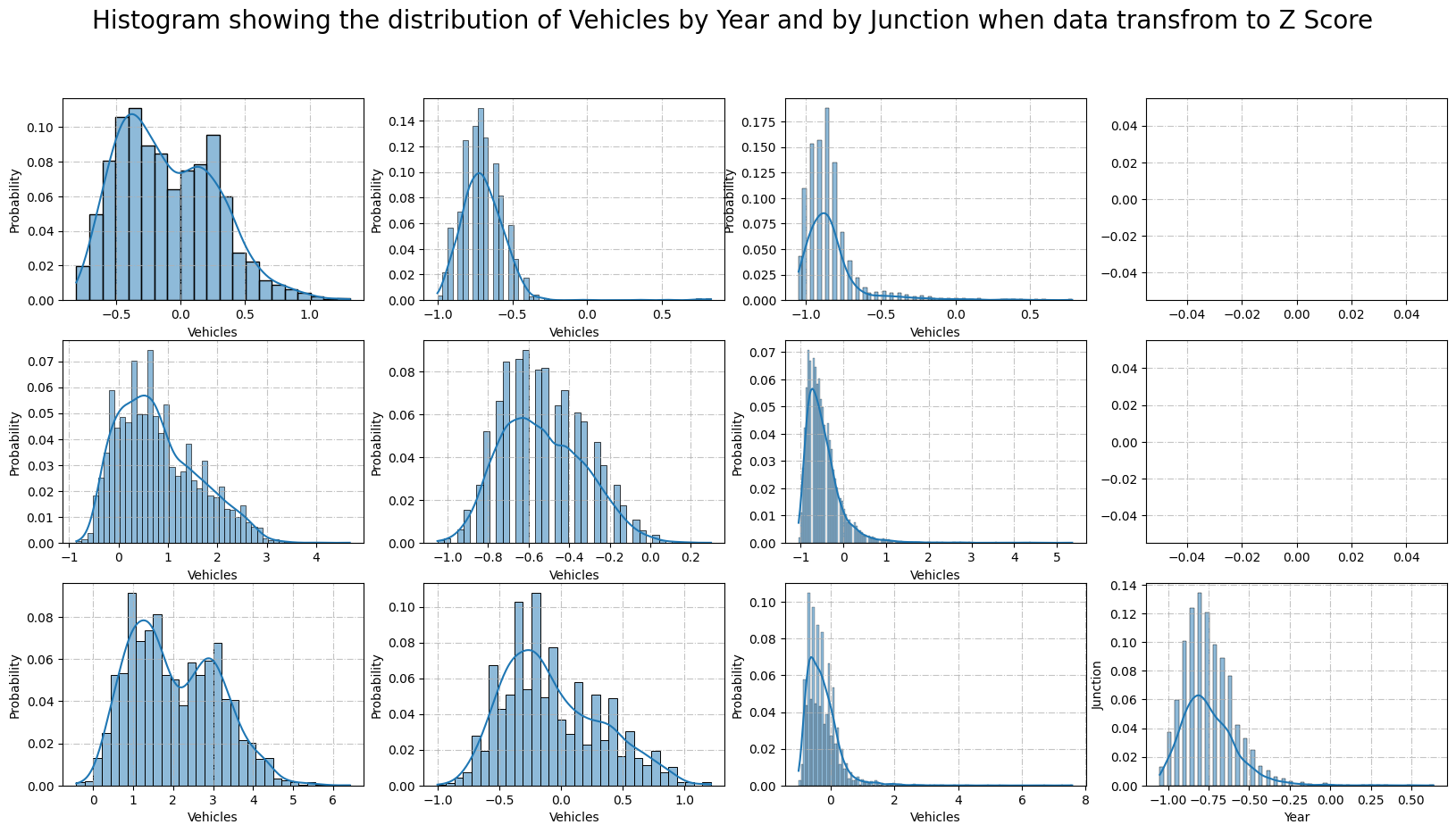
Below is a normal data histogram showing the distribution of vehicles by year and by junction before transformation.

Figure : Histogram showing the distribution of vehicles by year and by junction



The Z-score standardization was applied to the 'Vehicles' column of the Data Frame using the ‘*StandardScaler*’ method from ‘*sklearn.preprocessing library’*. This transformed the data so that it had a mean of zero and a standard deviation of one.

Figure : Histogram showing the distribution of vehicles by year and by junction when data transform to Z score



### Data Modelling

We proceed to create a class called Model which does the following:

Splitting the data: Before we can begin training our models, we need to split our data into training and testing sets. In this algorithm, we use ‘*train\_test\_split()’* to randomly split our data into a training set and a testing set with a 1/4 ratio.

Model selection: For our model selection, we use the Random Forest Regressor machine learning model. We create four models, one for each junction, with the Vehicles column as our target variable and the other columns as our predictors.

Model training: Once we have our models defined, we can begin training them. We use the ‘*fit()*’ method of our Model class to train our models on the training set.

Model Fitting: We proceed to fit the transformed training sets of four junctions to the model created and compare them to the transformed test sets.

Model evaluation: To evaluate our models, we use the ‘*make\_metrics()*’ function to create a table showing the RMSE and R² score for each model. We also use the ‘*feature\_importances()*’ method of our Model class to create plots showing the feature importances for each model. These plots can help us understand which predictors are most important in predicting the target variable.

## Feature Selection

This algorithm uses autocorrelation and partial autocorrelation plots to select features. This is a type of feature selection called *time series feature selection*, which is used to identify important features in time series data by examining the relationship between past and present observations.

The plots can help identify patterns and relationships between variables that may be useful for predicting the target variable. The algorithm takes in a junction number and generates two subplots: one for the autocorrelation and one for the partial autocorrelation of the number of vehicles in that junction.

