Predictive Maintenance by the Detection of Gradual Faults in an IoT-Enabled Public Bus

by

Gautam Vira

Thesis submitted to the
University of Ottawa
In partial fulfillment of the requirements
For the Master's degree in
Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

The widespread incorporation of Internet of Things (IoT) devices in various systems such as mobile phones, vehicles, and security systems is used to collect data. These large volumes of data can be used to train models to make predictions about various things. One such application that uses data from IoT devices is called predictive maintenance. Predictive maintenance involves collecting data from machines and using algorithms to analyze the machine's condition or determine if the machine requires maintenance or repairs. Prior to such a predictive approach, organizations would have to conduct scheduled check-ups for their vehicles to find out if the vehicles needed any repairs or maintenance; however, the scheduled check-ups resulted in downtime and shut-downs and so this proved to be costly and time-inefficient, as many of the vehicles would not require any kind of maintenance or repairs. The shift to predictive maintenance addresses these challenges by leveraging the continuous data stream from IoT devices to predict and prevent failures.

This thesis presents a clustering-based algorithm to perform predictive maintenance by detecting potential faults and gradual deterioration for IoT-based buses. The algorithm is tested for the cooling system and the engine torque system of a bus from the Société de Transport de l'Outaouais (STO) transit service. In order to overcome the lack of availability of real-world data, this work also generates two synthetic time series datasets that contain sensor readings belonging to the cooling system and the engine torque system to simulate normal buses, and buses with underlying potential faults and gradual deterioration that the standard maintenance systems would not detect. The synthetic data generation process leverages deep learning frameworks such as Generative Adversarial Networks and Long Short-Term Memory networks along with other statistical methods to make the data appear as realistic as possible. The results from the experiments conducted using the predictive models validate the reliability and accuracy of the proposed algorithm. The results showed that using the predictive maintenance algorithm resulted in all the faults being clustered accurately and appropriately based on the type of fault.

This thesis demonstrates that predictive maintenance not only enhances cost and time efficiency but also significantly improves user safety by enabling preemptive maintenance actions. While the predictive models implemented in this thesis focus on the cooling and engine torque systems, the methodology proposed is flexible and can be extended to other subsystems. Overall, this work contributes to the field of predictive maintenance by presenting an efficient and practical solution that ensures the reliability and safety of transportation systems.

Acknowledgements

I would like to express my deepest gratitude to everyone who has supported me during my journey to completing my thesis. First, I would like to sincerely thank my supervisors, Dr. Tet Yeap and Dr. Iluju Kiringa, for their invaluable guidance, support, and encouragement throughout the course of this research. Their expertise and insightful feedback have been instrumental in shaping this thesis, and I am immensely grateful for their mentorship and patience.

I would like to thank my family and friends, especially my parents, for their unwavering love and support. Their encouragement has been my greatest source of strength during difficult times. This work would not have been possible without their belief in me and their sacrifices.

I would also like to acknowledge the authors of the works cited in this thesis for inspiring me and providing a foundation for my work. Lastly, I would like to thank the University of Ottawa for providing me with the knowledge and resources that helped me in conducting my research.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The widespread incorporation of Internet of Things (IoT) devices in various systems such as mobile phones, vehicles, and security systems to gather data has contributed to the emergence of Industry 4.0 [35]. These IoT devices collect large volumes of data, known as big data, which prove to be very useful in gathering valuable insights about the user's experience and the device or component's performance. One such artificial intelligence application is called predictive maintenance, which uses such data, has gained popularity and is being used by various organizations.

Predictive maintenance utilizes IoT devices embedded in different kinds of devices and transport systems − road vehicles, railways, and airplanes − to collect data about their performance, and uses algorithms and methods such as active databases (rule-based), mathematical models, machine learning, and deep learning to predict values such as the remaining useful life (RUL) [119] of the machine or determine if the machine requires maintenance or repairs, that is, it predicts the deterioration of a machine or a group of machines that form a system.

Prior to such a predictive approach, organizations would have to conduct scheduled check-ups for their vehicles to find out if the vehicles needed any repairs or maintenance; however, the scheduled check-ups resulted in downtime and shut-downs and so this proved to be costly and time-inefficient, as many of the vehicles would not require any kind of maintenance or repairs. Furthermore, unexpected breakdowns of vehicles resulted in the organizations incurring heavy costs. Such unexpected breakdowns are common when organizations do not conduct periodic checks to save resources and time.

There are several maintenance strategies employed by organizations such as the aforementioned ones, they can be summarized into three primary categories −

1. Corrective maintenance: this is a reactive measure in which a component or system has suffered failure and needs to be urgently repaired or replaced. Corrective maintenance occurs as a result of unexpected and unscheduled events.

2. Preventive maintenance: this is a measure that most organizations have in place to maintain the health and ensure the performance of the machinery. This measure involves conducting periodic checks to make sure that nothing is wrong with the equipment, and if a fault is found, maintenance activities are conducted accordingly.

3. Predictive maintenance: this is a measure that estimates and predicts the probability of a component or equipment failing in the future. Such an approach allows organizations to avoid unexpected failures and skip non-required periodic check-ups that could be very costly.

The shift to predictive maintenance addresses these challenges by leveraging the continuous data stream from IoT devices to predict and prevent failures. By identifying potential issues before they escalate into major problems, predictive maintenance helps organizations optimize their maintenance schedules, reduce downtime, and minimize the risk of unexpected breakdowns. This proactive approach not only improves the efficiency and cost-effectiveness of maintenance activities but also enhances the reliability and longevity of the equipment.
In conclusion, the integration of IoT devices and the subsequent rise of Industry 4.0 have revolutionized maintenance strategies through the implementation of predictive maintenance. This approach harnesses the power of big data and advanced analytics to provide actionable insights, allowing organizations to transition from reactive to proactive maintenance practices. The result is a more efficient, cost-effective, and reliable maintenance system that significantly reduces downtime and extends the useful life of critical assets.

1.1 Motivation

Most organizations contain preventive measures in place such as conducting periodic check-ups to avoid unexpected failure of equipment [140]. However, most scheduled check-ups result in no fault being found and prove to be unnecessary. A periodic maintenance approach such as this can prove to be inefficient and costly but it is still required to ensure the health status of vehicles. Not doing so would potentially result in unexpected failures of equipment causing downtime and shut-downs. Such critical failures can prove to be extremely costly and must be avoided as much as possible. A way of contemplating the costs associated would be to imagine a car breaking down abruptly, this would be followed by having the car towed, diagnosing the vehicle for faults, and performing maintenance activities such as repairs and/or replacements. If the same event is scaled to the extent of an aircraft breaking down, or to an organization that owns a fleet of aircraft, then the consequences could be severely drastic and costly. The monetary costs associated with such a fleet system would be significant, in addition to this cost, and most importantly, the safety of passengers is of the highest priority and unexpected failures could put them in harm's way and such loss is immeasurable. Predictive maintenance is an approach that aims to solve these problems, it seeks to avoid requiring period check-ups as well as seeks to notify users of potential failures so that maintenance or repairs can be conducted in advance and avert any potential break-downs.
Predictive maintenance offers a solution to these issues. It aims to eliminate the need for periodic check-ups and provides timely notifications of potential failures. This proactive approach allows maintenance or repairs to be conducted in advance, preventing any potential breakdowns. Predictive maintenance employs advanced technologies and data

analytics to monitor equipment consistently, analyze historical data, and predict when a component is likely to fail. This method not only saves resources but also ensures safety and increases efficiency by addressing issues before they lead to critical failures.
The work in this thesis aims to exactly do that, save resources, ensure safety, and increase efficiency, by providing an approach for detecting potential faults and failures that can isolate the cause of deterioration to a specific component or component group. This specificity simplifies the diagnostic process and ensures that maintenance is performed only when necessary. This targeted approach minimizes downtime, reduces unnecessary maintenance activities, and enhances the overall reliability and safety of the equipment.
In summary, the shift from periodic maintenance to predictive maintenance represents a significant advancement in asset management. It aligns maintenance activities more closely with the actual condition of the equipment, thereby optimizing maintenance schedules, reducing costs, and ensuring that critical systems operate without unexpected interruptions. This thesis contributes to this field by developing methods to detect and diagnose potential faults more effectively, ensuring that organizations can maintain their equipment in the most efficient and safe manner possible.

1.2 Challenges

Although the shift from traditional maintenance activities to predictive maintenance offers significant advantages and benefits, there are challenges that need to be overcome to do so [32, 93]. Some of the significant challenges are as follows:

1. The development of reliable predictive maintenance models requires large amounts of data for training and testing, however, in most cases, sufficient data is not available.

2. In addition to the quantity of data, the quality of the data available needs to be good to ensure accuracy from the models and their ability to cover a wide range of possible scenarios.

3. Most vehicles have a complex mechanism that includes various components. Developing a generalized system for such a mechanism that takes into account the different operations and attributes of the components is difficult.

4. Many organizations may be concerned about the privacy and security of their data; they may be hesitant to allow experts to directly collect and analyze their data.

5. Scaling predictive maintenance systems from a single vehicle to a large fleet can require significant additional resources and extensive knowledge to distinguish small abnormalities within a fleet.

The challenges that come with the incorporation of predictive maintenance seem daunting but the advantages of predictive maintenance still make it a beneficial choice. The work in this thesis attempts to tackle several of these challenges such as the data quality and quantity issues along with privacy concerns can be mitigated by advanced synthetic data

generation methods, and the complexity and scalability concerns are addressed in this thesis by presenting a predictive maintenance approach that can be scaled to a fleet of vehicles.

1.3 Methodology

The work in this thesis presents an approach to conduct predictive maintenance by detecting the gradual deterioration (or gradual faults) in the components of a public bus. Gradual faults are often neither noticeable by the threshold mechanisms built around the sensors on a vehicle nor visible to the eye until the deterioration progresses to a severe level, resulting in either component failure or an immediate need for maintenance.
The data used in this thesis was collected in [66] from a public bus of the Société de Transport de l'Outaouais (STO) transit service of the Ottawa-Gatineau region. The data (containing sensor readings relevant to the cooling system and engine torque system) from a single bus was used to generate synthetic data belonging to the cooling system to simulate 40 buses using Gretel Synthetic's [52] PyTorch implementation of the doppleGANger [74] (DGAN) model and then manually introducing temporal patterns and trends that were not captured by the generative model. Similarly, the data collected from the bus was also used to generate synthetic data pertaining to the engine torque system to simulate 38 buses. This was done using a combination of probability distribution fitting, rule-based estimations, and the DGAN model.
After generating synthetic data, gradual deterioration in components was introduced to a small percentage of the buses. The faults introduced are based on common knowledge and heuristics, which are based on understanding the causes and effects of faults discussed across numerous automotive forums and blogs. Finally, the generated data was used to train clustering models to identify the sensor readings with deterioration or faults. This way, individual models that identify abnormal sensor readings belonging to components from the same bus can be grouped to identify the potential fault.
In summary, the approach presented in this thesis leverages advanced data generation and machine learning techniques to implement a predictive maintenance system for public buses. By simulating a larger fleet and introducing realistic fault scenarios, the study demonstrates the potential of clustering models to detect gradual deterioration, ultimately contributing to more efficient and reliable maintenance practices.

1.4 Contributions

The main contributions of this thesis are as follows:

1. Publication of Preliminary Results: Presented a paper [133] at the 2024 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE) outlining the proposed methodology and initial results, which laid the foundation for the more comprehensive analysis presented in this thesis.

2. A clustering approach to conduct predictive maintenance for an IoT-based vehicle. The presented algorithm is tested on the cooling system and the engine torque system of a public bus, but it could be extended to several other machines and vehicle subsystems.

3. Two datasets containing a total of 78 simulated buses −

   (a) A dataset containing synthetic sensor readings belonging to the cooling system to simulate 40 buses along with additional test cases.

   (b) A dataset containing synthetic sensor readings pertaining to the engine torque system for 38 simulated buses.

   Some of the simulated buses have gradual faults introduced to indicate functionality and performance-related problems.

The datasets generated in this work incorporate various seasonal and operational factors that may affect sensor readings without indicating faults. Hence, the datasets presented in this work can be used to train and test machine learning algorithms − supervised and unsupervised − for similar tasks while ensuring robustness.

1.5 Limitations

The limitations of the work in this thesis are as follows:

1. The presented predictive maintenance approach was tested and validated using synthetic data, however, the approach's credibility to work in real-world settings can only be confirmed by testing it on real data from buses over a period of time.

2. A thorough effort was made to generate synthetic data to represent real data as closely as possible, however, several factors such as the driver's behavior and driving patterns that may affect the sensor readings cannot be accounted for precisely.

3. The presented approach requires sensor data to be preprocessed to ensure that the clustering approach can detect gradual faults without its performance getting impeded by the noisy structure of sensor data.

4. Due to technical issues, the data collected in [66] was only for a duration of ten hours, hence, the synthetic data generated in this work was based on the analysis of sensor data only from a single bus and only for a small duration.

5. Due to availability reasons, the presented approach could only be tested on two subsystems of a bus. Real data for other subsystems such as the engine speed system and the braking system were not available for analysis and generation.

1.6 Structure of the thesis

The outline of the thesis is as follows − Chapter 2 contains a thorough survey and analysis of the existing predictive maintenance algorithms and methodologies across different modes of transport, a background of synthetic data generation techniques and approaches, and a background of popular machine learning algorithms mentioned in this thesis; Chapter 3 contains the proposed predictive maintenance algorithm, detailed explanations of the steps in the algorithm, and relevant preprocessing techniques and hyperparameter tuning techniques; Chapter 4 contains the analysis of the sensor data and the methodology to generate synthetic data as well as the fault introduction process; Chapter 5 contains the experiments conducted using the proposed predictive maintenance algorithm on the synthetic data generated and the results of the experiments; and lastly, Chapter 6 concludes the thesis by providing a summary of the work and pointing at interesting future directions of research in the domain.

Chapter 2

Background and Related Works

2.1 Predictive Maintenance Survey

The concept of predictive maintenance has been around for a long time, however, the recent surge in available data and machine learning algorithms has increased its popularity. The main works are covered across three popular modes of transport − road vehicles (automobiles), railways, and aircraft. Several approaches are used to conduct predictive maintenance in these modes of transport such as using mathematical models, threshold-based approaches, machine learning, deep learning, or a combination of these.

2.1.1 Related Surveys

The work in [35] conducts a survey on data-driven approaches for predictive maintenance specifically in the railway industry; the approaches mentioned are essentially machine learning and deep learning approaches for performing the predictions. Furthermore, the survey categorizes the proposed approaches based on the type of tasks, the frameworks used (or algorithms), the evaluation metrics, the datasets used, and so on. The analysis in the survey found that the increased loads on railway systems (due to increased production of cargo and transportation) cause damages throughout the structure while also being negatively impacted by certain environmental factors. The aim of the survey was to find trends among existing works such as aspects of the railways being subject to predictive maintenance methods, the type of data used for that purpose, the use and integration of deep learning models for the specific applications, and the solutions provided and how they are actually being used to perform the task of predictive maintenance within the railway industry.
The authors in [130] present a review that describes the use cases of predictive maintenance specifically in the automotive industry as well as discusses the challenges faced; they too primarily conduct a survey on machine learning-based approaches for performing predictive maintenance. Furthermore, the survey categorizes the proposed approaches based on the type of tasks and the frameworks used (or algorithms). The analysis of the works

surveyed in [130] gave the insights that the presence and availability of data for predictive maintenance-related tasks would benefit the advancement of the field, also that the approaches primarily followed a supervised learning approach, multiple data sources further improve the performance of the models, and that the use of deep learning methods includes a trade-off between performance and the lack of interpretation of the models and the lack of large volumes of labeled data. The aim of the conducted survey was to identify the most occurring maintenance cases, the most popular machine learning frameworks used in the approaches, and the most active authors.

The work in [140] presents a literature review of data-driven models as well for the predictive maintenance of railway tracks specifically. Furthermore, the literature review categorizes the proposed approaches based on the type of tasks and the frameworks used (or algorithms). The review found that the popular approaches for performing the task of predictive maintenance are deep learning (as was concluded in the other works), unsupervised learning techniques, and ensemble methods; the work also finds some of the top applications (or use cases) targeted in the works reviewed. The aim of conducting the literature review was to determine popular measurement methods (or data collection techniques) and the type of datasets used for the purpose of railway track engineering, the use and integration of models for the tasks of predictive maintenance, and the factors determining the selection of appropriate methods based on the type of data, type of issue, and so on.

The authors in [8] present a literature review of mathematical models and artificial intelligence techniques for predictive maintenance specifically in the automotive sector. The literature review summarizes the works reviewed in the paper and mentions their results and the challenges that they faced. Upon analyzing the several approaches in the work, the authors also conclude that the use of deep learning methods for the task of predictive maintenance comes with a trade-off between achieving high accuracies and requiring large volumes of data and being computationally expensive. Another common conclusion was the lack of available datasets for encouraging advancement in the field of predictive maintenance. Consequently, the evaluation of the proposed models for real-world deployment suffers due to the lack of real data and not synthetic data.

The work in [47] provides an analysis of the state of research advancements in prognostics for aircraft systems by focusing on prominent algorithms and their practicality and challenges. The work highlights the importance of prognostic and health management (PHM) in predicting the RUL to mitigate future breakdowns. The methodologies are grouped into three categories: a) physics-based modeling, b) data-driven techniques, and c) hybrid prognosis. The authors further discuss the the issues of lack of generalization, practicality, and scalability in most of the works, and also the lack of available datasets to allow researchers to validate their frameworks. Moreover, the authors conclude that although the incorporation of PHM in aircraft systems yields positive outcomes, research on the integration of hybrid PHM applications is lacking. Lastly, the authors emphasize on the future aspects and potential developments in hybrid prognostic approaches as having substantial scope.

The surveys related to the field of predictive maintenance mainly focus on only one mode of transport, moreover, the recent surveys have only reviewed approaches based on data-driven methods involving machine learning and deep learning algorithms. The survey presented in this thesis covers predictive maintenance algorithms and methodologies across three different modes of transport. Furthermore, the survey also includes various types of

frameworks that are not limited to machine learning and deep learning.

2.1.2 Predictive Maintenance in Railways

Mathematical Modeling-based approaches

The authors in [99] propose a predictive maintenance approach for railway systems specifically related to point systems. The work uses an Unobserved Components (UC) model in a state space framework. Their proposed system RCM2 is based on the combination of two maintenance techniques: Reliability Centered Maintenance (RCM1) [86] and Remote Condition Monitoring (RCM2) [79].
The framework presented in [99] contains an Unobserved Components (UC) model set-up in a State Space framework, this means that the model uses a reference curve to determine how close or how far the new incoming information is (component data). The approach uses data collected from manually introducing faults to the point mechanism of the railway; the main data used was the direction of movement of the point mechanism while the fault is characterized by the force vs. time curve. A threshold-based approach is used to determine if the condition of the point mechanism is good or bad depending on how far the input is from the reference curve. The threshold values are set from past experiences. The work presented in [99] is relatively older and does not seem feasible to be deployed in current situations due to how the railways have changed and the kind of data at hand but the idea behind the predictive maintenance approach is more than a rule-based approach which is widely used currently. Systems such as in [99] could be developed using much more complex systems although the idea behind the approach is simple.
The authors in [64] propose a complex fuzzy system-based thermography approach for conducting predictive maintenance in railways; their system is based on the concept of fuzzy logic which uses mathematical formulations that take in input variables and pass them through a set of rules to give out discrete outputs [145]. Their primary approach is to consider thermal changes caused by environmental conditions such as seasons, daylight, and the movement of trains. In their work, they capture thermal images of two components of the railway systems − the railway track and the pantograph catenary system. The authors state that the primary cause of failures in railway systems is due to the tension on the rail as the expansion of the rails inevitably impacts the wheels too. Furthermore, friction and overheating in the pantograph system caused by the electrical energy transmission from the catenary system could also result in defects which could cause substantial damage. In the proposed approach in [64], features are extracted from the thermal images of the contact point of the catenary wire and the joint points along the rail line. Furthermore, they also included seasonal conditions to account for the temperature at the contact points mentioned. The images captured were combined in a MATLAB environment (as the images were converted to signals) to be used for comparison with the results obtained from the complex fuzzy system. The complex fuzzy system was developed using data available about the medium such as annual temperature and annual daylight information. Also, a total of two hundred images were taken using an NEC F30W thermal camera from a real-world environment. The images were then processed using image and signal processing

techniques. The architecture proposed in [64] consists of two components − thermal image processing and estimations using a complex fuzzy system. The images are captured and processed and the distance between the rail lines is calculated; the fuzzy system is then used to determine the ideal distance between the lines given the environmental factors as the input; comparing the two obtained distances would then determine the health of the rail line. In essence, if the distance between the estimated distance and the imaged distance is too high, the track might deteriorate. Although the proposed method in [64] seems promising, the implementation in real-world scenarios might be extremely costly and not worth it; the study is based on environmental factors that cannot be changed and the results of the environmental factors seem obvious so installing thermal cameras and carrying out the required processing and computation could be futile. Furthermore, the images obtained would require further preprocessing and might be inaccurate due to the movement and vibrations of the train. Also, the authors base their entire model on thermography without any inclusion of other railway machine components that might require maintenance too.

Condition-based approaches

The authors in [27] propose an approach for conducting predictive maintenance for railway systems, specifically for railway tracks to monitor their degradation (specifically vertical deformation of the tracks). Their primary objective is to consider the stochastic nature of the real-world environments that potentially impact the conditions of railway lines that in turn cause service disruptions. As opposed to traditional long-term-based approaches such as scheduled maintenance, their approach is highly dynamic and relatively short-term as they allow for disruptions and unexpected faults to be taken into account dynamically into the model as parameters.
In essence, their focus is optimizing maintenance scheduling by using a predictive maintenance model while mainly focusing on the uncertainties (unexpected events) that follow maintenance planning. The first step is predicting the probability of track failure by evaluating degradation. However, they do so by adopting a "rolling-horizon approach" [27], as the name suggests, the model is designed to adapt to unexpected events (such as sudden significant degradation that could not be predicted beforehand). The "rolling-horizon approach" [27] would then determine a maintenance schedule by including the unexpected event in its calculations. The model is robust against unexpected events not only related to the railway track's condition but also unexpected events pertaining to the management. The model to evaluate degradation follows a risk-based (threshold-based) approach that is based on one of the railway management frameworks. In addition to the model, the "rolling-horizon" [27] framework allows for unexpected events from a previous time step to be added to any time step to make sure that previous schedules or fixes do not go unattended (dynamic inclusion of uncertainties). Furthermore, the framework also tries to increase the computational efficiency by reducing the number of railway lines monitored or evaluated at once; instead of evaluating all the railway lines at once, smaller subsets are taken thereby dividing the entire problem into sub-problems. Lastly, the authors in [27] use a mixed integer linear programming (MILP) model to perform the task of maintenance

scheduling. However, their proposed model is only relevant for frameworks containing a single railway line.
The authors in their work in [27] implement or evaluate their model on a single-line track in Sweden which was managed by Trafiverket (the Swedish Transport Administration). The authors claimed that their evaluation resulted in the model being able to adapt to certain uncertainties while delivering good performance. Although the proposed approach appears to be robust against uncertainties, it still does not take into consideration the availability of the tracks (free of trains) for scheduling maintenance which is a huge factor. Moreover, a practical approach would still require long-term planning and scheduling which the proposed approach does not consider. Lastly, the model's performance has been evaluated only on a single-track railway system which is far from being the real-world scenario.
The authors in [33] develop modules to facilitate the training of candidates in railcar learning factories. The modules contain several relevant components that are required for an individual to get acquainted with artificial intelligence tasks such as data acquisition, data preprocessing, data analytics, working with artificial neural networks, deployment of the model, and so on. The objective was to demonstrate the use of artificial intelligence for performing predictive maintenance by diagnosing wheel bearing conditions and predicting the remaining useful life (RUL) [20]. The work involved training a neural network on past data of the wheel bearing temperature using the Levenberg Marquardt algorithm [107] in a MATLAB 2018a environment. The data acquisition module focused on how collecting relevant data was based on its ability to determine the states of the components interested in, the preprocessing module focused on filtering raw data to remove noise, the model training module focused on the implementation of machine learning models by using the relevant features as inputs, and finally, the deployment module focused on how to integrate the models into existing systems to perform predictive maintenance in real-world environments. The work in [33] followed a rule-based approach for the prediction of the Remaining Useful Life (RUL) [20] by deciding temperature ranges that could negatively impact the wheel bearings and the severity of the impact, and also the performance of the lubricants given the temperature ranges. Furthermore, an artificial neural network was trained based on dynamic time series with the goal of predicting the temperature variations of the wheel bearings. The neural network uses the Levenberg-Marquardt algorithm (also known as the damped least-squares (DLS) algorithm) so a non-linear time series problem can be solved; the model uses the Mean Squared Error (MSE) as the metric.
The developed training modules in [33] might prove to be useful to get beginners acquainted with tackling a machine learning task but the demonstration of the modules does not propose any significant novelty to the field of existing predictive maintenance approaches. The paper does not even elaborate on their deployment module; providing an architecture of how the neural network could be deployed while collecting wheel bearing temperatures and could have been more supportive of their work.

Machine Learning-based approaches

The work in [73] proposes a machine learning approach using large volumes of data to improve the rail network velocity by conducting predictive maintenance. The authors use

historical detector data, failure records, maintenance records, train type data, and weather data to perform various analytics. The primary aim of the work is to use machine learning approaches to make accurate predictions and obtain a set of human interpretable rules that can be used by personnel to perform predictive maintenance. The large volume of detector data is collected by IBM along with the US Class I railroad by installing detectors along the rail tracks that log various mechanical observations (such as temperature, geometry, etc) when a railway train passes by. The detector data is then merged with other records such as failures, maintenance, and so on.

The work in [73] designs experiments to make predictions such as alarm predictions in which extreme failures caused by hot bearings are predicted seven days in advance, bad truck prediction in which the truck's deteriorated performance is detected by analyzing wheel patterns, bad wheel prediction in which wheel defects are detected using the wheel patterns, and asymmetric wheel wearing detection in which a specific issue of the wheels is detected. For the alarm prediction in [73], the first step was feature extraction in which historical detector records were aggregated and statistical approaches were used to extract features; hashing and parallelization were used to accelerate this process due to the large volume of data. The second step was dimensionality reduction which was done by using Principal Component Analysis (PCA). Finally, a variant of SVM was trained using the processed data with the aim of predicting if a severe failure will occur after a few days. The evaluation metrics for this task were true positivity rate and false positivity rate. Lastly, there was the task of obtaining a set of rules which was done by making predictions on the entire possible feature space (containing all possible values the input variables could take) and then establishing ranges by the linearization of the grids.

The second task of failure prediction in [73] combines two other prediction tasks into one − bad truck and bad wheel prediction. The modeling process of this and the previous task is similar but has some significant differences. The data used for this task was collected by extracting detector readings of the wheel/truck for a certain time window before an alarm was raised (followed by a repair). Doing so gave the authors the detector readings that would lead to a wheel or truck failure. A simple decision tree was used to predict if the wheels or trucks would suffer failures in the next three months; the reason for doing so was to obtain a set of rules; as every leaf of the decision tree would provide a basis for a logical rule.

The work in [73] provides an approach that involves training and testing on a large volume of data and achieving good results. However, the metric used for the first task could have been an F1-score with the targeted class as predicting failure. Also, for the second task, the authors do not mention extracting features from post-repair time windows to obtain good detector readings to make it a binary classification. The methods provided do not seem robust against concept drift as they make huge use of detectors on the railway side. Lastly, the approaches provided are generic to all trains passing on a track (irrespective of the type) and the models are not train-specific (not onboard).

The authors in [19] present the use of simple machine learning techniques for conducting predictive maintenance in railways, specifically for railway switches. Their work focuses mainly on employing tree-based machine learning algorithms to make various predictions. The authors believe that most railway systems are unable to fund the process of installing additional equipment for collecting data and performing prognosis (predictive

maintenance), so in their work, they use data collected from a sort of ticket system that is used to generate maintenance requests. The system used for collecting the data was used by a railway business.
The data collected from the maintenance report system results from four data sources: an asset register which contains details about the switches such as the type, location, and so on, the condition data which contains information about the most recent state of the switches which is decided by visual inspection, the notifications file which contains information about maintenance requests for the switches describing the details about the issues if maintenance is needed or nothing if no maintenance action is required, and the work-orders file which contains records about the maintenance actions that could be due to scheduled inspections, unexpected events, or excess funding.
The work in [19] performs basic data preprocessing steps such as cleaning the data, data type conversions, and using embedding techniques to handle string values. The authors then analyze the data and find the most caused issues, the switch components experiencing the most issues, and they selected the maintenance activities that contained the most amount of samples and data. The feature extraction phase involved the elimination of redundant features as well as uncorrelated features. Other popular data processing techniques such as scaling or normalization were not performed because the authors believed that such techniques do not impact the training of tree-based algorithms. The approach in [19] uses tree-based models to make predictions − supervised learning algorithms − because the data used in the work is labeled. The aforementioned notifications dataset was labeled based on the information it contained − if a sample contained work-order information, it means that the inspection resulted in maintenance, or else it meant that no maintenance was needed. This dataset resulted in being imbalanced so the authors used downsampling to even out the distribution of the dataset. Although the authors believed that the dataset was highly imbalanced, it had more samples (>twenty percent) in the minority class than in the idea of a high imbalance problem (<ten percent). The models used in the work are Decision Tree, Random Forest, and Gradient Boosted Tree. The Decision Tree model is used to serve as a baseline for their work. Various evaluation metrics were used for the models such as the accuracy and the F1-score.
The work in [19] proposes a simple and practical solution to the predictive maintenance problem of railway switches that does not require additional costly installation of devices. However, their work lacks exploration and novelty, the work did not try oversampling techniques or a hybrid of over and undersampling, the authors did not use existing techniques for proper hyperparameter tuning, and so on. A lot of improvements must be made to their approach for it to be feasible for deployment to a real-world scenario.

Deep Learning-based approaches

The authors in [48] propose an approach to predict rail and geometry defects and determine required inspections by integrating predictions with maintenance scheduling activities for railways. The work also proposes methods to avoid false negatives as the authors believe that it is much more costly to not detect a defect than to raise a false alarm. The authors

claim that the proposed architecture in [48] is robust against changing conditions. The architecture uses clustering methods, tree-based algorithms, recurrent neural networks, and reinforcement learning approaches. The data used in this work is rail data gathered over two years and contains three different datasets: a defect dataset that contains rail and geometry defects (and the type/category) found in segments of a railway network, an inspection dataset that includes the type of inspections conducted, the segments of the network inspected and so on, and finally a load information dataset that includes the daily load (total gross tonnage) endured by the segments of the railway network. The authors also matched the defects to the inspections conducted to better understand the distribution of the data.

The work in [48] suggests the clustering of railway segments that have common characteristics such as loads endured, seasonal conditions, and so on because the authors believe that a generalized approach may fail to determine the specific conditions resulting in defects for segments experiencing different conditions. They adopt the K-Means algorithm [54] to determine the optimal number of clusters such that each cluster has enough data to train the models, and also to perform feature selection. The machine learning models used in [48] for defect predictions are Random Forests (RF) and Recurrent Neural Networks (RNN), however, the architectures chosen are not very deep to avoid overfitting and to reduce computation complexity. The Mean Absolute Error (MAE) and Mean Squared Error (MSE) are used as metrics to maximize the accuracy of the models. Lastly, Multinomial Ordinal Regression is used to fit the outputs of the models to the relevant defect category. The performance of the models seems to be fair, however, the models have a tendency to mislabel defects and undershoot (increasing false negatives) due to the class imbalance in the dataset; as mentioned earlier, the authors believe that false negatives may prove to be way more costly than false positives so they propose a hybrid version by developing a loss function using Particle Swarm Optimization (PSO) [65] that combines the two machine learning models used; doing so did decrease the average undershooting of the framework.

In addition to the defect prediction, the work in [48] also provides an approach for maintenance scheduling that aims at taking into account the stochastic nature of defect occurrences using reinforcement learning techniques. The authors use a Markov Decision Process (MDP) [103] model to determine an optimal scheduling policy that minimizes the incurred costs given a set of states and actions. The results of the MDP model suggest that following a threshold-based approach is best for scheduling maintenance or inspections. Lastly, the authors introduce Restless Bandits [138] to make the scheduling policy more practical by introducing the factor of inspection crew limitation. The Restless Bandits mathematical model allows for dynamic updates depending on the situation and optimally allocates the resources with the aim of minimizing the cost.

The work in [48] appears to be extensive and robust while using easy-to-obtain data, however, the scheduling policies do not take into consideration the availability of inspection crews based on geographic factors. The entire workflow of the work might seem a little complex and difficult to implement for use in a real-world scenario.

2.1.3 Predictive Maintenance in Automobiles

Mathematical Modeling-based approaches

The work in [72] proposes an approach using Multi-Target Probability Estimation (M-IFN [16]) algorithms to conduct predictive maintenance in cars. The data used in this work contains sensor measurements and warranty claims (synthetic data and not real-world data) that are used to predict the probability and the time of failure (with the goal of notifying the user one or two weeks prior to the failure of a system) of a particular subsystem of cars. The proposed framework is compared to other models such as the single-target probability estimation (InfoFuzzy Networks [16]) algorithm and a reliability analysis model by Weibull.
The proposed work in [72] offers a novel mathematical approach for conducting predictive maintenance, however, the evaluation and case study are conducted on synthetic data which could not certainly determine if the approach is ready for real-world situations; the requirement of real data for validation purposes is vital to correctly evaluate the performance of models.