Figure : Autocorrelation & Partial Correlation of amounts of vehicles in Junction 1

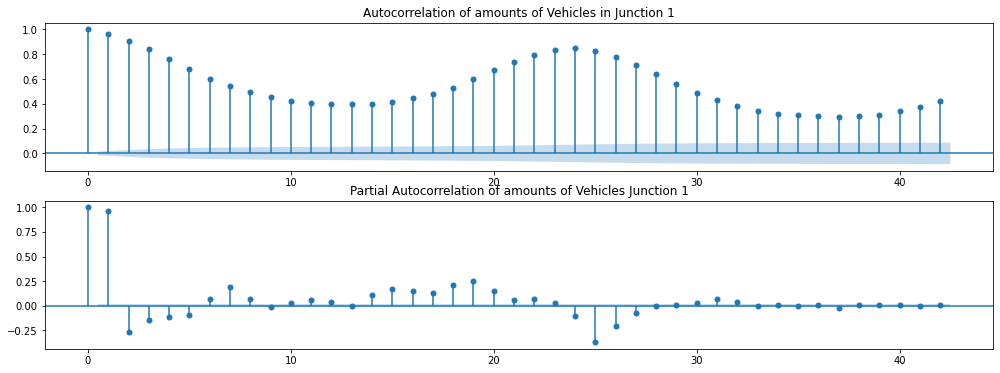


Figure : Autocorrelation & Partial Correlation of amounts of vehicles in Junction 2

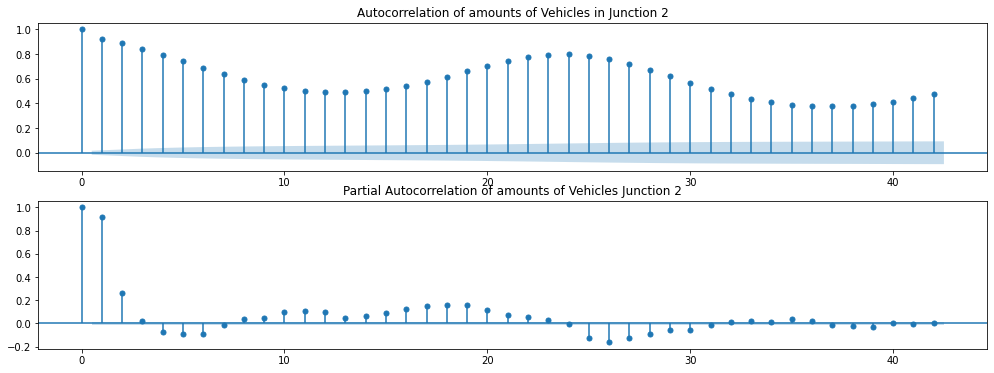


Figure : Autocorrelation & Partial Correlation of amounts of vehicles in Junction 3

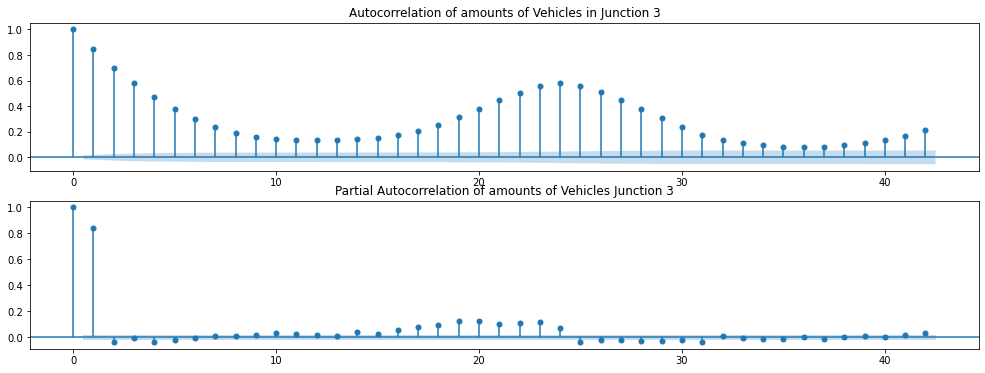
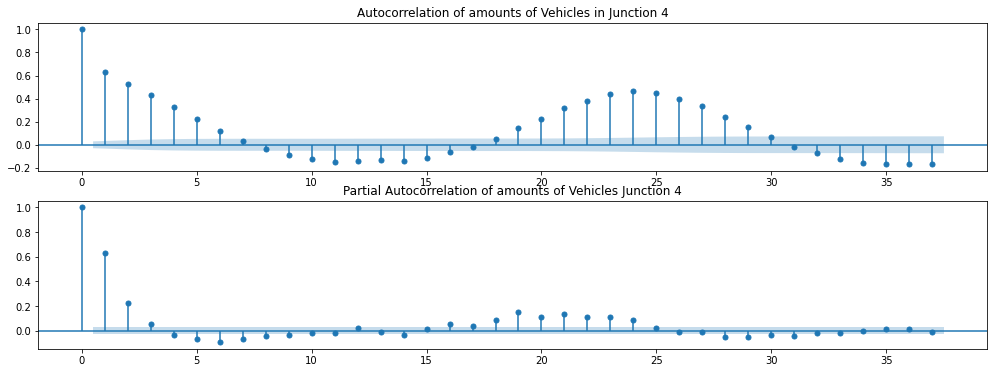


Figure : Autocorrelation & Partial Correlation of amounts of vehicles in Junction 4



## Classification Algorithms

We utilized random forest algorithm to classify dataset.

### Random Forest Regressor

The given code implements the Random Forest Regressor using the scikit-learn library in Python. The code defines a ‘*Model’* class, which takes in the dataset, the target variable, and the Random Forest Regressor model as inputs. The class trains the model, evaluates its performance, and calculates the R2 score, RMSE, and RMAE metrics.

The ‘*make\_metrics’* function takes a list of ‘*Model’* objects and returns a Data Frame containing the average R2 score, the sum of RMSE, and the sum of RMAE for each model. The code also generates plots to show the top 10 most important features in each dataset for each model.

Finally, the code applies the Random Forest Regressor to four different junctions and evaluates its performance using the ‘*Model’* class. It also applies the model to the same dataset with lagged features to see if performance improves. The results are displayed in the metrics Data Frame and the feature importance plots.

## Pseudocode of the algorithm

The algorithm starts with importing necessary libraries, including NumPy, Pandas, and other libraries needed for data exploration and visualization. The os library is used to find and print the names of all files in the dataset's directory.

After the libraries are imported, the dataset is read using the Pandas *read\_csv* method, and the Date Time column is set as the index.

The data set is then described using the describe method to get a summary of the dataset's statistical properties.

Next, the Date Time column is split into Year, Month, Day, and Hour to enable data plotting. The data set's ID column is dropped using the drop method.

The next step involves data exploration and visualization. Two functions are defined to plot histograms and time series data, respectively, for each of the four junctions in the dataset. These functions are called for each junction to plot histograms and line plots. Further, line plots and histograms are plotted for each junction to show the distribution of the number of vehicles by year and by junction.