Condition-based approaches

The authors in [122] propose an IoT-based predictive maintenance approach for an automobile fleet system. Their approach primarily consists of retrieving data via sensors embedded in vehicles and sending them to the cloud via IoT Gateways. The authors proposed a communication framework for vehicles to communicate with the cloud.
The work in [122] uses the Message Queue Telemetry Transport (MQTT) protocol − a flexible publish/subscribe-based messaging protocol − to enable communication between the vehicles and the server. The reason for doing so is that the MQTT protocol is lightweight and the IoT Edge does not have high computational power, limited storage, and due to being embedded in the vehicle, may not have strong connectivity. Within this messaging protocol, the vehicles of the fleet act as the clients and transmit sensor data to the cloud which acts as the broker. In order to implement the MQTT Protocol, libraries such as Eclipse Mosquitto [41] act as the MQTT Broker, and Eclipse Paho [42] provides libraries for MQTT Clients.
The authors further propose a framework for the entire implementation which follows a threshold-based approach to determine faulty components of the vehicles, however, they do not provide any description or evidence of how they came up with the thresholds. Moreover, they do not even list the thresholds or explain if they are based on individual vehicles or established via fleet-wide analysis.
Although the proposed approach in [122] seems viable, the work does not contain any relevant implementation of their framework - neither in real life nor in a simulation - the authors just hypothesize the performance of their model and perform some calculations to determine several percentages in savings if their approach is used.
The authors in [122] display a sample implementation of their proposed architecture via simulating a single connected vehicle but do not provide any results or evaluations based

on that, they do not even elaborate on the kind of data used to simulate the behavior of the car, that is, if they used any past data or used synthetic data.
The authors in [66] propose an unsupervised approach for dynamically selecting sensors to conduct predictive maintenance on a fleet of buses. The proposed work aims at presenting an improvement to an existing solution − the consensus self-organized models (COSMO) [20] approach − which involves finding deviations in signals or datastreams in a dynamic environment. The authors also propose an IoT architecture that contains three nodes − the vehicle node (representing the vehicle), the server leader node (representing a region containing several vehicles), and the root node (represents the entire fleet) − to enable data collection, storage and processing, and fleet administration. The authors gathered sensor data from a single public transport bus and simulated the behaviors of several buses using synthetic data characterized by faults and repairs.
The proposed IoT architecture proposed in [66] contains three layers − the perception layer (which contains the vehicle node) which provides an interface to the edge (sensors, networks, embedded systems) and performs lightweight storage and machine learning tasks. It connects to the second layer − the middleware layer (which contains the server leader nodes) − using the MQTT protocol for the same reasons as the ones mentioned in work [122]. The nodes in the middleware layer perform heavier tasks compared to the previous layer. The middleware layer in addition also has an interface to the uppermost layer − the application layer (which contains the root node) − using HTTPS. The application layer provides the main interface to the entire fleet via the root node.
The approach in [66] shares similar phases with the original COSMO [20] method which include sensor selection based on interestingness. Non-random and stable signals are considered as interesting signals [66]. Another similar phase would be the deviation detection phase which involves comparing a test bus to the entire fleet using distance metrics to compare the deviation of the test bus sensor data. However, the proposed method − improved consensus self-organized models (ICOSMO) − periodically reorganizes the selected sensors as opposed to its predecessor and so it does not require going through the sensor selection process again. ICOSMO reorganizes the sensors based on their contributions and their potential contributions in finding deviations.
The work proposed in [66] seems feasible as it utilizes data collected from a real-world scenario and it also proposes an architecture that could be possible to implement. However, the authors mention that the proposed model is not immune to concept drift so testing the model across several climatic, terrain, and traffic conditions might yield different results. Furthermore, the model was evaluated for a fleet using data from only a single bus; evaluation of the model on real rather than synthetic data might also yield different results.

Machine Learning-based approaches

The authors in [49] propose a data-driven pipeline called PREPIPE for conducting predictive maintenance in automobiles. The work aims at predicting the clogging status of the oxygen sensor (also known as lambda sensor) which is placed in the exhaust system of vehicles. The purpose behind the prediction is to control the combustion efficiency of the engine control unit and to control pollutant emissions. The data used in the work is

collected by using a diesel engine equipped with standard sensors as well as some additional sensors. The engine is then exercised via different loads and gas pedal presses to emulate different driving conditions. The data collection process uses two approaches − one collects data for an entire time-cycle at a lower frequency while the other only collects data for the last few minutes but at a higher frequency − however, the latter is only used to perform the prediction tasks. The collected data is then labeled with the help of domain experts based on the status of the oxygen sensor. The labeling of the time cycles is done using certain thresholds established and the labels define the severity of clogging of the oxygen sensor.
The proposed pipeline in [49] contains several steps starting with signal selection in which various supervised techniques (as they are able to determine the ability of all features to make predictions but this approach is time-consuming) and unsupervised techniques (that use statistical methods such as correlation and similarity) are tested to determine the best subset of signals that are able to well-define the status of the oxygen sensor. The next step is the windowing step in which the correct time window size of the signals is determined which is able to correctly evaluate the status of the oxygen sensor. Then the feature extraction phase converts the signals into useful features that represent the characteristics of the time series and can be used as inputs to the machine learning models. After extracting the features, the feature selection phase using supervised learning methods determines which of them should be used and which of them are not related to the target status. Because the current features only represent the present status of the sensor, a historicization phase determines if adding features from past cycles could improve the performance of the models, this is done by using feature selection again. Lastly, models such as decision trees, random forest, support vector machines, and neural networks are trained and tested using the cleaned and processed data. The metrics used to evaluate the performance of the models are F1-score and accuracy.
The results suggested that the feature selection and the signal selection steps greatly improve the score for the target class which is predicting when the oxygen sensor is severely clogged. However, the results table provided in the paper displays that historicization performs better than the signal selection step. The authors then compare their proposed pipeline with some deep learning approaches such as convolution neural networks and LSTMs and conclude that their model performs better but that might be largely due to the limited size of their dataset as deep learning methods (such as RNNs) would require larger time series to extract the patterns.
The work in [49] yields some interesting results however, some of the results mentioned contradict the results displayed in the paper. Furthermore, their comparison approaches do not seem fair given the size of data fed to models that require a large amount of data. Moreover, the approach proposed is not robust against concept drift and would have to manually be configured.

Deep Learning-based approaches

The work in [26] proposes a deep learning-based approach for conducting predictive maintenance in a fleet of automobiles by using Geographic Information Systems (GIS) [24] data.

GIS is a tool that can be used to retrieve geographical information such as weather and terrain data for a given location. They use time-between-failures (TBF), which is the time (in days) for an automobile between two consecutive breakdowns, as the target output for their neural network.
The work in [26] uses data from two sources − historical maintenance data containing failure information from a sizable fleet system and GIS data, specifically climatic features: temperature and rain. The authors used the vehicles' longitude and latitude to gather the mentioned GIS data and added it to the dataset, they also believed that additional GIS data such as traffic and terrain would be difficult to gather and its correctness could not be relied on compared to historical weather data which is easier to gather from various public sources.
The network architecture proposed in [26] consists of a deep learning model. Firstly, the nominal features from the historical vehicle data were one-hot encoded but this resulted in the dimensionality becoming too high and making the dataset sparse, so the one-hot encoded data was passed through an autoencoder to reduce its dimensionality. The historical data was then combined with the GIS data and used to train the deep neural network. Lastly, the weights of the input layer were extracted to determine the importance of all features. The authors in [26] found out that incorporating GIS data along with vehicle maintenance data did benefit the overall performance of the neural network. Moreover, they found out that a few climatic features were way more relevant than some vehicle features.
Although the work in [26] displayed improved and promising results, it only uses time-between-failures as the output which could be used to tell us approximately when a car would break down. The work does not contain any functionality for failure detection, the evaluation metric used in the work is Root Mean Squared Error (RMSE) and the best results have an RMSE of more than a year so adopting this approach in a real-world scenario does not seem feasible. Furthermore, incorporating the GIS data along with the historical data only improved the score by approximately one day which again needs to be weighed against the cost of retrieving such data and using it to train a model that is more complex.

2.1.4 Predictive Maintenance in Aircraft

Mathematical Modeling-based approaches

The authors in [131] propose an approach that uses proportional hazard models (PHM) [71] for the reliability estimation of components of aircraft by identifying operational factors that impact those components. Two types of Proportional Hazard Models are implemented: time-dependent and time-independent which use the operational attributes as covariates. The work follows a data-driven approach and uses historical maintenance and operational data. The data is received from a fleet of aircraft operating in the Asia-Pacific region.
The method proposed in [131] contains identifying data related to components from the maintenance dataset and the corresponding data of the components are extracted from the flight information dataset. The components are then categorized in two ways: un-

expected failure requiring unscheduled maintenance or maintenance as part of scheduled maintenance. The next step is to identify the flights that could have caused the unexpected failure of a particular component. This identification is done based on the time at which the maintenance occurred and by retrieving the flights prior to the maintenance based on a time frame. The data is then analyzed using two approaches: Extreme Value Analysis (EVA) in which the flights chosen for a particular component failure are narrowed down to a single flight that has the most abnormal operational feature values, and the Maximum Difference Analysis (MDA) in which the operational factors are chosen based on their probability of causing unexpected failures of a particular component.
The previously mentioned steps were important for preprocessing and analyzing the data at hand. The next step involves the modeling for reliability estimation; firstly a Generalized Renewal Process model is used as a baseline that does not include any operational factors and only the times of the two categories. The time-dependent proportional hazard model is used by introducing covariates (to a Generalized Renewal Process model) that are the means of the operational factors' values for one flight. The time-independent proportional hazard model is used by introducing covariates that are operational factors' values that vary over time for all flights related to the component failure.
The metrics used for the evaluation of the reliability models are Maximum Likelihood Estimation (MLE) and the goodness-of-fit. The evaluation results suggested that the time-dependent proportional hazards model had better results, especially for a larger number of operational factors but that resulted in heavy computation times. An additional use other than scheduling maintenance based on the reliability estimates would be to conversely predict the values of the covariates and use them to determine failure probabilities for a future time.
The work in [131] proposes a unique approach for failure predictions, however, their experimentation involved training and testing on the same dataset without any split causing data leakage (due to limited data), as such, further tests would have to be conducted to properly evaluate their work. Secondly, the number of operational factors chosen needs to be better justified for reliability estimation. Also, the impact of one component on the failure of another should be investigated. The estimations with good scores are computationally expensive and could not be used in a real-time situation, also the overall accuracy of the model needs to be tested due to the reasons mentioned earlier regarding the data, so the deployment of this method in a real-world scenario does not seem feasible.
The work in [132] proposes a method for conducting predictive maintenance of redundant aircraft systems that have various degradation trends and wear values. The aim of this work is to optimize the maintenance planning of the systems based on their wear profiles and Remaining Useful Life (RUL) estimate while minimizing the costs and considering other operational aspects of an airline that influence the maintenance process. In addition to the optimization problem, the work also presents methods of estimating the future degradation trends of components based on an assumption of their wear profiles. The data used in this work is field data from hydraulic systems.
The work in [132] involves a method to identify a wear model (from a multiple model method) for a component that is degrading non-linearly or without a specific interval time frame by the use of an Extended Kalman Filter. Another step includes using a multiple-model approach to analyze systems that contain multiple wear profiles of aircraft

components, this is to avoid the problem of selecting a single model that would accurately represent the wear pattern of a component. Then the optimization of the maintenance planning of the degrading systems is done by considering the operational factors of an airline (limitations) and also minimizing the costs. Lastly, a combination of the methods is used to conduct all the predictions and predict future operational costs for finding the optimal maintenance schedules.
The optimization problem in [132] specifically pertains to line maintenance of aeronautical systems. Line maintenance is a situation resulting in unscheduled maintenance caused due to unexpected events or scheduled maintenance not requiring any special training, equipment, or facilities. The problem is solved by focusing on the redundancy of components and their extent of degradation as well the future estimates of their degradation to calculate the future operational costs while taking into account certain constraints or limitations that might cause disruptions.
The methods proposed in [132] take into consideration a lot of factors and might be able to provide an optimal solution for maintenance planning however, their proposed algorithm involves an exhaustive search method, which might be computationally very costly and even infeasible due to the stochastic nature of the maintenance process as pointed out by the other works.

Machine Learning-based approaches

The authors in [43] present an approach to use Particle Swarm Optimization (PSO) [103] for model selection and feature selection to perform predictive maintenance by predicting the Remaining Useful Life of aircraft engines. The authors used support vector machines for the task, more specifically a variant of support vector machines − ϵ-SVR (Support Vector Regression) [40]. The reason for choosing this variant is that it allows for minimizing the generalized error as opposed to the standard support vector regression machine which aims at minimizing the training error. The primary purpose of this work is to propose an approach that involves the simultaneous tackling of two important machine learning tasks − feature selection and hyperparameter tuning − using Particle Swarm Optimization (PSO) and they used the task of Remaining Useful Life (RUL) prediction to support and evaluate their work. The dataset used in this work was provided by NASA in [46] and contains the remaining flight duration for electric aircraft.
The approach in [43] performs basic preprocessing − just splitting the dataset into a training test and a test set. The training phase of the approach relies majorly on Particle Swarm Optimization (PSO) as it is used for feature selection (to eliminate redundancy and irrelevant inputs). Lastly, the testing phase is also simple as it just involves testing the performance of the support vector machine model on the test set. The metrics used for the evaluation of their model are the Mean Absolute Percentage Error (MAPE) and the Mean Absolute Error (MAE).
The authors in [43] believed that the features would have to be converted to a binary vector of the length of the number of features for PSO to be used for feature selection. Consequently, the binary version of Particle Swarm Optimization is used for selecting relevant features. The model selection (finding the optimal hyperparameters for the support

vector regression model) was conducted in parallel by using the continuous Particle Swarm Optimization variant. The Particle Swarm Optimization Algorithms are run iteratively, separately, and simultaneously until the optimal results are found which are then fed to the SVM model to be trained and tested.
The work in [43] primarily focuses on the use of Particle Swarm Optimization and uses predictive maintenance incorrectly. The work does not use a relevant dataset for a maintenance problem but for a battery consumption problem. Furthermore, the authors claim their problem was a big data problem while they used a small number of samples with a small number of features. Furthermore, the authors just used one machine learning algorithm - a variant of the Support Vector Regression Machine - to support their entire work without using any other models for comparison.
The work in [69] proposes a regression solution for performing predictive maintenance by using only the event logs from post-flight reports. The flight reports consist of event logs that contain information about equipment failure along with some other description such as the system or subsystem that may have caused it. The aim of the regression model is to predict the next occurrence of an event (event of interest or target event). The regression model does so by determining the risk that the event might occur given the past set of events. The proposed method was evaluated on data received from a fleet of aircraft and the target prediction made was the failure of components related to the landing gear.
As mentioned earlier, the event logs consist of a timestamp, a unique identifier, a description of the failure containing the systems, and so on. The proposed work in [69] only requires the timestamp and the event id (unique identifier). The events from the log are preprocessed in a few steps starting with removing events that occur rarely, repeated events within a single time segment are removed so a single time segment will have unique events (doing so reduces noise), events that occur frequently are penalized as they might be trivial and might add noise (the authors suggest a token embedding approach to do so), events occurring in consecutive time segments are removed and only the first event is used as they might be trivial and might add noise, and lastly, the feature selection phase.
The work in [69] uses a Multiple Instance Learning (MIL) [78] approach as the authors believe that the target event is related to a bunch of instances and not just a single instance. In other words, the aim of the regression model is to predict the chance an event occurs based on its preceding events (a bunch and not all) and not just one related event. Another concern that the authors had was the imbalance in the event occurrences (catastrophic events occur rarely) so to address this problem and the problem of having unrelated preceding events, a sliding window approach is adopted to perform oversampling of the relevant intervals.
A random forest model is used to implement the method proposed in [69] by training and testing it. For comparison, a binary classifier was trained that took in a bunch of instances as input (based on the Multiple Instance Learning (MIL) approach) and predicted if the target event will occur. The deployment of the proposed model would include inputting the time series data and the output would be the risk of the event occurring for each time step, deciding whether the target event would occur or not was based on an established threshold.
The work in [69] claims to be the first approach that uses only the post-flight reports to develop a model for performing predictive maintenance but their approach does not use

any features relevant to causing the target failure such as the components of the aircraft, maintenance types, and other flight-related information. Furthermore, the scores achieved by their model are low and so deploying their approach in a real-world scenario does not seem practical.
The authors in [13] propose an analysis system that uses clustering, an unsupervised learning technique, to perform predictive maintenance in aircraft, specifically in aero-engines. The work uses the Train_FD001 dataset that contains simulated degradation data of aero-engines and uses a subset from the C-MAPSS dataset to evaluate the performance of the trained model. The proposed work in [13] reports that the proposed system was able to lower the costs associated with maintenance activities and also increase the aircraft's uptime. The aim of the proposed work is to predict the state of the aero-engines and the decision on whether to conduct maintenance or not is carried out.
The architecture in [13] uses Principle Component Analysis (PCA) to reduce the dataset's dimensionality and perform feature extraction. The work uses the K-Means Clustering algorithm to perform the unsupervised learning task. The clustering approach works in line with an anomaly detection approach in which the datapoints that are away from the dense regions are marked as abnormal; this follows a step that determines the transition of a datapoint from the dense region to the outlier region thereby proposing a predictive maintenance approach.
The approach is similar to the one presented in this work, however, [13] primarily follows the idea of anomaly detection by finding points that are away from the dense region, as opposed to detecting specific potential faults in individual components; some aero-engines would have to be on the verge of failure for them to be marked as an anomaly in a large fleet of aircraft. Additionally, using PCA to reduce the size of the dataset (dimensionality reduction) may also result in the loss of key trends and features present in the data over time. Hence, reducing the dimensions of sensor data may not be the ideal move when considering complex temporal trends. Furthermore, they only use Euclidean distance as the metric for clustering, which may not give accurate results for data with noise and would be unable to capture and interpret complex trends. Consequently, many normal engines were clustered as "alarming", whereas, only 4 from the dataset actually needed maintenance; hence, resulting in a significant number of potential false positives. This implies that their model overgeneralizes and misclassifies based on density and not actual abnormalities and gradual, underlying faults.

Deep Learning-based approaches

The authors in [141] propose an architecture to conduct predictive maintenance in aircraft by using convolutional transformers. In addition to proposing a model for the task, their work also introduces a multivariate time series dataset of many labeled flights consisting of sensor data called the National General Aviation Flight Information Database − Maintenance Classification (NGAFID-MC) dataset.
The presented dataset in [141] was created using five years of textual maintenance records retrieved from the NGAFID. The records were clustered based on the type of maintenance issue which were then validated by domain experts. The individual flights were then ex-

tracted and labeled as 'before' or 'after' depending on whether the flights occurred before or after a maintenance action using a maintenance record log. The work in [141] focused on two maintenance issues as they contained the majority of the records while the others contained very small numbers of records. For individual maintenance issues, five flights before the date of maintenance action were extracted as bad flights (implying that those flights lead to defects) and five flights after the maintenance action date were marked as good flights (implying that the flights were in good condition, free of defects).
The authors in [141] believed that a deep learning approach should be the way to tackle the problem at hand as feature selection (using feature importance analysis) would not be practical due to stochastic factors and due to the kind of data. Also, the data used is time series containing sequences of a large number of time steps so a deep learning framework would be viable. Furthermore, the authors chose simple augmentation methods for their data to avoid complexity as advanced methods would require the training of additional generative models. The work used image augmentation techniques as opposed to traditional time series augmentation techniques as the authors felt that those were not suitable for the kind of data they used. The image augmentation techniques used were cutmix, cutout, and mixup.
The architecture of the network proposed in the work [141] is called Convolutional Multi-Headed Self Attention (Conv-MHSA). The architecture takes in the input data and passes it through a series of 1D Convolutional layers to extract sequence embeddings from the inputs while also reducing the resolution of the input to lower complexity. The sequence embeddings are then passed through stacked MHSA encoder layers after which the output is pooled and passed through a dense layer to get a prediction. The authors also used other architectures such as Convolutional Long Short-Term Memory Networks and Convolutional Gated Recurrent Unit (GRU) Variational Auto Encoders for comparison with their proposed architecture.
The presented dataset in [141] serves as a benchmark in multivariate time series classification and could potentially help in enabling further studies in the domain, however, the dataset could contain mislabeled datapoints due to the stochastic nature of maintenance. Secondly, the proposed architecture seems to perform better than the other models employed for comparison in terms of computation complexity as well as classification performance but the model only takes in the last few thousand time steps per flight as inputs; a more deployable approach would require being able to take at least a full flight's data at once to find patterns and make predictions. Also, the current model does not predict the kind of maintenance issue so requiring a full check-up might be expensive.

2.1.5 Summary

The summaries of the mentioned works have been provided in the tables in this subsection. The works have been grouped and categorized by the main methods used in their frameworks, the categories include mathematical models such as statistical approaches, threshold-based models that follow a threshold-based approach for estimations, deep learning models, and machine learning models such as ensembles, tree-based, and SVM models that are not deep neural network frameworks. The summaries include the type of methods

used, the task for which that type of method was used, the dataset (type of data) used for the task, and the work that proposed the method. Table 2.1 provides a summary of works related to automobiles, Table 2.2 provides a summary of works related to railways, and Table 2.3 provides a summary of works related to aircraft.

Method	**Work**	**Task**	**Dataset**
Mathematical Models	[66]	Dynamic sensor selection	Sensor data and synthetic data
	[72]	Probability and time of failure prediction	Sensor measurement and warranty record data
Condition-based	[122]	IoT-based predictive maintenance	(Unknown)
	[66]	Deviation detection	Sensor data and synthetic data
	[20]	Deviation detection	Fleet sensor data
Deep Learning	[26]	TBF prediction	Historical maintenance data and GIS data
	[49]	Oxygen sensor clogging prediction	Diesel engine sensor data
Machine Learning (Non-Deep Learning)	[49]	Feature selection and oxygen sensor clogging prediction	Diesel engine sensor data

Table 2.1: Summary of predictive maintenance approaches for automobiles

Method	Work	Task	Dataset
Mathematical Models	[27]	Dynamic inclusion of uncertainties and maintenance scheduling	Single-line track data
	[48]	Fitting outputs to relevant classes and hybridization	Defect, inspection, and load information data
	[99]	Point systems fault analysis	Experimentally collected data
Condition-based	[27]	Degradation evaluation	Single-line track data
	[33]	RUL prediction	Wheel bearing data
Deep Learning	[33]	Demonstration of training modules	Wheel bearing data
	[48]	Rail defect prediction and maintenance scheduling	Defect, inspection, and load information data
Machine Learning (Non-Deep Learning)	[64]	Estimation of track and pantograph system deterioration	Thermal images
	[48]	Rail defect prediction and clustering	Defect, inspection, and load information data
	[73]	Extreme failure and bad component prediction, obtaining rule sets	Detector data, failure and maintenance records
	[19]	Railway switch issue predictions	Maintenance request system data

Table 2.2: Summary of predictive maintenance approaches for railways

Method	Work	Task	Dataset
Mathematical Models	[69]	Oversampling and bagging instances for training	Post-flight reports containing maintenance event logs
	[131]	Identification of operational factors impacting aircraft components and reliability estimation	Maintenance and operational data
	[132]	Component wear profile analysis and degradation estimation, maintenance scheduling	Hydraulic systems field data
Deep Learning	[141]	Maintenance issue prediction	Multivariate time series sensor data
Machine Learning (Non-Deep Learning)	[43]	Model and feature selection, RUL prediction	Electric aircraft data by NASA
	[69]	Target event prediction	Post-flight reports containing maintenance event logs
	[13]	Aero-engine degradation prediction	Synthetic degradation data and test data

Table 2.3: Summary of predictive maintenance approaches for aircraft

2.2 Synthetic Data Generation

Synthetic data is data that is not recorded by actions or occurrences in the real world but is generated manually and artificially for specific tasks [60]. Some of the main reasons for generating synthetic data are to simulate real-world scenarios and to test algorithms and models for real-world scenarios [92]. Another important use of generating synthetic data is for privacy concerns − in situations including sensitive data (such as medical/legal records), this data may not be publicly available due to privacy concerns of the users, however, generating synthetic data based on the real data can help in extracting the useful information and trends, and in replicating the same in artificially created data while maintaining user confidentiality [3, 12, 104]. Moreover, in situations where sufficient real data is not available for the purpose of training and testing machine learning models, the generation of synthetic data based on the real data can help in providing the data required for the task [89, 92]. Generating synthetic data is cheaper and quicker in such cases when compared to developing systems for further data collection and waiting for the collection process to gather sufficient data.
The generation and use of synthetic data depend on the situation [39]. In general, there are two types of synthetic data −

- Fully synthetic data - in this case, the synthetic data is not generated by using any real-world data. This type of synthetic data is generated when only the variables of the data are known but there is no actual data to extract information from.

- Partially synthetic data - the synthetic data is generated by extracting information from real-world data. This type of synthetic data retains underlying features and information from real-world data and is able to imitate it while discarding sensitive information (such as names, identifiers, etc.).

There are several techniques to generate synthetic data that are based on the type of data to be generated, the complexity of the technique, and the uses of the synthetic data. Synthetic data generated can be of various types such as tabular data, images, text, time series, videos, and so on. The following subsections describe some popular synthetic data generation techniques.

2.2.1 Using a statistical distribution

Using a statistical distribution to generate synthetic data involves analyzing the real-world data and its underlying distribution. Computations and algorithms are created to perform statistical tests and analysis on real-world data to generate synthetic data that contains a distribution close to the real-world data [77, 100]. The statistical tests and analysis of the real-world data yield the underlying distribution of the data or its best fit to an existing probability distribution type such as the normal distribution, exponential distribution, uniform distribution, and more. After identifying the closest distribution to the real-world data and its statistical parameters to establish the data's domain, value range, and other statistical attributes, random samples can be drawn from the distribution to generate synthetic data. Doing so results in synthetic samples being chosen from a distribution very similar to that of the real-world data. Thus, the resultant synthetic dataset will have similar statistical attributes and values to replicate real-world scenarios while maintaining differences from real data [44].
In the absence of any real-world data, this method can still be used if there is an understanding of the data variables and their respective distribution parameters. For example, to generate synthetic ages of drivers in a city, it is justified to assume that the age of the drivers will be above 18 and have a mean of approximately 35 [121]. This knowledge can be used to calculate distribution parameters and generate synthetic data from a known statistical distribution without requiring real-world data. Using a distribution for generating synthetic data is a simple approach, however, the method is unable to generate good quality synthetic data in complex situations demanding complex data types. Complex data types such as images, videos, speech, and so on cannot be characterized only based on the underlying distribution of their values; in such situations, using more advanced methods to generate synthetic data becomes necessary.

2.2.2 Agent-Based Modeling

Agent-based modeling (ABM) can be used to generate synthetic data by creating a simulation in which individual entities (or agents) interact with each other in an environment [59, 75]. ABM is useful for generating synthetic data in situations requiring detailed

and complex interactions and behavior in a system. The main characteristics of ABM are [17, 56]:

- Agent – The agents defined are autonomous and have specific characteristics and constraints associated with them that determine their way of interacting with other agents.
- Environment – The environment serves as a space that contains the agents and enables them to interact in a predefined setting by providing context for those interactions.
- Interaction – The interactions mentioned earlier are essentially actions made by the agents that result in outcomes affecting other agents.
- Emergence – Emergent phenomena are the results or outputs of the interactions between the individual agents that were not programmed or explicitly entered into the setting but arose only through the interactions of the agents.

Using ABM to run simulations and to collect/record data from the agent states and interactions over time can yield synthetic data containing detailed and complex attributes, and patterns that cannot be modeled using other techniques; this is because the outcomes of the simulations are not assumed beforehand or explicitly decided but arise only based on the interactions of the agents over time [36]. However, running simulations for ABM can be challenging for a model with many agents and behavioral rules as it would be computationally intense to incorporate the large number of interaction possibilities for the agents. Additionally, developing a simulation requires sufficient domain knowledge and data/details pertaining to the setting to ensure quality and accuracy.

2.2.3 Using machine learning

Machine learning models can be used to generate synthetic data from real-world data. Instead of fitting real-world data to a statistical distribution and identifying the distribution parameters, machine learning models can be fit to the data to generate synthetic data. This approach involves training machine learning models on real-world data to learn their characteristics and values to output artificially synthesized data [70, 76]. An advantage of using machine learning models is when the real-world data does not have a good fit with an existing distribution type, that is, the statistical properties of the data vary and cannot be captured by a type of known distribution; in this case, a machine learning model can be trained to generate data that correlates with the real-world data [31].
There are several popular machine learning models that are used for fitting real data and generating synthetic data. Some of the models used are:

- Decision trees and Random Forests (RF) – Decision trees are models that split a dataset into subsets based on event outcomes and form a tree-like hierarchical structure. RF are ensemble models that use decision trees as building blocks. Decision

trees and RF can be reverse-engineered to generate synthetic data by sampling from leaf nodes or by using the structure of the trees to create new data points [21, 34].

- k-Nearest Neighbors (kNN) − kNN is a supervising learning classifier that uses the proximity of a data point to surrounding points to classify the data point. kNN can be used to form classes based on proximity and to generate synthetic data by using interpolation between points in a group to create new points [61, 144].

- Regression models − Regression models such as linear regression and polynomial regression are models that fit an equation to the data to establish a relationship between independent variables and dependent variables. Regression models can be used to generate synthetic data by using random samples to get predictions from a model trained on real data and then adding noise to it [117].

- Synthetic Minority Over-sampling Technique (SMOTE) − SMOTE is an oversampling technique that generates synthetic data for a minority class by using interpolation between real data points [25].

A variety of machine learning models can be used to generate synthetic data and are advantageous due to their non-parametric approach in situations when the real data does not adhere to an existing distribution. However, simple machine learning models (non-deep learning) also fail to capture intricacies, high-dimensional patterns, and complex relationships or trends. Moreover, using non-deep learning models requires significant feature engineering to make the trends easier to capture making the process less automated and more dependent on intervention from users.

2.2.4 Using deep learning

Deep learning methods are popular for generating synthetic data for complex data types such as images, videos, speech, and time series. Deep learning models such as Variational Autoencoders (VAEs), Recurrent Neural Networks (RNNs), Generative Adversarial Networks (GANs), or Transformers can be used to generate synthetic data [4]; these models train on real data to learn its characteristics and patterns with the goal of generating or outputting similar samples. The idea behind using these neural networks is that they are trained using a smaller number of parameters than the amount of input data so that the models are required and forced to discover underlying characteristics to generate it [95]. VAEs encode real data into a low-dimensional latent space by learning the representation of the data. Synthetic data can be generated by sampling from the latent space and decoding the samples to get similar synthetic data [85, 136]. RNNs such as Long Short-Term Memory (LSTM) networks are used for generating sequence data such as time series due to their ability to learn temporal patterns. LSTMs use a gating mechanism that allows them to control and retain information from previous inputs allowing them to learn temporal relations [4, 7]. GANs are extremely suitable for generating synthetic data, especially for images and videos. GANs are composed of two neural networks − a generator and a discriminator. The two neural networks are trained together to contradict the purpose of each

other. The generator is trained to generate synthetic data and the discriminator is trained to distinguish between real data and synthetic data [123]. Lastly, transformers use an architecture known as the attention mechanism (or self-attention mechanism) to capture long-range relationships and dependencies in sequences, especially for Natural Language Processing (NLP). Attention mechanisms calculate the weights of elements in a sequence and their contribution to the occurrence of a particular element, this calculated weight provides the attention scores that lead to dependencies being captured. Transformers can generate synthetic data by using the captured relations and dependencies from the real data [53].
However, the performance of these models largely relies on being trained on large volumes of data. Using deep learning models to generate good-quality synthetic data demands the availability of large volumes of data for training. Furthermore, the required data needs to be broad and diverse to ensure that the model covers the entire spectrum of patterns and relationships possible in the real data.

2.3 Machine Learning

There are numerous machine learning models that are used for a variety of tasks ranging from predictive maintenance and synthetic data generation to a lot more. Some of the popular machine learning models mentioned throughout this thesis are described in this section.

2.3.1 Regression Models

Regression models are models that establish a relationship between a dataset's dependent and independent variables; the models do so by fitting an equation to the data and determining the correlation between the variables and the analysis of its significance [38]. There are two broad categories of regression models − a) simple regression models in which there is only one independent variable and one dependent variable, b) multiple regression models in which there are several independent variables impacting one dependent variable. Moreover, there are several variants of regression models, however the most popular is linear regression. Linear regression uses a single regression line (also known as the best-fit line) to establish a linear relationship between a dependent variable and an independent variable. The linear relation using simple linear regression is given in Equation 2.1, where Y is the dependent variable, m is the slope of the line, X is the independent variable, c is the y-intercept, and ϵ is the error term [84].

$$Y = mX + c + \epsilon \tag{2.1}$$

For more complex relations, polynomial regression can be used to establish non-linear relationships by using a curved line instead. Polynomial regression is an extension of linear regression and can be characterized by Equation 2.2 where θ is the coefficient determining

the slope of the independent variables and n is the degree of the polynomial [96]. The θ parameters are often calculated using the maximum likelihood estimation (MLE) by testing different values of θ iteratively to optimize the equation for finding the best fit. Note that Equation 2.2 provides the equation for simple polynomial regression.

$$Y = \theta_0 + \theta_1 X + \theta_2 X^2 + ... + \theta_{n-1} X^{n-1} + \theta_n X^n + \epsilon \tag{2.2}$$

Linear regression and polynomial regression models are useful in cases where the dependent variable is continuous. However, in cases where the dependent variables are discrete, or in classification tasks, logistic regression can be used. Logistic regression is a regression model that estimates the probability of mutually exclusive outcomes. In essence, the output of a logistic regression model is a binary classification and provides one of two possible outcomes based on probability [57].
Regression models are simple and flexible models for performing analysis on datasets and establishing relationships between variables. However, regression models are sensitive to outliers and assume that the data is independent (except for the dependent variable). In essence, if the independent variables are highly correlated, understanding their impact and effect on the dependent variable can become difficult.

2.3.2 Decision Tree and Random Forest

A decision tree is a supervised learning algorithm that has the structure of a tree consisting of branches, nodes, leaves, and a root. A decision tree starts at the root and follows a hierarchical structure down to the leaves. The tree branches downward and splits at nodes based on decisions to form subsets of the dataset. The bottom of the tree contains the leaves that denote all the possible outcomes for a dataset. In other words, a decision tree creates a flowchart to represent decision-making within a dataset by finding points to split the data to ensure that all the points in a dataset have been grouped under appropriate labels. A decision tree can be used for classification and regression tasks [125]. There are several advantages of decision trees such as they offer a simple structure that is easy to interpret for data analysis purposes, the performance of decision trees does not rely on their suitability to the underlying distributions of the data, and their robustness to outliers. However, decision trees can easily overfit if the structure and complexity of the tree are not carefully tuned. Furthermore, the trees need to often be regularized by eliminating branches that split on low-importance features through a process called pruning [83]. Also, decision trees are unstable to changes in the data, that is, slight changes in the data can produce very different trees.
A combination of decision trees can form an ensemble known as the random forest model. Random forests consist of multiple decision trees and combine their outputs to generate a final result. Random forests follow an ensemble method known as bagging (or bootstrap aggregation). Bagging implies the random sampling of data with replacement (this implies that the data points can be chosen more than once) to create subsets of the data [18]. The subsets created are then used to train individual decision trees and an aggregate of the outputs of the decision trees is used as the final output of the random forest model. Doing so ensures a forest of trees with low correlation as they are generated on random subsets

containing random features; this reduces variance in a noisy dataset and mitigates the risk of bias and overfitting. Additionally, random forests can also be used for classification and regression tasks. Random forests provide higher accuracy when compared to individual decision trees and reduce the risk of overfitting due to the bagging approach and randomness. However, due to the complex structure of random forests, they become difficult to interpret for data analysis purposes when compared to a single decision tree. Furthermore, they also require a lot more resources when compared to individual decision trees [108].