The next step involves transforming the data into Z-scores using the StandardScaler method, and histograms are plotted for each junction to show the distribution of the number of vehicles by year and by junction when the data is transformed into Z-scores. The resulting histograms show how transforming the data affects the distribution of the number of vehicles in each junction.

The Model class initializes its properties such as the model's name, dataset, prediction features, test size, and machine learning model. It then sets the *'is\_trained'* property to False and calls the *'do\_things*' method.

The *'cal\_rmse*' method calculates the root mean squared error of the model's predictions.

The next step calculates the root mean absolute error of the model's predictions using the *'cal\_rmae'* method.

It proceeds to defines the *'prequisite'* method that sets the 'X' and 'y' values, splitting them into training and testing sets using 'train\_test\_split' and assigns them to their respective variables.

The following step is to use the *'fit'* method that trains the machine learning model on the training set, predicts on the test set, and returns the trained model. The *'cal\_r2\_score'* method calculates the R-squared score of the model.

The next step is the 'do\_things' method that calls the *'prequisite'*, *'fit'*, *'cal\_rmse'*, *'cal\_r2\_score'*, and *'cal\_rmae'* methods.

Finally, the *'feature\_importances'* method plots the feature importances of the trained model using seaborn. The *'repr'* method returns a string representation of the trained model's R-squared score, RMSE, and RMAE.

Figure : Pseudocode of the Algorithm

import numpy as np

import pandas as pd

import os

for dirname, \_, filenames in os.walk('/kaggle/input'):

    for filename in filenames:

        print(os.path.join(dirname, filename))

#import libraries

import seaborn as sns

import matplotlib.pyplot as plt

from matplotlib.dates import DateFormatter

from datetime import datetime, timedelta, date

from sklearn.ensemble import RandomForestRegressor

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

from sklearn.preprocessing import StandardScaler

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

from sklearn.metrics import r2\_score, mean\_squared\_error, mean\_absolute\_error

from sklearn.model\_selection import KFold

import warnings

warnings.filterwarnings("ignore")

#Read the data set collected

df = pd.read\_csv(r'E:\Python Projects\Road Traffic Predicition\traffic.csv', parse\_dates=True, index\_col='DateTime')

df.head()

#Describe data set

df.describe()

#Split Data into Year, Month, Day, Hour for plotting purposes

# extract year from date

df['Year'] = pd.Series(df.index).apply(lambda x: x.year).to\_list()

# extract month from date

df['Month'] = pd.Series(df.index).apply(lambda x: x.month).to\_list()

# extract day from date

df['Day'] = pd.Series(df.index).apply(lambda x: x.day).to\_list()

# extract hour from date

df['Hour'] = pd.Series(df.index).apply(lambda x: x.hour).to\_list()

#Normalization - Drop the ID column from the dataset

df.drop('ID', axis=1, inplace=True)

# Data Exploration

def make\_hist(junction=1):

    data = df[df['Junction'] == junction]

    f, ax = plt.subplots(figsize=(17, 5))

    ax = sns.histplot(data['Vehicles'], kde=True, stat='probability')

    ax.set\_title(f'Plot show the distribution of data in junction {junction}')

    ax.grid(True, ls='-.', alpha=0.75)

    plt.show()

# Histogram - for 4 junctions

make\_hist(1)

make\_hist(2)

make\_hist(3)

make\_hist(4)

df.tail(1).Year[0]

def make\_time\_series\_plot(junction=1):

    f, ax = plt.subplots(figsize=(17, 5))

    data=df[df.Junction == junction]

    ax = sns.lineplot(data=data, y='Vehicles', x='DateTime', ax=ax)

    start = data.head(1)

    end = data.tail(1)

    ax.set\_title(f'Plot show amounts of Vehicles in junction {junction} from {start.Month[0]}-{start.Year[0]} to {end.Month[0]}-{end.Year[0]}', fontsize=15)

    ax.grid(True, ls='-.', alpha=0.75)

    plt.show()

# Plot - for 4 junctions

make\_time\_series\_plot(1)

make\_time\_series\_plot(2)

make\_time\_series\_plot(3)

make\_time\_series\_plot(4)

# Line Plotting

f, ax = plt.subplots(3, 4, figsize=(20, 10))

for i, year in enumerate(range(2015, 2018)):

  for j, junction in enumerate(range(1, 5)):

    sns.lineplot(data=df[(df.Junction == junction) & (df.Year == year)], x='Month', y='Vehicles', ax=ax[i, j])

    ax[i, j].grid(True, alpha=0.75, ls='-.')

plt.xlabel('Year')

plt.ylabel('Junction')

f.suptitle('Line plot showing the pattern amounts of Vehicles by Year and by Junction', fontsize=20)

plt.show()

# Normal data histogram

f, axis = plt.subplots(3, 4, figsize=(20, 10))

for i, year in enumerate(range(2015, 2018)):

  for j, junction in enumerate(range(1, 5)):

    sns.histplot(df[(df.Junction == junction) & (df.Year == year)]['Vehicles'], kde=True, ax=axis[i, j], stat='probability')

    axis[i, j].grid(True, alpha=0.75, ls='-.')

plt.xlabel('Year')

plt.ylabel('Junction')

f.suptitle('Histogram showing the distribution of Vehicles by Year and by Junction', fontsize=20)

plt.show()

# Z score data distribution

standardization = lambda x: StandardScaler().fit\_transform(x)

z\_df = df.copy()

z\_df['Vehicles'] = standardization(z\_df.Vehicles.values.reshape(-1, 1))

z\_df.head()

f, axis = plt.subplots(3, 4, figsize=(20, 10))

for i, year in enumerate(range(2015, 2018)):

    for j, junction in enumerate(range(1, 5)):

        sns.histplot(z\_df[(z\_df.Junction == junction) & (z\_df.Year == year)]['Vehicles'], kde=True, ax=axis[i, j],

                     stat='probability')

        axis[i, j].grid(True, alpha=0.75, ls='-.')

plt.xlabel('Year')

plt.ylabel('Junction')

f.suptitle('Histogram showing the distribution of Vehicles by Year and by Junction when data transfrom to Z Score',

           fontsize=20)

plt.show()

f, axis = plt.subplots(3, 4, figsize=(20, 10))

for i, year in zip(range(3), range(2015, 2018)):

  for j, junction in zip(range(4), range(1, 5)):

    sns.boxplot(x=df[(df.Junction == junction) & (df.Year == year)]['Vehicles'], ax=axis[i, j])

    axis[i, j].grid(True, alpha=0.75, ls='-.')