2.3.3 Support Vector Machines

Support vector machine (SVM) is a supervised machine learning learning algorithm that works by finding a line or hyperplane that partitions the data based on their classes. An optimal hyperplane or line found maximizes the distance between the closest data points of two separate classes. There can be multiple hyperplanes to differentiate among multiple classes. The optimal hyperplanes are determined by the maximum distance between support vectors, which are vectors that run through data points of the two classes that are closest to each other [102,126]. Hence, hyperplanes form a decision boundary and reside in the optimal space between two adjacent support vectors that maximize the closest distance between the two classes. There are several variants of SVMs, but they can be grouped into two broad categories [114] −

- Linear SVM: Linear SVM is used in cases where the data can be separated into two classes using a straight line.
- Non-Linear SVM: Most situations in the real world present non-linearly separable data, and this is where non-linear SVMs can be used. In such situations, kernel functions are used to map the data into a higher-dimensional feature space and enable linear separation. This technique is known as the "kernel trick" that converts a non-separable problem into a separable one.

SVMs can be used for classification as well as regression tasks, however, are more commonly used for classification tasks in high-dimensional data as they scale well with dimensionality. Furthermore, using the "kernel trick" appropriately allows for various complex problems to be solved by using SVM. However, choosing an appropriate kernel function is not easy and may require significant hyperparameter tuning. Furthermore, the results from SVMs are not easy to interpret for data analysis purposes to understand causes and effects.

2.3.4 Neural Networks

Neural networks are machine learning algorithms that fall into the category of deep learning. Neural networks are designed to imitate the human brain in interpreting and processing information to give out a result. The structure of neural networks consists of numerous interconnected nodes (known as neurons or perceptron [110]) across several layers [11]. Neural networks are also known as artificial neural networks (ANNs).
The architecture of a simple neural network can be described as follows −

- Input layer − This is the layer through which information is fed to the neural network. The data is processed by the input layer and passed on to the next layer.
- Hidden layer − There can be one or more hidden layers in a neural network. The data is passed through the hidden layers and analyzed based on the importance of the features by using weights and biases.
- Output layer − This is the last layer and it produces the final output for the given input. The output layer can consist of any number of neurons depending on the task. For example, the output layer may contain only one neuron for binary classification tasks.

The layers of a neural network contain neurons that usually calculate a weighted sum of any input they receive. In essence, weights are assigned to the inputs of all neurons when the neural network is initialized; these weights determine the importance of their respective variables, that is, the larger the weight associated with a variable the more it contributes to the final output. Additionally, a bias term is assigned to every neuron. So the inputs to a neuron are multiplied with their respective weights and are added together with the bias term. The weighted sum calculated from a neuron is then passed through an activation function that determines if the information from that neuron will be passed forward; if the output of the activation exceeds a threshold, the output is passed to the next layer. Hence, the output of the previously described neuron will then become the input of another neuron and the process continues forward, this is known as a feedforward network that characterizes most neural networks [2]. On the other hand, neural networks use an algorithm known as backpropagation for training and to improve their performance. Backpropagation involves adjusting the weights in a neural network iteratively through feedback loops. The weights are adjusted to give more importance to variables that help in making correct predictions and less importance to variables resulting in incorrect outputs [109].
Neural networks are very popular due to their performance and accuracy. There are several benefits of using neural networks for machine learning tasks such as their flexibility to learn from almost any type of data that can be translated into a numeric form, and their ability to adapt to new information and to discover hidden and complex patterns in high-dimensional datasets. However, neural networks often require a large volume of data to train on to ensure high accuracy and therefore, are computationally expensive to train. Furthermore, neural networks are often considered a "black box" as they offer little to no knowledge about the relationships between the independent and dependent variables [14] and make it difficult to interpret the performance of neural networks.

2.3.5 Recurrent Neural Networks and Long Short-Term Memory Networks

Recurrent neural networks (RNNs) belong to the category of deep learning and are a type of neural network. They are most commonly used for sequential data and time series

data such as natural language processing (NLP), forecasting problems, and other ordinal or temporal problems. The key difference between RNNs and traditional neural networks is that RNNs contain a memory component that allows them to take into account selective information from previous inputs to help in making predictions for the current input. Furthermore, traditional feedforward networks have different weights assigned to different neurons but RNNs share the same weight throughout a layer [81]. Although there are some differences in the functioning and architecture between traditional neural networks and RNNs, they both use backpropagation for training and modifying the weights. However, the backpropagation algorithm for RNNs is slightly different (known as backpropagation through time (BPTT)) as the errors for every time step are summed across a layer due to the weight sharing [137].
RNNs perform well with sequential data and time series due to their ability to realize context from previous inputs that influences the current inputs, furthermore, since they share weights, they also increase training and memory efficiency. However, RNNs are known to suffer from two things − vanishing gradients and exploding gradients. The gradients are values that dictate in which direction should the weights be shifted to reduce errors. The problem of vanishing gradients implies that the values of the gradients keep on decreasing throughout the training process until they become extremely small (approximately 0) and this causes the neural network to stop learning anymore. On the other hand, the problem of exploding gradients implies that the values of the gradients become too large during the training process and result in an unstable network that keeps missing the minima due to the large changes in the weights and is unable to converge. Either way, the problems arise due to the multiplication of derivatives across the layers of a neural network, so depending on the size of the derivatives, gradients can explode or vanish. In RNNs, the gradient depends on previous inputs so it exponentially grows or shrinks due to the large number of multiplications at each time step [98].
Long Short-Term Memory (LSTM) networks are a type of RNN created to avert the vanishing gradient problem. RNNs face difficulty in remembering information over a long time due to their vanishing gradient problem, however, LSTMs are designed to avoid that and capture long-term dependencies [82]. The components of an LSTM are −

- Input gate − It determines what new information will be learned from the current input.
- Forget gate − This controls what information should be retained from the previous cell and what should be discarded.
- Output gate − This passes the updated information from the current cell based on the current input and previously stored information.
- Hidden state − Similar to an RNN, LSTMs also have hidden states to represent short-term memory from the previous timestamp to the current timestamp.
- Cell state − This represents the long-term information that is stored in the network across all timestamps. Information is added to and removed from this by gates based on importance.

LSTMs are designed to retain long-term information and avoid the problem of vanishing gradients by using a complex architecture that controls what information is retained based on relevance and importance [143]. However, due to the increased complexity, LSTMs take longer to train and are computationally more intensive than simple RNNs.

2.3.6 Generative Adversarial Networks

A generative adversarial network (GAN) is a deep learning model that is made up of two neural networks. The two neural networks are trained to compete against each other to generate realistic data. GANs are unsupervised learning models that can be used to generate synthetic data such as images, audio, videos, time series, and other complex data types [5]. The two neural networks in a GAN are −

1. Generator − It learns to generate realistic data by modifying an input noise vector with the goal of "fooling" the other neural network.
2. Discriminator − The goal of this network is to distinguish between real data and the artificial data generated by the generator.

The training of a GAN involves fake data generated by the generator to appear as real as possible to the discriminator. The outputs of the generator are fed to the discriminator along with real data so that it can distinguish between the two. Throughout the training process, the generator keeps improving with the goal of maximizing the error rate of the discriminator in distinguishing real and fake data. GANs also use backpropagation to improve the error rates of both − the generator and discriminator.
GANs use a loss function to represent the distance between the data generated by the GAN model and the real data [29]. The standard loss function used by GANs, known as the minimax loss, can be expressed as in Equation 2.3, where E_x is the expected real data, $D(x)$ is the discriminator's probability that the instance x is real, E_z is the expected value of the fake data generated by the generator, $G(z)$ is the output from the generator, z is the input noise vector to the generator, and $D(G(z))$ is the discriminator's probability that the instance is fake. The generator tries to minimize this function, whereas the discriminator tries to maximize it, hence the name: minimax loss function [50]. Specifically, the generator tries to minimize the second part of the equation: $log(1-D(G(z)))$ so that the discriminator is unable to detect fake samples.

$$E_x[log(D(x))] + E_z[log(1 - D(G(z)))] \tag{2.3}$$

The use of GANs is advantageous for many reasons as they are known to generate realistic data such as images, videos, text, and audio. They are also used to enhance the quality of existing data. Despite the promising results from GANs, they require careful hyperparameter tuning and experimentation due to their complex architectures to achieve good results. Moreover, the training of traditional GANs may suffer from problems like vanishing gradients and mode collapse. Vanishing gradients in GANs may occur when the discriminator performs significantly better than the generator and so the generator is

heavily penalized creating instability in the network. Mode collapse in traditional GANs occurs when the generator only produces very limited and similar samples that work well against the discriminator. This results in the generator not being able to learn to generate a wide range of samples as was intended [146]. Wasserstein Generative Adversarial Network (WGAN) was introduced as an alternative to traditional GANs to tackle the problem of mode collapse and vanishing gradients. In WGANs, the discriminator is replaced by a critic; the critic does not distinguish between real and fake data but gives out a score without being constrained between 0 and 1. The critic's score is bigger for instances that it thinks are real than for instances it thinks are fake [9]. The loss functions used by WGANs are expressed in Equation 2.4 and Equation 2.5, where $D(x)$ is the critic's output for a real instance x, $G(z)$ is the output from the generator, z is the input noise vector to the generator, and $D(G(z))$ is the critic's output for a fake instance.

$$Critic\ Loss : D(x) - D(G(z)) \tag{2.4}$$

$$Generator\ Loss : D(G(z)) \tag{2.5}$$

Both the generator and the critic try to maximize their loss functions. The critic tries to maximize the difference between its outputs for real and fake instances, and the generator tries to generate samples in an attempt to maximize the discriminator's output for the fake samples.

Chapter 3

Gradual Fault Detection

The primary aim of this thesis is to provide an approach to conduct predictive maintenance by detecting gradual faults (or gradual deterioration) in a public bus. Gradual faults are often not noticeable by the sensors on a vehicle nor visible to the eye until the deterioration progresses to a severe level, resulting in either component failure or immediate need for maintenance. The difference between regular faults and gradual faults is characterized by the extent to which a component has suffered degradation as well as the resultant loss due to the degradation. Regular faults are either abrupt or caused by degradation over time which was not corrected or prevented but both cases result in component failure or severe deterioration. On the other hand, gradual faults could be described as the intermediate stage between the normal functioning of a component and the failure of a component. Gradual faults deteriorate the component slowly over a period of time and the effect of such faults is often not noticeable due to the small magnitude of impact they cause. Furthermore, gradual faults are much easier to rectify because the component suffering deterioration is still in a recoverable stage, that is, it has not suffered severe degradation or failure yet. Additionally, the detection of gradual faults could help in maintaining the functions of a component system (a group of components dependent on each other's functioning) by fixing a deteriorating component before the component fails and hinders the functioning of the whole component system. [142] describes gradual faults as the most difficult to detect and thus, it is essential for systems to be able to detect gradual faults in a timely and accurate manner. The primary concern of gradual fault detection is to detect and identify anomalies or unusual trends before they surpass the stage of using preventive measures to avoid failure in components. Therefore, detecting gradual faults in a timely manner is important so that the deterioration does not progress to a stage where only corrective measures are able to resolve the component's issues. [87, 124] describe gradual faults as having no evident characteristics due to their slow progression. Therefore, existing systems to detect faults are insufficient to avoid component failure as the systems are only able to detect abnormalities after a certain threshold has been crossed by which time the component has already failed or suffered serious deterioration resulting in the need for immediate repairs or replacements [55, 87].
Regular faults are often detected by using rule-based approaches such as the approaches mentioned in [20, 27, 33, 66, 122]. Such approaches include establishing a threshold from

a knowledge base to identify a component as faulty when the readings from it cross the set threshold. Although such methods work well for the task of fault detection, they are unable to prevent the fault by predicting the failure. This is due to the small magnitudes of deterioration that lead to faults that go unnoticed by threshold-based systems. Moreover, even advanced machine learning techniques implemented for the task of fault detection may not be able to detect gradual faults due to the varying nature of such faults. The task of detecting gradual faults involves detecting abnormal temporal patterns and looking for the smallest indicators of deterioration by comparing trends observed in component performance values across several component groups and/or machines. On the other hand, the task of detecting regular faults could be as simple as observing the final output values of a component group's performance and comparing it to a threshold value to classify it as faulty or not.
The varying nature of gradual faults makes it essential to develop methods that are able to detect abnormalities on such a small scale. Developing gradual fault detection mechanisms could not only prevent faults and failure altogether, but also ensure safety, efficiency, and a decrease in maintenance costs.

3.1 Predictive Maintenance Algorithm

This section presents an algorithm to conduct predictive maintenance by using unsupervised clustering. K-Means clustering is used for detecting deterioration and unusual trends in the generated data. The metrics used to perform the clustering are Euclidean distance and Dynamic Time Warping (DTW) [15,23,68,88]. The primary idea is to form a) clusters containing normal buses, and b) clusters that identify component deterioration and faults by grouping similar deterioration and degradation trends together. K-Means clustering using Euclidean distance is used as the baseline to compare to K-Means clustering using DTW. Euclidean distance is chosen as the base approach, because it is the common choice for clustering due to its low complexity and simplicity; it involves calculating the distance between a time series and the cluster centroid point-by-point without taking into account any trends or patterns. However, complex trends such as those found in coolant temperature and engine system readings are difficult to distinguish using just a simple metric, which is why DTW is used. DTW temporally aligns sequences and calculates the distance between points that are similar even if the indices of the two points do not align in the time series. However, the complexity of DTW is higher compared to the Euclidean distance method $-$ $O(n * m)$, where n is the length of the first sequence and m is the length of the second sequence [113]. Therefore, DTW should be used only if Euclidean distance fails to form accurate clusters. Additionally, the silhouette score is used to evaluate the quality of clusters, and the elbow method [127] is used to select the optimal value of K (number of clusters). Algorithm 1 provides an idea of how predictive maintenance can be conducted for a machine. The algorithm can be described as follows:

1. Select sensors relevant to a component or component group with the help of domain experts or knowledge base.

2. Restructure the sensor data so that each sensor is represented by a time series dataset containing several sequences, that is, a single time series dataset will belong to one sensor and will contain several instances or sequences, and each instance or sequence will contain multiple sensor readings over time. Instances could be imagined as individual buses and sequences could be imagined as separate trips.

3. Apply preprocessing steps such as reshaping, scaling, and cleaning the sensor data so that it can be used to train a model.

4. Choose a value of K (number of clusters) based on two methods − a) existing knowledge of the number of different types of faults to be detected or b) using the elbow method.

5. Train two K-Means clustering models for each time series dataset − one using Euclidean distance as a metric and the other using DTW.

6. Use the silhouette score as a metric to evaluate clustering quality. If the silhouette score of the Euclidean model is greater than or equal to the score of the DTW model, select the Euclidean model as the final clustering model for that sensor, otherwise select the DTW model as the final clustering model for that sensor.

7. Retrieve the buses belonging to the cluster with faults from the chosen model by sorting the clusters by size and selecting any clusters after the largest one (the largest cluster will have all the normal sensor readings). Use the retrieved buses to identify the causes and types of faults with the help of domain experts to perform the necessary maintenance.

Algorithm 1 Predictive Maintenance Algorithm

Require: A dataset containing sensor readings
Ensure: Relevant Sensor Selection

```
TimeSeries ← SensorData                          ▷ Restructure and reshape
Preprocessing(TimeSeries)
if Types of faults known then
    K ← NumberOfTypes
else
    K ← ElbowMethodOptimal
end if
EuclideanKMeans.fit(TimeSeries)
DTWKMeans.fit(TimeSeries)
if EuclideanSilScore ≥ DTWSilScore then
    FinalModel ← EuclideanKMeans
else
    FinalModel ← DTWKMeans
end if
FinalModelClusters.sort(by=size, ascending=False)
Faults ← FinalModelClusters[1 :]
```

Section 3.2 describes in detail the process of implementing the K-Means clustering algorithm for the proposed algorithm, that is, K-Means clustering using the Euclidean distance as the metric, and K-Means clustering using DTW as the metric. Section 3.3 further describes and explains the use of K-Means clustering for the task of time series clustering. It contains information regarding the preprocessing of time series (specifically sensor data), hyperparameter selection related to selecting a value of K as well as initializing cluster centers (centroids) for the models.

3.2 K-Means Clustering – Overview

K-Means clustering is an unsupervised machine learning algorithm that involves assigning data points to clusters by checking how close a data point is to a cluster. K-Means clustering models group data points based on their similarity by comparing patterns and variations between them and the models do not require any prior training to do so. The absence of labeled data makes K-Means clustering a useful and essential algorithm. It is considered one of the most powerful and popular clustering approaches [6, 120].
Algorithm 2 presents the structure of the K-Means clustering algorithm. The general overview of the K-Means clustering algorithm can be given as follows:

1. Choose a value for K, which is the number of clusters.
2. Initialize centroids, which are the cluster centers.
3. For each data point, the distance between the data point and all centroids is calculated, and the data point is assigned to the nearest centroid.
4. The cluster centers are re-initialized by calculating the average of all the data points belonging to their specific clusters
5. Steps 2 to 4 are repeated for a set number of iterations to reach convergence.