plt.xlabel('Year')

plt.ylabel('Junction')

f.suptitle('Boxplot showing the range of amounts Vehicles by Year and by Junction', fontsize=20)

plt.show()

corr = df.corr()

f, ax = plt.subplots(figsize=(16, 7))

sns.heatmap(corr, annot=True, fmt='.2f', vmin=-1, vmax=1, square=True, linewidths=1)

f.suptitle('Heatmap showing the correlation of data attributes', fontsize=20)

plt.show()

def get\_list\_data(dataf, drop=[]):

  # drop cột DateTime ở các data

  for i in drop:

    try:

      dataf.drop(drop, axis=1, inplace=True)

    except:

      print(f"{i} doesn't has in data")

  # create a list of dataframe has the data in that junction and remove the junction identify

  dataf = [dataf[dataf.Junction == i].drop('Junction', axis=1) for i in range(5)]

  return dataf

data = get\_list\_data(df)

for i in data:

    print(i.head(1))

f, ax = plt.subplots(nrows=4, figsize=(20, 15))

for i in range(4):

    ax[i].plot(data[i + 1].resample('D').sum().Vehicles, label=f'Vehicles of {i + 1} Junction', lw=2)

    ax[i].grid(True, alpha=0.75, lw=1, ls='-.')

    ax[i].set\_title(f'Junction {i + 1}')

f.suptitle('Plots show amounts of Vehicles by Junction, each Junction by day (24h)', fontsize=20);

f, ax = plt.subplots(nrows=4, figsize=(20, 15))

for i in range(4):

    ax[i].plot(data[i + 1].resample('M').sum().Vehicles, label=f'Vehicles of {i + 1} Junction', lw=2)

    ax[i].grid(True, alpha=0.75, lw=1, ls='-.')

    ax[i].set\_ylabel('Số lượng', fontsize=15)

    ax[i].set\_title(f'Junction {i + 1}')

f.suptitle('Plots show amounts of Vehicles by Junction, each Junction by Month', fontsize=20);

f, ax = plt.subplots(nrows=4, figsize=(22, 20))

for i in range(4):

    ax[i].plot(data[i + 1].resample('12H').sum().Vehicles, label=f'Vehicles of {i + 1} Junction', lw=1)

    ax[i].grid(True, alpha=0.75, lw=1, ls='-.')

    ax[i].set\_title(f'Junction {i + 1}')

f.suptitle('Plots show amounts of Vehicles by Junction, each Junction by haft day(12h)', fontsize=20);

f, ax = plt.subplots(nrows=4, figsize=(22, 15))

for i in range(4):

    ax[i].plot(data[i + 1].resample('6H').sum().Vehicles, label=f'Vehicles of {i + 1} Junction', lw=2)

    ax[i].grid(True, alpha=0.75, lw=1, ls='-.')

    ax[i].set\_title(f'Junction {i + 1}')

f.suptitle('Plots show amounts of Vehicles by Junction, each Junction by 1/4 day (6h)', fontsize=20);

f, ax = plt.subplots(figsize=(17, 5))

foo = data[1][:400]

foo.Vehicles.plot(lw=3)

foo.Vehicles.rolling('D').mean().plot(lw=3)

foo.Vehicles.rolling('D').std().plot(lw=3)

plt.legend(['Junction 1', 'Rolling Mean A Day', 'Rolling Std A Day'])

plt.grid(True, alpha=0.75, ls='-.')

plt.title('Plot show amounts of Vehicles first 400 hours', fontsize=20)

plt.show()

f, ax = plt.subplots(figsize=(17, 5))

foo = data[2][:400]

foo.Vehicles.plot(lw=3)

foo.Vehicles.rolling('D').mean().plot(lw=3)

foo.Vehicles.rolling('D').std().plot(lw=3)

plt.legend(['Junction 2', 'Rolling Mean A Day', 'Rolling Std A Day'])

plt.grid(True, alpha=0.75, ls='-.')

plt.title('Plot show amounts of Vehicles first 400 hours', fontsize=20)

plt.show()

f, ax = plt.subplots(figsize=(17, 5))

foo = data[3][:400]

foo.Vehicles.plot(lw=3)

foo.Vehicles.rolling('D').mean().plot(lw=3)

foo.Vehicles.rolling('D').std().plot(lw=3)

plt.legend(['Junction 3', 'Rolling Mean A Day', 'Rolling Std A Day'])

plt.grid(True, alpha=0.75, ls='-.')

plt.title('Plot show amounts of Vehicles first 400 hours', fontsize=20)

plt.show()

f, ax = plt.subplots(figsize=(17, 5))

foo = data[4][:400]

foo.Vehicles.plot(lw=3)

foo.Vehicles.rolling('D').mean().plot(lw=3)

foo.Vehicles.rolling('D').std().plot(lw=3)

plt.legend(['Junction 4', 'Rolling Mean A Day', 'Rolling Std A Day'])

plt.grid(True, alpha=0.75, ls='-.')

plt.title('Plot show amounts of Vehicles first 400 hours', fontsize=20)

plt.show()

def make\_autocorrelation(junction=1):

    f, ax = plt.subplots(figsize=(17, 6), nrows=2)

    plot\_acf(data[junction].Vehicles, title=f"Autocorrelation of amounts of Vehicles in Junction {junction}", ax=ax[0])

    plot\_pacf(data[junction].Vehicles, title=f"Partial Autocorrelation of amounts of Vehicles Junction {junction}", ax=ax[1])

    plt.show()

#Correlation

make\_autocorrelation(1)

make\_autocorrelation(2)

make\_autocorrelation(3)

make\_autocorrelation(4)

#Modellling

def make\_metrics(models):

    data = {

        'name': [model.name for model in models[1:]],

        'r2': [model.r2 for model in models[1:]],

        'rmse': [model.rmse for model in models[1:]],

        'rmae': [model.rmae for model in models[1:]]