Algorithm 2 K-Means Clustering Algorithm

Require: A dataset
$K \leftarrow NumberOfClusters$
$Centroids \leftarrow InitializationValues$
for given number of iterations **do**
  for i in length($dataset$) **do**
    $NearestCentroid \leftarrow$MinDistance($dataset[i]$, $Centroids$) ▷ Find nearest centroid by comparing distances
    $Clusters[NearestCentroid] \leftarrow dataset[i]$ ▷ Data point assigned to the nearest cluster
  end for
  $Centroids \leftarrow$ClusterMeans($Clusters$) ▷ Centroids are updated with cluster means
end for

The evaluation of the quality of clusters created using the K-Means algorithm is most popularly done by two methods − a) using the silhouette score or other distance measures, and b) validating the clusters by visually examining the cluster plots [63, 134]. The silhouette score is calculated as $b - a/max(a, b)$, where a is the mean distance of a point with the other points of its cluster and b is the distance between the single point and its nearest cluster. The silhouette score values are in the range $(-1, 1)$, where a score of closer to 1 indicates strong and accurate clustering while a score closer to -1 indicates poor clustering. Other popular evaluation metrics such as accuracy are not entirely relevant or appropriate due to the machine learning approach being unsupervised. However, in scenarios where labeled data is available and clustering is used, the aforementioned methods can be used.

3.2.1 Euclidean K-Means

The simple form of K-Means clustering involves assigning data points to clusters by comparing the square root of the sum of squared distances or the Euclidean distance, however, advanced variants of the K-Means clustering algorithm use more complex metrics to compare the distance between a data point and the cluster center [62]. The simple K-Means algorithm − using Euclidean distance − can perform well for many tasks including data without complex trends and data containing easily distinguishable elements. The primary goal of using the Euclidean approach is to calculate the distance of all data points to cluster centers and then assign the data points to clusters that have the smallest Euclidean distance to the data point.
One limitation of using the Euclidean distance as a metric for clustering is that the time series being compared need to be of the same length. This would require additional preprocessing steps to make sure that all the time series have the same length in a way that the temporal trends are preserved and there is no significant data loss. On the other hand, the simplicity of calculating the Euclidean distance results in a low complexity − $O(n)$, where n is the length of the time series. Equation 3.1 based on the information in [115] provides the idea of how to calculate the Euclidean distance between two time series or sequences; where E is the calculated Euclidean distance, N is the length of both sequences, x_i and y_i are corresponding points from the sequences.

$$E = \sqrt{\sum_{i=1}^{N} (x_i - y_i)^2} \tag{3.1}$$

3.2.2 Dynamic Time Warping K-Means

Dynamic time warping (DTW) is a technique that aligns two sequences in a non-linear way to match each other's trends [15, 88]. The idea behind using DTW is that it aligns similar trends between two time series that are being compared. The implication of this idea is that the comparison does not depend on the indices or the length of the time series, that

is, a single point from one time series is matched with the most suitable (and similar in terms of value) point from another time series while preserving the sequence of values even if the two time series are not aligned to exhibit congruence. The result of this is that the distance between similar points is calculated to obtain a more definitive and accurate result that takes certain trends into account. DTW is used as a metric to calculate the distance between time series and centroids in K-Means clustering. Doing so will ensure that similar structure time series that are not aligned in a time-space are not clustered separately but are clustered together based on their similarity to the cluster center. The quality of the cluster center to accurately represent such temporally varying yet similar time series also depends on the use of DTW for finding the mean of a cluster. However, a major drawback of using DTW is its high complexity of $O(n * m)$ where n is the length of the first time series and m is the length of the second time series. The quadratic complexity of DTW is not always feasible, especially in cases where long time series need to be compared [113]. Another key aspect of using DTW for K-Means is DTW Barycenter Averaging (DBA). DBA iteratively uses dynamic time warping and aligns all the series with an evolving average [23, 68]. This process results in a more accurate centroid computation that accurately represents the data in its cluster. A barycenter could be imagined as a cluster centroid (of a time series) and the goal of DBA is minimizing the distance between itself and the data. This differs from the Euclidean approach of finding the average barycenter because the Euclidean approach calculates the arithmetic mean of points across all the time series based on their index and without any alignment. Figure 3.1 taken from [128] provides a visual illustration of the difference between how points are aligned and compared using DTW and the Euclidean distance.

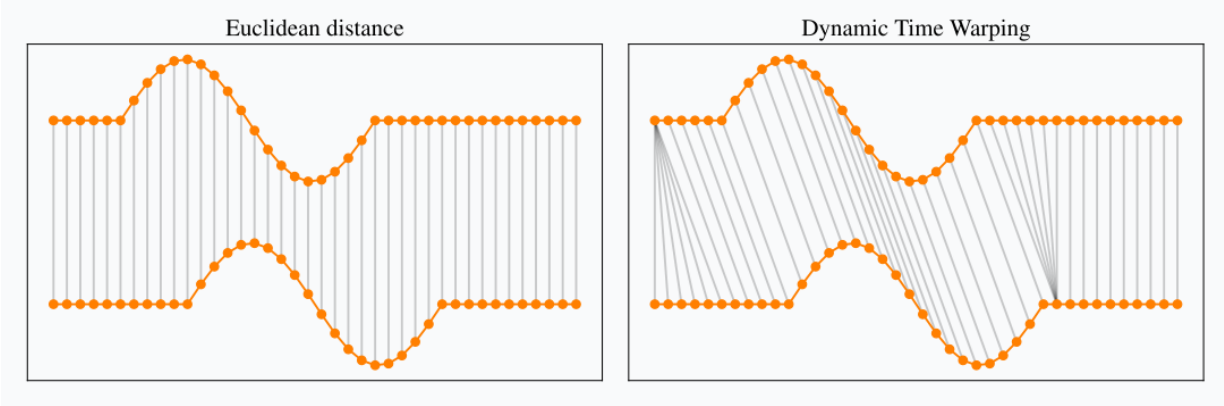

Figure 3.1: Comparison between Euclidean and DTW alignment

There are certain constraints and rules that are followed when finding an optimal path using DTW [147]:

- Every point from a sequence must form a one-to-many mapping with the points from another sequence, that is, every point from one sequence must be matched with at least one point from the other sequence.

- The first index of one sequence must at least be matched with the first index of the other sequence.
- The last index of one sequence must at least be matched with the last index of the other sequence.
- The mapping of indices from one sequence to the indices of another must be monotonically increasing, that is, the order of time is preserved by ensuring that no mapping from one point in a sequence goes back in time.

The mathematical formulation of DTW can be expressed as follows based on [68]: Consider two time series $X = [x_1, x_2, ..., x_i, ..., x_n]$ and $Y = [y_1, y_2, ..., y_j, ..., y_m]$. The first step involves creating a cost matrix M of dimensions: n x m and every point in the matrix (i, j) is a score of alignment between the points: x_i and y_j. The cost matrix can be filled up by starting from the initial point $(M(1, 1))$ and using Equation 3.2:

$$M(i, j) = |x_i - y_j| + min(M(i-1, j), M(i, j-1), M(i-1, j-1)) \tag{3.2}$$

After filling the cost matrix M, a warping path can be calculated that maps the elements of X and Y and minimizes the distance between them by aligning them. The warping path can be calculated by starting at the extreme point $(M(n, m))$ and traversing to the initial point $(M(1, 1))$ and using Equation 3.3; where P is the warping (or optimal) path and K is the length of the path:

$$P_{k=1}^{K} = min(M(i-1, j), M(i, j-1), M(i-1, j-1)) \tag{3.3}$$

Lastly, after finding the optimal path P, the normalized distance D can be calculated from Equation 3.4:

$$D = \frac{\sum_{i=1}^{K} P_i}{K} \tag{3.4}$$

3.3 Preprocessing and Hyperparameter Tuning for Time Series K-Means

3.3.1 Preprocessing

Sensor data from vehicles is usually collected as measurements over time, therefore, making the data collected a time series. Sensor data is usually noisy and not smooth. Furthermore, the recorded sensor readings may contain missing or incorrect values [58]. This is a result of incorrect values being transmitted, sometimes due to the sensitivity of sensors or other malfunctions, and also due to the high frequency of transmission of data from sensors [1]. Hence, raw sensor data usually cannot be used for training machine learning models directly after being stored. However, despite the presence of noise throughout the time series, useful information and patterns can be extracted by applying relevant preprocessing techniques. The preprocessing techniques being used usually depend on the task

at hand [129] but some popular techniques for time series are discussed in this subsection. The presence of null or missing values in a time series hinders the training of a machine learning model. However, there are simple methods to fix this situation depending on the magnitude of null values present. If the magnitude of missing values present in a time series is insignificant and very small, it could be considered safe to drop these values, that is, remove the time steps with null values from the data, without worrying about information loss. On the other hand, if there are many time steps with missing or null values scattered across a time series dataset, then dropping the values for those time steps could result in information loss. Null values are often replaced by the mean value or the mode of that group but it is not the best practice for a time series as the sensor's dependency on the time steps might not be consistent. Therefore, a forward-fill method should be used so that the first non-null value after a missing value (or values) is used to fill it [91, 106, 135]. This ensures that the temporal trends are preserved as well as possible.

Another issue with sensor time series data is its characteristic of being noisy and not smooth. A very popular approach to fix this issue and smoothen the data to represent underlying trends is to use the moving average with a reasonable window size [28, 139, 148]. To calculate the moving average, the sum of W consecutive sensor readings is divided by W iteratively throughout the time series, where W is the size of the moving average window. Doing so, can not only help in the removal of outliers and erratic readings from the data but can also help in the understanding of trends and distributions in the data by providing a more clear visual representation when plotted as opposed to just plotting the raw data. Additionally, instead of using all the data from a sensor to train a model, considering only a value range (such as quartile or percentile values) that is suitable for and relevant to the model's task may improve its performance [37]. In essence, if a sensor time series contains values highly concentrated around its mean, and a model for a task related to its lowest values needs to be trained, considering only values under a low percentile might be more relevant and useful.

Based on the proposed algorithm in Section 3.1, individual time series datasets only contain values from a single sensor so scaling the data is not necessary for the algorithm in most cases. However, based on the sensor data and the task at hand, scaling could be applied across the time series.

3.3.2 K Selection

The value of K in K-Means stands for the number of clusters the model will group the data into. The value of K must usually be set manually before beginning the training of the clustering model [120]. This is one of the shortcomings of using the K-Means clustering algorithm because the performance of the model and the clustering quality hugely depend on the value of K to be optimal [30, 101]. There are various methods to find an optimal value for K such as – a) using a knowledge base to identify the clustering goals and setting a value based on that, b) using the silhouette score to determine clustering quality [118], and c) the elbow method which involves picking the optimal value for K based on the sum of squared distances. The elbow method is a reliable and popular technique for either

finding an optimal value of K or for validating the value of K chosen by other means [80]. The task of choosing an optimal value of K for clustering sensor time series is implemented using two techniques in this work – a) Using existing knowledge on the types of faults to identify, and b) using the elbow method to validate the chosen value. The value of K is chosen based on the number of types of gradual faults to identify, doing so implies that the K-Means clustering model is expected to cluster the time series based on fault types and there is an extra cluster for all the non-faulty time series to be clustered into. Ultimately, it involves choosing such a value for K so that there is one cluster of time series with no faults (most likely the one with the highest size) and all other clusters contain time series containing faults. A specific fault gets detected and clustered into only the cluster containing time series with that type of fault. This makes detecting the type and cause of fault much easier for domain experts because the faulty time series are not jumbled up across several clusters but are instead grouped by their similar faulty trends. Additionally, if the type of faults to detect are unknown, the elbow method could be adopted straight away to choose an optimal value of K; it could also be combined with additional techniques such as silhouette score analysis and validation by domain experts.
Using the elbow method to find an optimal value for K involves two steps – a) calculating inertia (using the sum of squared distances) for a range of values of K, and b) plotting the outputs for visual analysis. Inertia is calculated by measuring the sum of squared distances between every point and its centroid [105, 112]. Equation 3.5 based on the documentation of clustering from [115] provides the formula for calculating inertia; where I is the inertia, J is the total number of clusters, N_j is the total data points belonging to cluster j, x_i is a single data point from a cluster, and μ_j is the cluster centroid of the cluster.

$$I = \sum_{j=1}^{J} \sum_{i=1}^{N_j} ||x_i - \mu_j||^2 \tag{3.5}$$

After calculating the inertia for a range of values of K, a graph containing the inertia against the value of K can be plotted to observe the decrease in the value of inertia with increasing values of K. An optimal value of K could be identified as the value after which there is no significant decrease in inertia, that is, the sum of squared distances within a cluster have been minimized reasonably and will not minimize significantly any further beyond that particular value of K. Figure 3.2 contains a graph containing inertia values (y axis) and a range of values from 2 to 6 for K (x axis) and it could be believed that the optimal number of clusters to group the data with an acceptable and reasonable sum of squared distance values is 3.

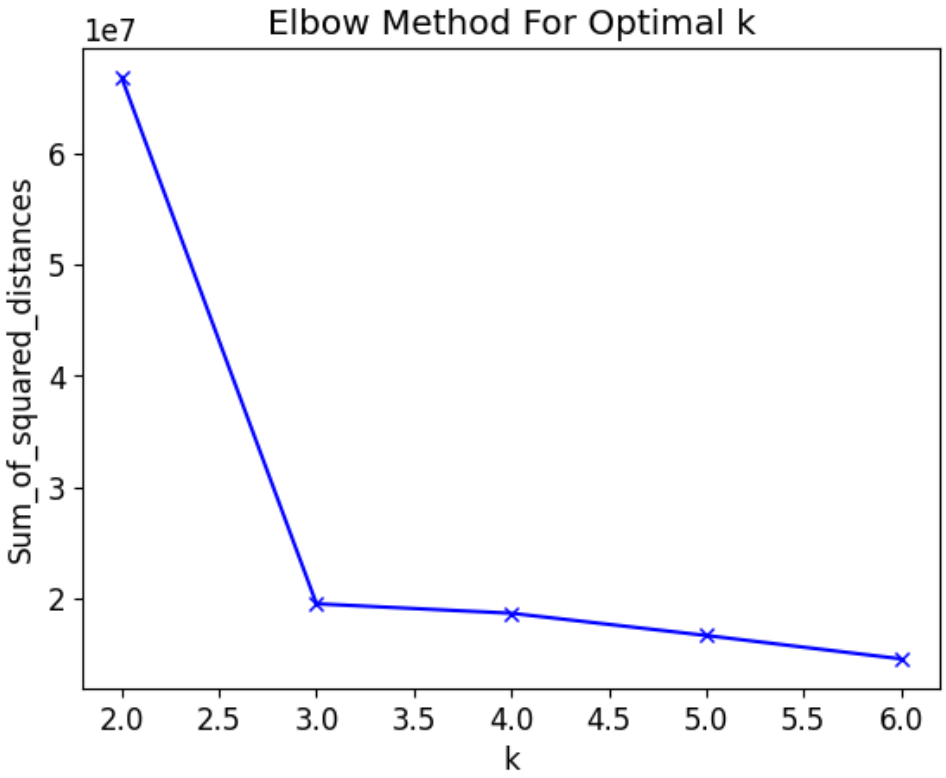

Figure 3.2: Elbow method for checking the optimal value of K

3.3.3 Centroid Initializing

The first step of the K-Means clustering algorithm involves initializing the cluster centers (also known as centroids). The initialization of the centroids can be done either by choosing arbitrary values for cluster centers, or by using random values, or by using advanced techniques. It is found that the quality of clusters significantly depends on the initialization of the cluster centers. Therefore, simply using random initialization values may not be enough in many cases as the K-Means algorithm is then more likely to fail to cluster the data into accurate representations of visible groups in the data (that is, visible groups in the data plot) [10, 51, 67]. An example that could be considered as bad centroid initialization would be in the case of a centroid being initialized between two visible groups in a data plot and another centroid being initialized far away from these two groups; in this case, the two groups would be clustered into one (belonging to the centroid in between) and the second centroid would not have any data points in its cluster. On the other hand, two centroids could be initialized very close to each other and a visible group of data points in a plot surrounding them would get divided between the two clusters. Hence, to mitigate such issues while using the K-Means algorithm using random initialization, adopting advanced techniques becomes imperative. K-Means++ [10, 111] is an advanced version of K-Means which differs only in the step of centroid initialization. K-Means++ could be considered as an algorithm just for the purpose of centroid initialization after which the steps of regular

K-Means follow.
The idea behind K-Means++ is to choose the initial values of centroids as far from each other as possible. The implication of such an idea is increasing the chance of centroids being located in separate groups of data points. Furthermore, the centroids are picked from data points so there is assurance that there will be some data points around the centroids. The steps to perform K-Means++ for a more efficient centroid initialization are as follows:

1. Randomly initialize the first centroid from the data points.
2. Calculate the distance between all the data points and the centroid closest to them that has been previously initialized.
3. Choose the data point for initializing the next centroid with the probability of being chosen directly proportionate to its distance calculated in step 2, that is, the data point with the maximum between itself and its closest centroid will be chosen as the next centroid.
4. Repeat steps 2 and 3 until K centroids have been initialized.