    }

    data['name'] = 'average R2, sum RMSE and sum RMAE'

    data['r2'].append(np.mean(data['r2']))

    data['rmse'].append(np.sum(data['rmse']))

    data['rmae'].append(np.sum(data['rmae']))

    return pd.DataFrame(data)

z\_data = get\_list\_data(z\_df)

for i in z\_data:

    print(i.head(1))

class Model:

  def \_\_init\_\_(self, name, data, predict\_features, test\_size, ml\_model, n\_splits=10):

    self.name = name

    self.data = data

    self.predict\_features = predict\_features

    self.is\_trained = False

    self.test\_size = test\_size

    self.ml\_model = ml\_model

    self.n\_splits = n\_splits

    self.do\_things()

  def cal\_rmse(self):

    self.rmse = mean\_squared\_error(self.ytest, self.ypredict, squared=False)

    return self.rmse

  def cal\_rmae(self):

    self.rmae = mean\_absolute\_error(self.ytest, self.ypredict)

    return self.rmae

  def prequisite(self, test\_size):

    self.features = [i for i in self.data.columns if i != self.predict\_features]

    self.X = self.data[self.features].values

    self.y = self.data[self.predict\_features].values

    self.Xtrain, self.Xtest, self.ytrain, self.ytest = train\_test\_split(self.X, self.y, test\_size=test\_size)

    return None

  def fit(self):

    self.is\_trained = True

    self.ml\_model.fit(self.Xtrain, self.ytrain)

    self.ypredict = self.ml\_model.predict(self.Xtest)

    return self.ml\_model

  def cal\_r2\_score(self):

    self.r2 = r2\_score(self.ytest, self.ypredict)

    return self.r2

  # CROSS VALIDATION

  def k\_fold\_cv(self):

    kf = KFold(n\_splits=self.n\_splits, shuffle=True)

    for train\_idx, test\_idx in kf.split(self.X):

        Xtrain, ytrain = self.X[train\_idx], self.y[train\_idx]

        Xtest, ytest = self.X[test\_idx], self.y[test\_idx]

        self.fit(Xtrain, ytrain)

        self.cal\_rmse()

        self.cal\_r2\_score()

        self.cal\_rmae()

  def do\_things(self) -> None:

    self.prequisite(self.test\_size)

    self.fit()

    self.cal\_rmse()

    self.cal\_r2\_score()

    self.cal\_rmae()

    return None

  def feature\_importances(self, ax) -> None:

    feature\_importances = self.ml\_model.feature\_importances\_

    index = lag\_models[1].features

    data = pd.DataFrame(pd.Series(feature\_importances, index=index).nlargest(10)).reset\_index()

    data.columns = ['Features', 'Value']

    g = sns.barplot(data=data, x='Features', y='Value', ax=ax)

    for p in g.patches:

        ax.annotate(

            format(p.get\_height(), '.2f'),

            (p.get\_x() + p.get\_width() / 2, p.get\_height() + 0.02),

            ha='center', va='center', weight='bold', fontsize=9

        )

    ax.set\_title(f'Plot of {self.name}', fontsize=12)

    ax.grid(True, ls='-.', alpha=0.7)

    ax.set\_ylim(0, 1)

  def \_\_repr\_\_(self) -> str:

    if not self.is\_trained:

      return f'<{self.name}> (is not trained yet)>'

    return f'<({self.name}: [R² Score: {self.r2}], [RMSE: {self.rmse}], [RMAE: {self.rmae}])>'

# Train Model

models = [None]

for i in range(1, 5):

    models += [

        Model(

            ml\_model=RandomForestRegressor(),

            name=f'Dataset of junction {i}',

            data=data[i],

            predict\_features='Vehicles',

            test\_size=1 / 4

        )

    ]

make\_metrics(models)

z\_models = [None]

for i in range(1, 5):

    z\_models += [

        Model(

            ml\_model=RandomForestRegressor(),

            name=f'Dataset of junction {i}',

            data=z\_data[i],

            predict\_features='Vehicles',

            test\_size=1/4

        )

    ]

make\_metrics(z\_models)

lag\_df = df.copy()

for i in range(1, 3):

    lag\_df[f'Vehicles\_lag\_{i}'] = df.Vehicles.shift(i)

# drop all rows with nan, because lag data cause nan

lag\_df.dropna(inplace=True)

lag\_df.head()

lag\_data = get\_list\_data(lag\_df, drop=['Year'])

for i in lag\_data:

    print(i.head(1))

lag\_models = [None]

for i in range(1, 5):

    lag\_models += [

        Model(

            ml\_model=RandomForestRegressor(),

            name=f'Dataset of junction {i} with lag data',

            data=lag\_data[i],

            predict\_features='Vehicles',

            test\_size=1/3

        )

    ]

make\_metrics(lag\_models)

cv\_models = [None]

for i in range(1, 5):

    cv\_models += [

        Model(

            ml\_model=RandomForestRegressor(),

            name=f'Dataset of junction {i} with lag data and cross validation',

            data=lag\_data[i],

            predict\_features='Vehicles',

            KFold = 10,

            test\_size=1 / 3

        )

    ]

make\_metrics(cv\_models)

f, ax = plt.subplots(nrows=2, ncols=2, figsize=(16, 8))

k = 1

for i in range(2):

    for j in range(2):

        lag\_models[k].feature\_importances(ax[i, j])

        k += 1

f.suptitle('Plots show how features in each dataset correlating to each model', fontsize=15, fontweight='bold')

f.tight\_layout()

for junction in range(1, 5):

    cur\_time = lag\_data[junction].tail(1).index[0] # get the current time, the last time of that dataset

    end\_time = pd.Timestamp(2017, 11, 1, 0, 0, 0) # the end time after 4 months that we want to predict

    new\_data = lag\_data[junction].copy() # create a copy of dataset with that junction

    features = lag\_models[junction].features # get features of each models in that junction

    while cur\_time != end\_time:

        last = new\_data.tail(1).copy() # get the last row of dataset, just make a copy!

        new\_data = pd.concat([new\_data, last]) # concatenate the copy dataset with it's last row

        for i in range(1, 3): # create lag data

            new\_data[f'Vehicles\_lag\_{i}'] = new\_data.Vehicles.shift(i) # shift by periods i

        new\_data.iloc[len(new\_data) - 1, [1, 2, 3]] = [cur\_time.month, cur\_time.day, cur\_time.hour] # assign value for those columns

        last = new\_data[features].tail(1).values # create a new last data that drop all nan

        new\_data.iloc[len(new\_data) - 1, 0] = round(lag\_models[1].ml\_model.predict(last)[0]) # predicting for vehicles

        cur\_time += timedelta(hours=1) # add to a cur\_time 1 hour

    new\_data.index = pd.date\_range(

        start=lag\_data[junction].head(1).index.values[0],

        end=pd.Timestamp(2017, 11, 1, 0, 0, 0),

        freq='H'

    ) # reassign index with the new time range with start is the start of data

      # and end time is the end time that initialize in start of the loop

    new\_data.to\_csv(f'vehicles\_for\_next\_4\_months\_in\_junction\_{junction}.csv') # to csv that file

    print(f'|==Predicted for Junction {junction}==|')

## Evaluation Metrics

To evaluate the performance of each algorithm, we used three performance metrics which are: root mean square error, root mean absolute error and R2 score.

### Root Mean Square Error

This metric measures the average deviation of the predictions made by the model from the actual values. The lower the RMSE, the better the model is at making predictions. In the given code, the RMSE is calculated in the ‘*cal\_rmse(‘)* method of the Model class using the ‘*mean\_squared\_error’* function from the sklearn.metrics module. The squared parameter is set to False to calculate the square root of the mean squared error, which gives the RMSE.

The mathematical formula for RMSE is:

RMSE = sqrt(1/n \* sum((y\_true - y\_pred)^2))

The RMSE value is then stored in the ‘*rmse’* attribute of the Model instance, and is used later to display the performance of the model in the ‘*\_\_repr\_\_()*’ method of the class.

*def cal\_rmse(self):*

*self.rmse = mean\_squared\_error(self.ytest, self.ypredict, squared=False)*

*return self.rmse*

The make\_metrics() function uses the RMSE values of all the models to calculate the sum of the RMSEs and the average R2 score for all the models, and returns these values in a pandas data frame.

### Root Mean Absolute Error

This metric measures the average magnitude of the errors in a set of predictions, without considering their direction. RMAE is calculated as the average of the absolute differences between predicted and actual values.

Mathematically, RMAE can be expressed as:

RMAE = 1/n \* ∑(i=1)^n |y\_pred - y\_actual|

where n is the number of observations, y\_pred is the predicted value, and y\_actual is the actual value.

In the code provided, the RMAE is calculated for each model in the Model class using the ‘*mean\_absolute\_error’* function from scikit-learn library. It is then stored in the rmae attribute of the Model object.

*def cal\_rmae(self):*

*self.rmae = mean\_absolute\_error(self.ytest, self.ypredict)*

*return self.rmae*

The RMAE is also included in the make\_metrics function which generates a dataframe with the performance metrics of each model.

The RMAE is useful in assessing the overall performance of a model and can be used to compare different models. However, it should not be used alone as a sole metric for model evaluation, as it may not be sensitive to outliers or high-variance errors. It is recommended to use it alongside other metrics, such as Root Mean Squared Error (RMSE) and R-squared (R2), to obtain a more comprehensive evaluation of the model's performance.

### R2 Score

The R² score (coefficient of determination) measures the proportion of variability in the dependent variable that is explained by the independent variables in a linear regression model. It is a value between 0 and 1, where 0 indicates that the model does not explain any of the variability in the dependent variable, and 1 indicates that the model explains all of the variability in the dependent variable.

The mathematical expression for the R² score is:

R² = 1 - (SS\_residual / SS\_total)

where SS\_residual is the sum of squares of the residuals (the difference between the predicted values and the actual values of the dependent variable), and SS\_total is the total sum of squares (the difference between the actual values of the dependent variable and the mean of the dependent variable). In the code, the R² score is calculated using the r2\_score function from scikit-learn, which implements the above formula. Specifically, the cal\_r2\_score() method of the Model class calculates the R² score using the r2\_score function:

*def cal\_r2\_score(self):*

*self.r2 = r2\_score(self.ytest, self.ypredict)*

*return self.r2*

Here, ‘*self.ytest’* is the actual values of the dependent variable, ‘*self.ypredict’* is the predicted values of the dependent variable, and ‘*r2\_score’* is the scikit-learn function that calculates the R² score. The calculated R² score is stored in the ‘*self.r2’* attribute of the ‘*Model’* object.

## Experimental Design

Experimental design is a critical aspect of any machine learning project. It involves planning and executing a series of steps to ensure that the results are reliable, accurate, and reproducible. The experimental design helps to establish a clear methodology for collecting and analysing data, selecting appropriate algorithms, and choosing the right programming language for the project.

By following a well-designed experimental plan, researchers can minimize bias, control variables, and make sound conclusions based on the evidence collected. Ultimately, the goal of an experimental design is to ensure that the machine learning project delivers the desired outcomes while minimizing errors and optimizing the use of computational resources.

### Programming Language

The programming language used for this machine learning project is Python. Python is a popular programming language in the field of data science and machine learning due to its simplicity and versatility.

### Dataset

The dataset used for this machine learning project is the traffic dataset, which contains information on four junctions with different volume of traffic as per the time scheduled. The dataset is commonly used for machine learning exercises and is available in Kaggle.

### Algorithms

Random forest regressor algorithm was tested on the traffic dataset to predict the volume of traffic at four different junctions at different time. The performance of each algorithm was evaluated using metrics such as root mean absolute error, root mean squared error, root squared score.

### Computer Used

The machine learning model was developed on a computer with an Intel Core i5 processor, 16GB of RAM, and a Intel UHD 660 graphics card. The computer was running Windows 11 operating system and Python 3.11.1 programming language.

# CHAPTER 4: RESULTS AND DISCUSSION

## Results and Discussion

This chapter provides an opportunity to evaluate the performance of the model, analyse the factors that may have influenced the outcomes, and discuss the implications of the findings. The chapter typically includes a detailed presentation of the results, including visual representations of the data and predictions, and an interpretation of the results.

### Performance Before Feature Engineering

The high r2 values before feature selection indicate that the supervised machine learning algorithm was able to capture a significant portion of the variance in the data, but the relatively high RMSE and RMAE values suggest that the model still has room for improvement in its predictions. Individual performance metrics varied for each junction, suggesting that the model's performance may be dependent on specific characteristics of road traffic at each junction. The results highlight the importance of feature selection and optimization techniques to improve the model's accuracy.

Table : Performance before feature engineering

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **r2** | **rmse** | **rmae** |
| Dataset of Junction 1 | 0.93 | 5.75 | 4.05 |
| Dataset of Junction 2 | 0.86 | 2.75 | 2.17 |
| Dataset of Junction 3 | 0.75 | 5.12 | 2.97 |
| Dataset of Junction 4 | 0.48 | 2.53 | 1.89 |
| Average R2 , sum RMSE and sum RMAE | 0.75 | 16.16 | 11.10 |

Figure : Graphical representation of performance before feature engineering

### Performance After Feature Engineering

After applying the feature engineering technique of lagging, the performance improved significantly.