Chapter 4

Data

The data used in this thesis was first collected in [66]. The data consists of sensor readings from a bus of the STO transit service of the Ottawa-Gatineau region. The sensor readings from the bus were received as packets via the Society of Automotive Engineers (SAE) J1939 protocol [94], which were then decoded. J1939 messages contain two components: the Controller Area Network (CAN) ID that contains the Parameter Group Number (PGN) and the data bytes that contain the Suspect Parameter Number (SPN). The Suspect Parameter Number serves as an identifier of the sensor readings that are encoded in the data bytes and they are grouped together based on the Parameter Group Number based on the component groups. The PGN are identified by their fixed location in the CAN ID and the individual sensor readings belonging to that particular PGN can be identified based on the SPN information of that sensor. Upon locating the relevant bytes for a sensor (or SPN), the values need to be converted from hexadecimal to decimal, scaled using the unit-per-bit values, and the offset mentioned needs to be applied [45]. The J1939 specification document, which is published by the SAE, contains all the information that is required to retrieve and decode the physical data values of sensors. The sensors chosen for this work belong to the cooling system and the engine torque system of the bus. The messages were transmitted and recorded at an interval of 1 millisecond to 100 milliseconds approximately and stored in a file of size 650 megabytes; the data was collected from a single bus over a duration of approximately ten hours in November 2018.
The Parameter Group Numbers found most relevant to the cooling system of a bus are PGN 64817 (Fan Drive #2), PGN 65262 (Engine Temperature 1), and PGN 65263 (Engine Fluid Level/Pressure 1). Furthermore, the specific Suspect Parameter Numbers selected are SPN 1598 in PGN 64817 (Fan 2 Speed with rotations-per-minute units), SPN 110 in PGN 65262 (Engine Coolant Temperature with degree Celsius units), and SPN 111 in PGN 65263 (Engine Coolant Level 1 with percentage units). There are other relevant PGN-SPN groups to a bus' cooling system such as Fan 1 Speed (PGN-SPN 65213-1639), Engine Fuel 1 Temperature 1 (PGN-SPN 65262-174), Engine Oil Temperature 1 (PGN-SPN 65262-175), Engine Oil Level (PGN-SPN 65263-98), Engine Oil Pressure (PGN-SPN 65263-100), and Engine Coolant Pressure 1 (PGN-SPN 65263-109), however, these sensors could not be used because the decoded sensor readings showed that the sensors either were not able to record the values or the stored values were just noise.

The Parameter Group Numbers found most relevant to the engine torque system of a bus are PGN 61444 (Electronic Engine Controller 1) and PGN 61443 (Electronic Engine Controller 2). Moreover, the most useful Suspect Parameter Numbers chosen for this work are SPN 512 in PGN 61444 (Driver's Demand Engine - Percent Torque in percentage units) which is the requested output torque by a driver based on its accelerator pedal position and other speed control modes that are set, SPN 513 in PGN 61444 (Actual Engine - Percent Torque in percentage units) which is the calculated output torque as a percent of a reference torque value, and SPN 91 in PGN 61443 (Accelerator Pedal Position 1 in percentage units) which is the ratio of the actual current position of the accelerator pedal to its maximum possible position. Similar to the cooling system, there are other PGN-SPN groups that are relevant to the engine torque system but could not be used because no meaningful values were found in their readings. They are Engine Speed (PGN-SPN 61444-190), Engine Retarder Selection (PGN-SPN 61441-973), Remote Accelerator Pedal Position (PGN-SPN 61443-974), Accelerator 2 Pedal Position (PGN-SPN 61443-29), Actual Maximum Available Engine-Percent Torque (PGN-SPN 61443-3357), Engine Percent Load at Current Speed (PGN-SPN 61443-92), and Engine's Demand Percent Torque (PGN-SPN 61444-2432).
Section 4.1 and Section 4.2 contain detailed information regarding the data processing, synthetic generation, and fault simulation processes conducted to synthetically generate data for this work and create simulated buses with realistic trends, patterns, and gradual faults to use for training models for the task of gradual fault detection. Section 4.1 specifically describes the synthetic data generation process for the cooling system and Section 4.2 for the engine torque system.

4.1 Cooling System Data

4.1.1 Data Processing and Generation

Three sensor readings belonging to the cooling system were chosen for this work - Fan 2 Speed, Engine Coolant Temperature, and Engine Coolant Level 1 — as these were the only data bytes that contained values that seemed realistic and accurate. Upon decoding the values, the resultant dataset was stored in a comma-separated values (CSV) format, where each row had a timestamp associated with a J1939 packet. In order to generate synthetic data using a generative model with a Long Short-Term Memory (LSTM) network as the generator, categorical features had to be removed and the dataset needed to be restructured and reshaped. The dataset was reshaped to be of the format [Fan Speed, Coolant Level, Coolant Temperature] as columns for the three sensors chosen. Reshaping the dataset resulted in only one sensor reading being in one row of the dataset (e.g., 4096.0, NaN, NaN), this is because decoded values were appended to the dataset row-by-row to keep the reshaped dataset in line with the time column. The null values were filled using the forward fill method in which an initial value is used to replace the following null values in a column until a non-null value is reached. All these steps resulted in a final time series dataset that contained three sensors, one in each column.

The setup of the original dataset containing physical values from three sensors was ready after the reshaping; this dataset was used to generate synthetic data to simulate 40 buses and for the same up-time duration as the real data, which is for ten hours. The first step in the generation of synthetic data included using the DGAN model from Gretel [52]. The DGAN model is based on a Generative Adversarial Network (GAN) that uses an LSTM network as a generator. A few things to note about the model is that it does not support categorical features as mentioned previously so the dataset had to be restructured so that each column denotes a sensor and contains continuous values (decoded sensor readings). Secondly, the model only allows for a fixed length of sequences, and variable-length sequences were not supported. An example to explain the sequence length would be to consider a dataset that contains the temperature readings of a room every hour for seven days; such a dataset could contain a sequence length of twenty-four resulting in seven sequences of daily temperature readings. The data available for use in this work was only sensor data for a single bus so splitting the dataset into a fixed sequence length required repeated trying and testing of different values. Several combinations of hyperparameters were tested by trial and error to generate realistic data that also captured the temporal relations as well as included some randomness to simulate the stochastic aspects that lead to certain trends in the sensor readings. The hyperparameters configured were - the number of layers in the generator and discriminator, the dimension of the input noise, the number of units in an LSTM cell, the sequence length, the batch size, and the number of epochs. Finally, it was found that smaller batches and sequence lengths along with three or four layers in the network were the best combination. However, the resultant synthetic data failed to capture almost all of the temporal relations. This was evident in the sensor readings for the synthetic coolant temperature which were almost flat and constantly around 90 degrees Celsius in comparison to the real data, which contained temperature dips. Figure 4.1 provides a comparison between the moving average (window size of 500) of the originally recorded coolant temperature values (red) and the synthetically generated values (blue). The figures related to sensor readings in this thesis contain the physical values (readings/measurements in the sensor's respective units) along the y-axis and the indices of the datapoints along the x-axis.

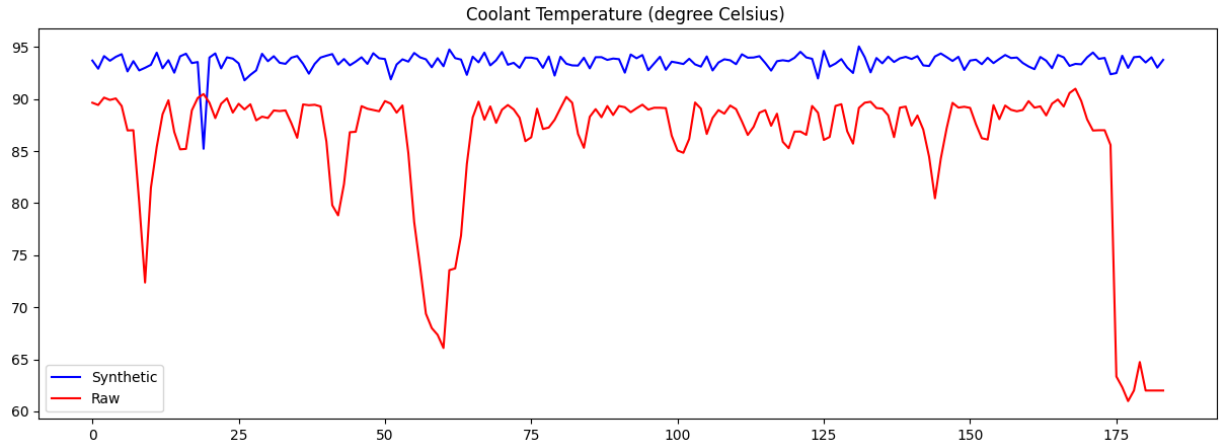

Figure 4.1: Comparison between original and synthetic coolant temperature readings

Another instance of this was visible in the fan speed readings: the real data contained only two unique values - 4096 rpm and 0 rpm - with only fifty readings that had a value of 0 displaying that the fan was turned off for a few seconds across the whole time and the fan's operational value was at 4096 rpm. The synthetic data had no values of 0 rpm but consisted of values around 4096 rpm, which does not seem to be accurate for the functionality of an electric fan. Figure 4.2 provides a comparison between the moving average (window size of 500) of the originally recorded fan speed values (red) and the synthetically generated values (blue).

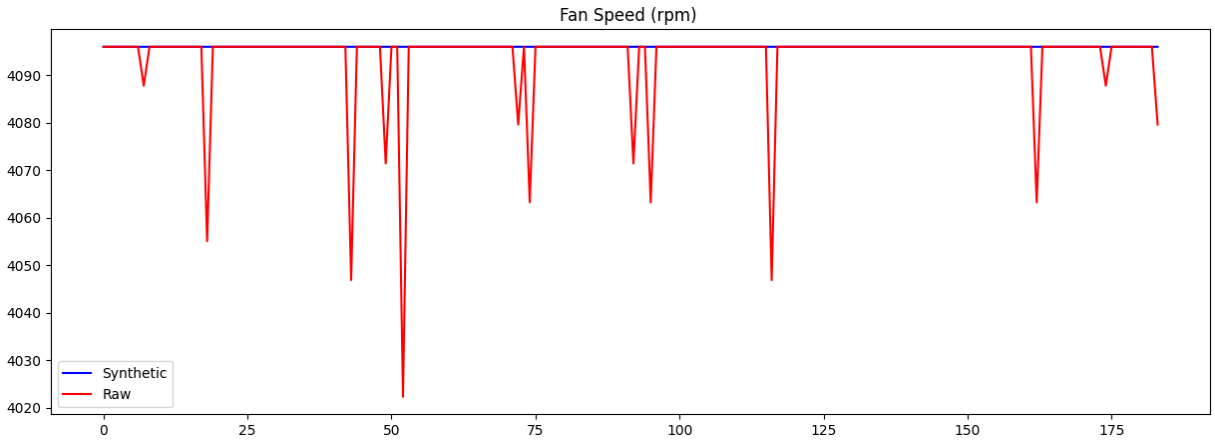

Figure 4.2: Comparison between original and synthetic fan speed readings

One benefit of synthetic data generation was observed in the values of coolant level. The real sensor readings consisted of two values: 50% and 0%; this indicates that the accurate value of the coolant level was 50% of the total container and there were a lot of readings in which the sensor was not able to record the real value so it was recorded as 0%. On the other hand, the synthetic data only had the actual value that was 50% but for all the generated buses, which does not hold true in real life, that is, buses could operate with varying levels of coolant without it causing a problem or failure. Figure 4.3 provides a comparison between the moving average (window size of 500) of the originally recorded coolant level values (red) and the synthetically generated values (blue).

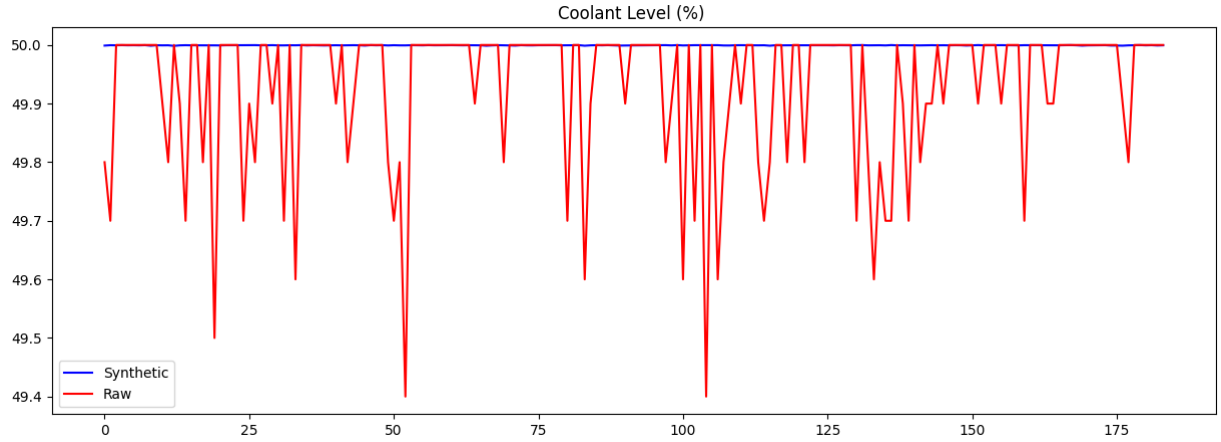

Figure 4.3: Comparison between original and synthetic coolant level readings

Lastly, to provide a comparison between the originally recorded values of the cooling system sensors and the synthetically generated values, Figure 4.4 contains plots with readings for all three sensors from the original bus, and Figure 4.5 contains plots with synthetically generated readings for all three sensors for a single bus. Due to the large disparity in value ranges among the three sensors, only a few small dents are visible in the plots in Figure 4.4 but it is visible that all the plots in Figure 4.5 are completely straight lines without any dents or trends.

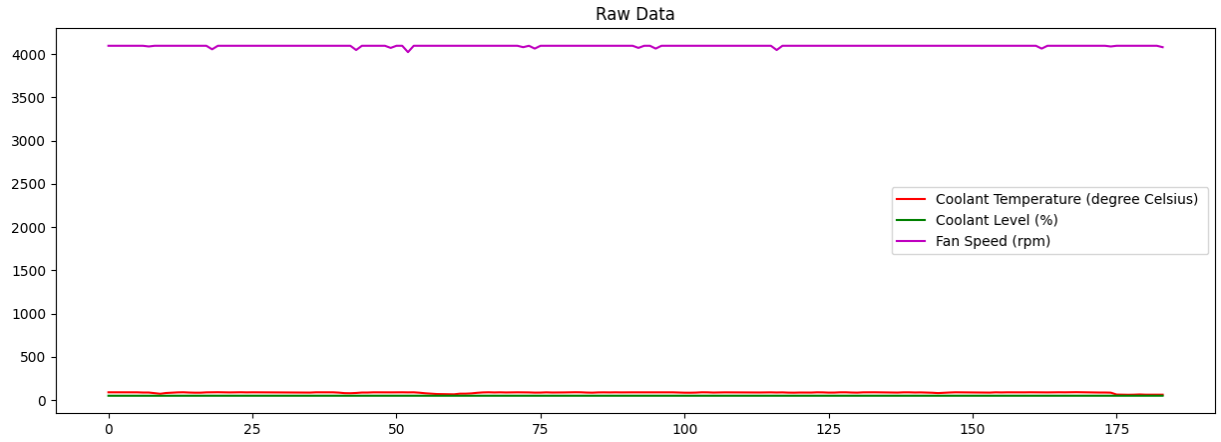

Figure 4.4: All real cooling system sensor readings together

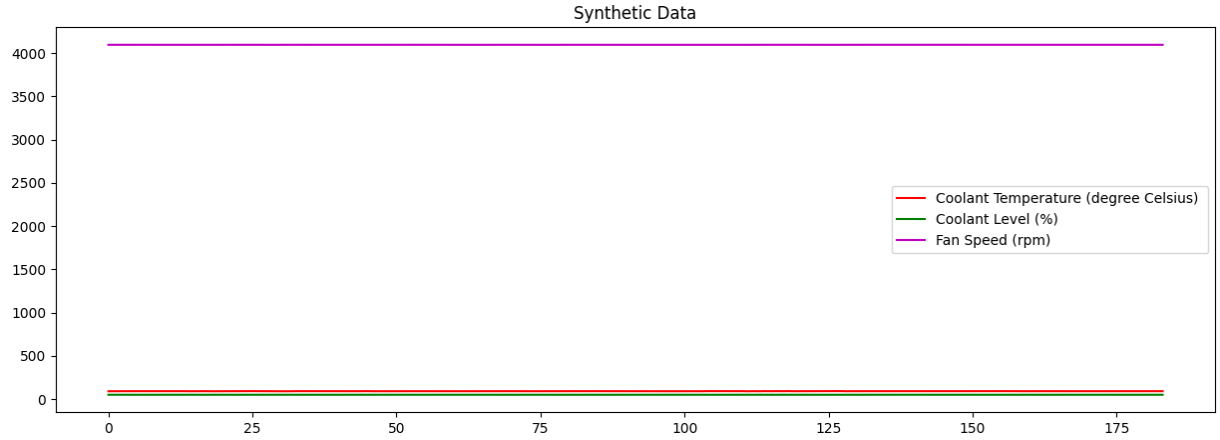

Figure 4.5: All synthetic cooling system sensor readings together

The main reason behind using the generative model to generate synthetic data was to get as close to the distributions and trends as possible. Although many temporal relations have not been captured, the generated data provides some useful insights such as:

- There was no relation found between the fan turning off and the coolant temperature and levels.
- The coolant temperature's normal working temperature is around 90 degrees Celsius and the temperature dips needed an explanation because the bus was running for almost the whole duration.
- The coolant level values included 0% but this should be disregarded and filtered out.

The aforementioned insights will be explained in Subsection 4.1.2 because they are relevant to simulating the trends that were not captured by the generative model.