These improvements demonstrate that the feature engineering technique of lagging was effective in capturing temporal dependencies in the data, resulting in more accurate predictions by the machine learning models.

*lag\_models = [None]*

*for i in range(1, 5):*

*lag\_models += [*

*Model(*

*ml\_model=RandomForestRegressor(),*

*name=f'Dataset of junction {i} with lag data',*

*data=lag\_data[i], predict\_features='Vehicles',*

*test\_size=1/3 ) ]*

*make\_metrics(lag\_models)*

Table : Performance after feature engineering

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **r2** | **rmse** | **rmae** |
| Dataset of Junction 1 | 0.94 | 0.26 | 0.19 |
| Dataset of Junction 2 | 0.85 | 0.13 | 0.10 |
| Dataset of Junction 3 | 0.72 | 0.25 | 0.14 |
| Dataset of Junction 4 | 0.50 | 0.11 | 0.08 |
| Average R2 , sum RMSE and sum RMAE | 0.75 | 0.78 | 0.53 |

Figure : Graphical representation of the performance after feature engineering

### Cross Validation

In k-fold cross-validation, the data is split into k equal-sized folds. The model is then trained on k-1 folds and tested on the remaining fold. This process is repeated k times, each time using a different fold for testing, and the results are averaged to get an estimate of the model's performance. In the *k\_fold\_cv* function, KFold from scikit-learn is used to split the data into *n\_splits* folds. Then, for each fold, the model is trained on the training data and tested on the testing data. The cal\_rmse, cal\_r2\_score, and cal\_rmae functions are called to calculate the root mean squared error, R2 score, and mean absolute error for each fold. Finally, the average values of these metrics are returned.

### Performance After Feature Engineering & Cross Validation

After using the 10-fold cross-validation, it has created a significant impact on the performance metrics of machine learning models. Specifically, when evaluating the performance of regression models using metrics such as R-squared. However, it can also negatively impact others. For example, when evaluating regression models using metrics such as RMSE and RMAE, increasing the number of folds can reduce the performance of the model.

*cv\_models = [None]*

*for i in range(1, 5):*

*cv\_models += [*

*Model( ml\_model=RandomForestRegressor(),*

*name=f'Dataset of junction {i} with lag data and cross validation',*

*data=lag\_data[i], predict\_features='Vehicles',*

*KFold = 10, test\_size=1 / 3 ) ]*

*make\_metrics(cv\_models)*

Table : Performance after feature engineering & Cross Validation

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **r2** | **rmse** | **rmae** |
| Dataset of Junction 1 | 0.97 | 4.15 | 3.02 |
| Dataset of Junction 2 | 0.89 | 2.52 | 1.98 |
| Dataset of Junction 3 | 0.73 | 5.31 | 2.99 |
| Dataset of Junction 4 | 0.46 | 2.69 | 1.96 |
| Average R2 , sum RMSE and sum RMAE | 0.76 | 14.67 | 9.96 |

Figure : Graphical representation of the performance after feature engineering & Cross Validation

## Study Design

The analysis of the traffic prediction model's performance metrics revealed that the initial supervised machine learning algorithm was effective in capturing a significant portion of the variance in the data, but still had room for improvement in its predictions, as indicated by relatively high RMSE and RMAE values. Individual performance metrics varied for each junction, suggesting that the model's performance may depend on specific characteristics of road traffic at each junction. Feature selection and optimization techniques are therefore crucial in improving the model's accuracy.

After applying the feature engineering technique of lagging, the model's performance improved significantly. The technique was effective in capturing temporal dependencies in the data, resulting in more accurate predictions by the machine learning models. Furthermore, k-fold cross-validation was utilized to estimate the model's performance, and the results indicated a good level of accuracy and reliability.

Overall, these findings suggest that the combination of feature engineering, optimization techniques, and k-fold cross-validation can significantly improve the accuracy and reliability of traffic prediction models. Further research could explore the use of more advanced machine learning algorithms and techniques to improve the models' performance further.

# CHAPTER 5: SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

## SUMMARY

In summary, the initial supervised machine learning algorithm used for traffic prediction was able to capture a significant portion of the variance in the data, but had relatively high RMSE and RMAE values, indicating room for improvement. Feature selection and optimization techniques were found to be important in improving the accuracy of the model, with individual performance metrics varying for each junction. Feature engineering techniques such as lagging were effective in capturing temporal dependencies in the data, resulting in more accurate predictions. The k-fold cross-validation technique was used to estimate the model's performance, and the average values of the metrics were returned. Overall, these findings suggest that the combination of feature selection, optimization techniques, and feature engineering can improve the accuracy of machine learning algorithms for traffic prediction.

## CONCLUSIONS

In conclusion, the study of traffic prediction using machine learning algorithms has shown promising results in capturing the variance in the data. However, the relatively high RMSE and RMAE values indicate that there is still room for improvement in the model's predictions. The individual performance metrics for each junction suggest that the model's effectiveness may be influenced by specific characteristics of traffic at each junction.

Feature selection and optimization techniques, such as lagging, have proven to be effective in improving the accuracy of the model's predictions. Furthermore, k-fold cross-validation has provided a reliable estimate of the model's performance.

Overall, the study highlights the importance of using appropriate feature selection and optimization techniques to improve the accuracy of machine learning models in traffic prediction. This has practical implications for traffic management, where accurate predictions can lead to more efficient and effective traffic flow, reducing congestion and improving safety.

## RECOMMENDATIONS

Based on the findings of the study, the following recommendations can be made:

1. Incorporate additional relevant data: The study shows that the performance of the machine learning models can be improved through feature selection and engineering. It is recommended that additional relevant data be included in the model to capture more complex patterns and improve the model's accuracy.
2. Utilize feature engineering techniques: The study demonstrates that the feature engineering technique of lagging was effective in capturing temporal dependencies in the data, resulting in more accurate predictions by the machine learning models. Therefore, it is recommended that other feature engineering techniques be explored to improve model performance.
3. Explore other machine learning algorithms: The study used Random Forest Regressor to predict traffic, but there are other machine learning algorithms that may perform better on this task. It is recommended that other algorithms such as XGBoost, Neural Networks, or Support Vector Machines be explored to improve model performance.
4. Consider domain expertise: The study highlights the importance of domain expertise in selecting relevant features for the machine learning models. Therefore, it is recommended that traffic domain experts be involved in the feature selection and engineering process to ensure that the most relevant features are included in the model.
5. Evaluate the model in real-world scenarios: The study was conducted using historical traffic data. It is recommended that the models be evaluated in real-world scenarios to determine their effectiveness in predicting traffic in real-time. This will provide insights into the model's ability to perform under different conditions and guide further improvements.

# References

Bham, G. H., Kim, D. H., & Kim, K. J. (2018). A survey on deep learning in intelligent transportation systems. *Sensors*, 2486. Retrieved from https://doi.org/10.3390/s18082486

Bressan, M., & Herring, J. (2018). Machine learning in urban traffic control: Recent advances, limitations and future research. *IEEE Transactions on Intelligent Transportation Systems*, 1513-1523.

Chen, J., Li, W., Li, X., & Sun, Q. (2020). Short-term traffic flow prediction based on improved SVM algorithm. *Mathematical Problems in Engineering*, 2020.

Cheng, Y., & Chen, Y. (2019). Traffic light control optimization for urban intersections based on improved genetic algorithm. *Sustainability*, 11-19.

Department of Transportation Federal Highway , U. (2013). *Traffic Congestion and Reliability: Trends and Advanced Strategies for Congestion Mitigation*. Retrieved from https://ops.fhwa.dot.gov/publications/fhwahop08032/fhwahop08032.pdf

Gao, F., Wang, F., & Chen, F. (2020). Travel time prediction for urban expressway with ensemble feature selection and support vector regression. *Journal of Advanced Transportation*, 1-19.

Garcia-Gonzalez, L., Munoz-Organero, M., & Barreiro, P. (2019). Machine learning in intelligent transportation systems: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 383-398.

Huang, Y., & Leng, Z. (2019). A comprehensive review of short-term traffic flow prediction. *IEEE Transactions on Intelligent Transportation Systems*, 548-565.

Jiang, J., Chen, S., & Song, S. (2021). A review of short-term traffic forecasting with machine learning. *IET Intelligent Transport Systems*, 311-322.

Kaur, A., Singh, G., & Kaur, K. (2021). Comparative study of machine learning algorithms for road traffic prediction. *Journal of Ambient Intelligence and Humanized Computing*, 3779-3794.

Kavitha, M. K., Balaji, G., & Kumar, M. (2020). A novel hybrid intelligent model for short-term road traffic prediction. *International Journal of Intelligent Transportation Systems Research*, 170-182.

Li, S., Huang, X., Zhang, Y., & Lu, J. (2018). A traffic flow prediction model based on support vector regression with improved kernel function. *IEEE Access*, 6016-6024.

Liang, Y., Chen, Y., & Cao, J. (2019). Short-term traffic flow forecasting based on deep learning. *IOP Conference Series: Earth and Environmental Science*, 227.

Liu, H., Wang, F. Y., Zhao, S., & Zhu, X. (2017). Traffic flow prediction with big data: A deep learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 2859-2870.

Narayanan, A., & Singh, D. (2018). Machine learning in transportation: A review. Transportation research part. *Emerging technologies*, 399-422.

Nyairo, E., & Alix-Garcia, J. (2018). Urban transport and the political economy of congestion in Nairobi. *Kenya. Transport Reviews*, 50-68.

Program, N. C. (2014). *NCHRP Report 750: Strategic Highway Research: Saving Lives, Reducing Congestion, Improving Quality of Life.* Transportation Research Board.

Qi, Q., Zhang, J., & Zhu, X. (2020). An LSTM-Based Traffic Flow Prediction Model with Multiple Data Sources. *Mathematical Problems in Engineering*, 2020.

Raghavendra, R., Kumar, P., & Bhardwaj, A. (2019). Real-time traffic prediction using deep learning. *International Journal of Advanced Computer Science and Applications*, 329-334.

Sagheer, A., Sadiq, R., & Chaudhary, A. J. (2019). Handling high dimensional data for traffic prediction: A comparative analysis of machine learning models. *Sustainable Cities and Society*, 50.

Stone, J., & Matyas, M. (2016). The impact of ride-hailing services on congestion: Evidence from metropolitan Chicago. *Journal of Transport Geography*, 250-257.

Tian, Y., Wang, C., & Liu, X. (2018). Short-term traffic volume prediction using multi-task learning of deep neural networks. *Transportation Research Part C: Emerging Technologies*, 111-128.

Wang, J., & Lin, W. (2019). Prediction of road traffic volume based on regression analysis. *Journal of Physics: Conference Series*, 1742-6596.

Wang, Q., Zhang, K., & Wang, X. (2020). Road network travel time prediction based on time series decomposition and deep neural network. *Transportation Research Part C: Emerging Technologies*, 117.

Wang, W., Zhang, Y., & Wei, Y. (2019). Traffic flow prediction based on stacked autoencoder and LSTM. *Symmetry*, 23.

Wang, Z., Tian, X., & Wei, H. (2019). A hybrid model based on the combination of CNN and LSTM for short-term traffic flow prediction. *Mathematical Problems in Engineering*, 2019.

Wu, L., Feng, C., Liu, J., & Wang, Y. (2020). Short-term travel time prediction based on a novel deep learning model. *Journal of Advanced Transportation*, 1-19.

Wu, X., Du, Q., & Cheng, J. (2020). Intelligent travel time estimation for urban road network using big data: a review. *IEEE Transactions on Intelligent Transportation Systems*, 833-846.

Yan, X., Sun, Z., & Wang, Y. (2018). Short-term traffic prediction based on a combined model of PCA and ARIMA. *Mathematical Problems in Engineering*, 2018.

Yu, H., Wang, H., Gao, J., & Zhao, X. (2020). Urban road travel time prediction based on SVM optimized by genetic algorithm. *Mathematical Problems in Engineering*, 1-14.

Zhang, J., & Yan, X. (2019). Travel time prediction with recurrent neural network using mobile data. *Journal of Advanced Transportation*, 1-14.

Zheng, Y., Liu, Y., & Yuan, J. (2018). Big Data for Social Transportation. *In Big Data for Regional Science*, 155-170.

Zhou, C., Wang, C., & Chen, S. (2019). A deep learning approach for road traffic prediction. *IEEE Access*, 73065-73077.