4.1.2 Manual Trend Introduction

In addition to the uncaptured trends, the synthetic data generated using the DGAN model contained a lot of fluctuations from one reading to the next, which made the synthetic data appear very noisy. The problem of noise is tackled in the data preprocessing step. To make the synthetic data more realistic, the temporal trends mentioned earlier that were not captured by the generative model needed to be manually introduced. An initial step made sure that all the data generated was within the permissible data ranges of the particular sensors, that is, the synthetic sensor readings could not have a value that is impossible to record, so all the data were clipped between the minimum and maximum values allowed for each sensor.
All generated buses had a coolant level of 50%, which is not true in real life, therefore, the coolant levels of the buses were changed based on probability - there was a 30% chance that the buses' coolant level would be increased by a value between 5% and 15% (resulting

in the final values being between 55% and 65%) and a 15% chance that the coolant level would be increased by a value between 15% and 30% so the resulting coolant levels would be between 65% and 80%.
The coolant temperature dips that existed in the original data were missing in the synthetic data. The steep temperature dips in the real data from 90 degrees Celsius to approximately 60 degrees Celsius were justified by a few reasons: the engine speed was observed for the same timestamps as the temperature dips and a rotations-per-minute of zero during those timestamps suggested that the engine was not running whenever there was a large temperature dip. The sensor readings were recorded in the month of November when it is usually cold in Canada. The dip in temperature followed by a rise took place over thirty minutes - these reasons imply that the bus was not running (turned off) for a period of approximately twenty minutes which resulted in the engine cooling down faster due to the cold weather. The introduction of temperature dips was based on the assumptions that:

- A bus that was thought to be in perfect condition would have one break causing a significant temperature dip.
- A significant dip could be caused only in the cold months and not during summer because it would take longer than twenty minutes for the engine to cool down and consequently the coolant.
- Buses would not be turned off immediately after turning them back on (no consecutive temperature dips).
- A bus would not have a temperature dip in the last twenty percent of its up-time due to a break.

The dip sizes were either small (indicating that the bus was stationary for a small period of time in the cold), medium (indicating that the bus halted for a few minutes or was turned off for approximately five minutes, or large (indicating that the bus was turned off for a break). The sizes of the dips were determined by the number of time steps for which the temperature will keep decreasing and an appropriate number of time steps for it to come back to the functioning temperature. Furthermore, the temperature decline was not consistent or uniform; there was a very small probability that the temperature would go up by one degree, a higher probability that it would go down by one degree, and a significantly high probability that it would remain the same; this resulted in an inconsistent drop, which is how sensor recordings appear in real data. However, the temperature was not allowed to go below or above a set threshold based on acceptable coolant temperature values. Similarly, the temperature increase was not uniform and the probabilities of the temperature increasing, decreasing, or remaining the same were distributed as mentioned earlier but over fewer time steps. Figure 4.6 contains actual coolant temperature readings from the real bus and Figure 4.7 contains synthetic coolant temperature readings for a simulated bus after the introduction of trends.

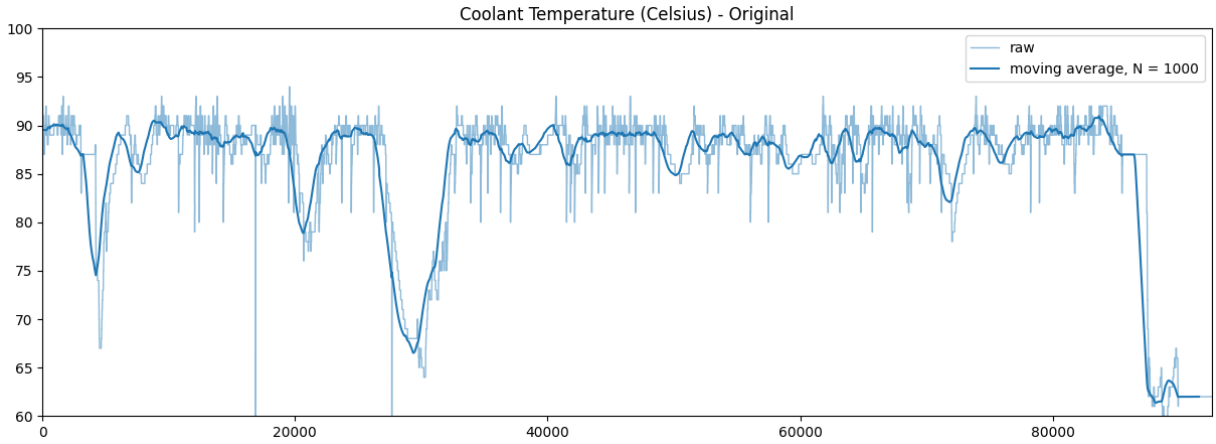

Figure 4.6: Real coolant temperature

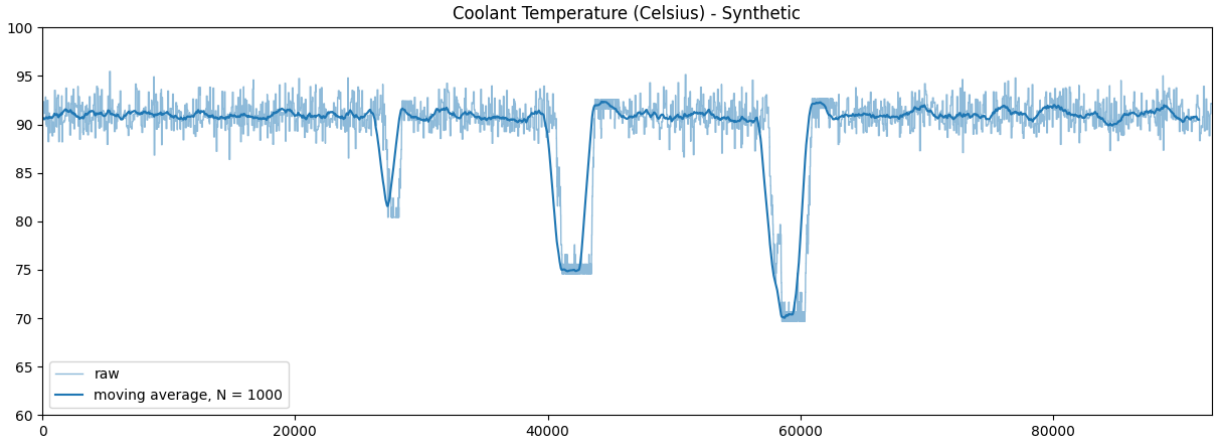

Figure 4.7: Synthetic coolant temperature with trends introduced

The rotations-per-minute (rpm) at which the radiator fan operated was found to be 4096 rpm. Firstly, all the values that were not 0 rpm were made equal to the acceptable operation value - 4096 rpm. Secondly, as mentioned earlier, the generative model failed to capture the trend of the fan turning off (the rotations-per-minute being zero). An effort was made to find out the threshold behind the fan turning off by comparing the coolant temperature and fan speed at the same time steps, but there was no relation found between the coolant temperature decreasing and the fan turning off. The fan speed was also compared to the engine speed data, but the recorded speed data consisted of just noise, so no relation could be established apart from the readings with a value of 0 rpm, which implied that the engine was turned off. Therefore, the fan speed was set to zero for a few readings during the temperature drop, which seemed to be the most justifiable approach instead of randomly introducing fan turn-offs across the time series. Figure 4.8 contains actual fan speed readings from the real bus and Figure 4.9 contains synthetic fan speed readings for a simulated bus after the introduction of trends. Note that the y-axis limits in Figure

4.9 and Figure 4.8 are set between 4040 rpm and 4100 rpm for better visualization of the moving average; the vertical lines indicate readings of 0 rpm and that the fan was turned off.

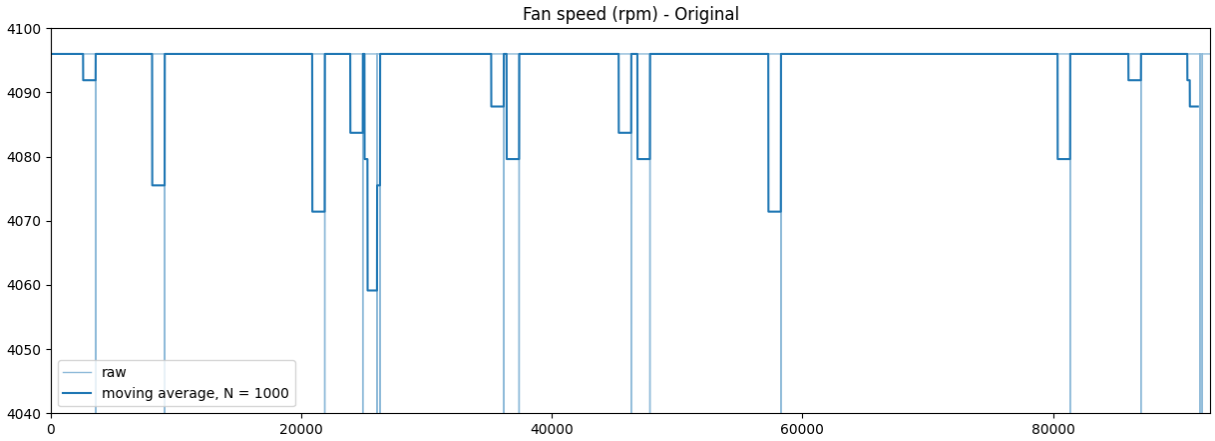

Figure 4.8: Real fan speed

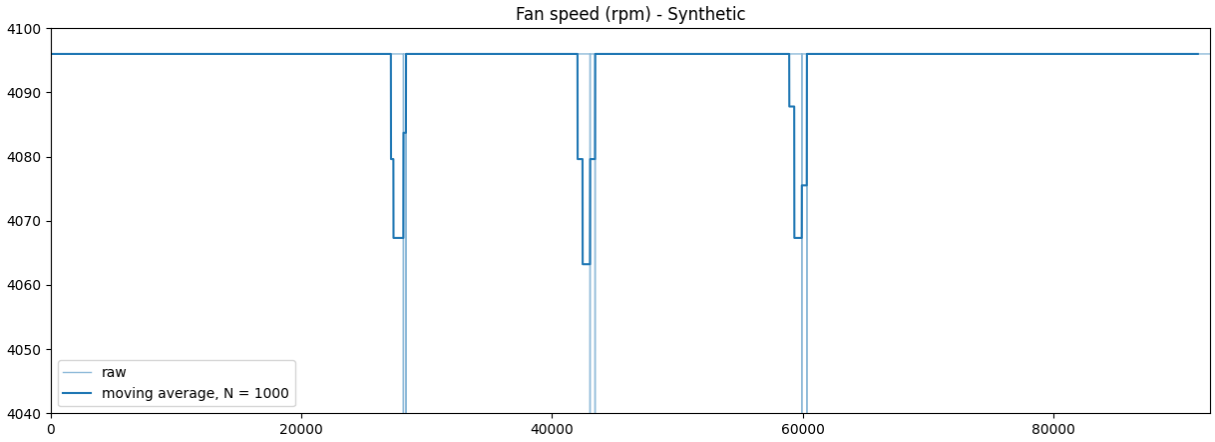

Figure 4.9: Synthetic fan speed with trends introduced

The real data and the generated data with the trends introduced do not appear to look exactly the same, but it still seems like a step in the correct direction to generate good-quality simulations as opposed to previous works in which synthetic data was generated by simply fitting the real data to a distribution and sampling from it such as in [66]. Moreover, all the simulated buses have unique trends depending on their coolant levels, temperature dip sizes and frequencies, and fan speed patterns. All the trends were introduced based on probability so there are also some instances in which there are no temperature drops, which implies that the bus was operating in summer and was running for the whole duration, consequently, the fan was not turned off for that bus up-time indicating that the fan ran throughout the duration of the bus' journey. The real bus data had only fifty recordings for which the fan was turned off, so the fan turning off was not a very important trend. The

trends introduced cover many possibilities and scenarios, so if the synthetic data is used in the training of a model due to the unavailability of real data, the synthetic data will enable the model to be robust against seasonal and operational factors (that are acceptable and normal) without those factors being interpreted as anomalies.

4.1.3 Gradual Fault Introduction

After the trends were introduced to the generated synthetic data, the sensor data of 40 buses was simulated. However, the synthetic data so far was assumed to be of normal buses, that is, without any faults. Therefore, the next step was to introduce faults to the existing synthetic data. The faults introduced were gradual and subtle, in essence, they were not abrupt or large enough to trigger caution warnings, failures, or breakdowns in buses because those faults can be easily identified using a threshold-based approach. The assumption for fault introduction was that 20% of the buses in the dataset had potential faults; this assumption was made for simplicity to avoid imbalance problems due to the lack of large volumes of data required to tackle such a problem. Hence, eight buses had potential faults in them. The faults introduced were based on heuristics, common sense, and knowledge base. There were four types of faults introduced: the first fault involves lowering the coolant level of two buses by 10% and 15%, respectively, to indicate a potential leakage in the cooling system. The reduction of the coolant level was linear and over the duration of the buses' up-time, indicating that the coolant was leaking throughout the buses' trips. Note that the reduction was based on 15% of the initial value and not 15% of the entire coolant capacity. Figure 4.10 shows the normal coolant level readings that were later modified to introduce the first fault. On the other hand, Figure 4.11 shows the coolant level readings containing the first type of gradual fault.

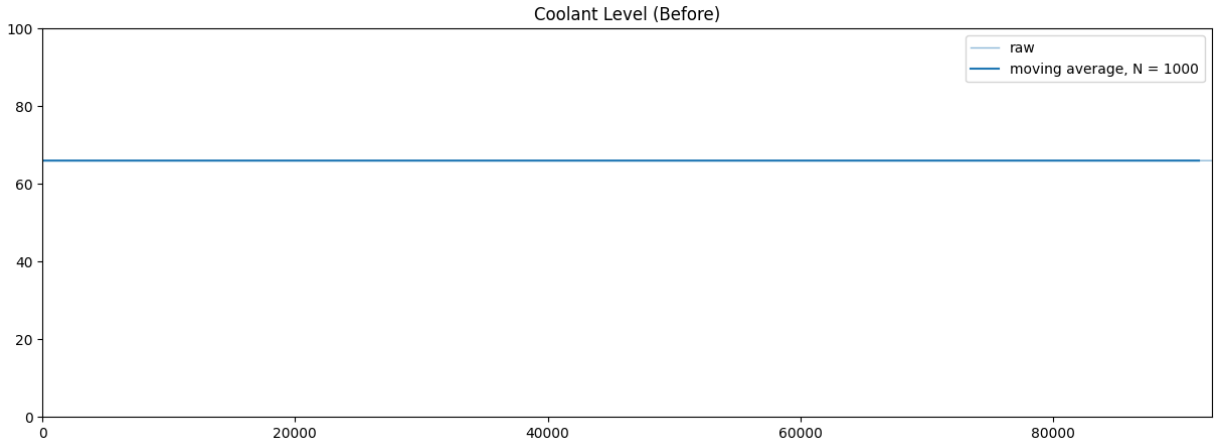

Figure 4.10: Coolant level before introducing the first fault

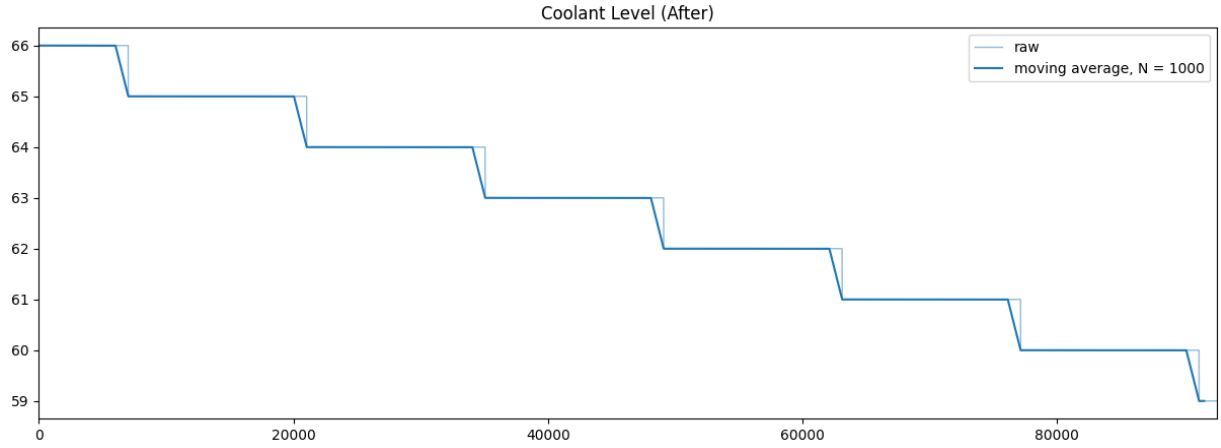

Figure 4.11: Coolant level after introducing the first fault

The second fault involves lowering the rpm at which point the fan of two buses operated by 30% and 35%, respectively, and increasing the coolant temperature slightly (by 5% and 7%, respectively) to indicate a fan performance deterioration. The decrease in the rpm was uniform and for all the values that were not 0% (non-zero values). On the other hand, the increase in the coolant temperature was linear and over the duration of the buses' up-time just like the coolant level decrease. Figure 4.12 shows the normal fan speed readings and Figure 4.13 shows the normal coolant temperature readings that were later modified to introduce the second type of fault. On the other hand, Figure 4.14 shows the fan speed readings and Figure 4.15 shows the coolant temperature readings containing the second type of gradual fault.

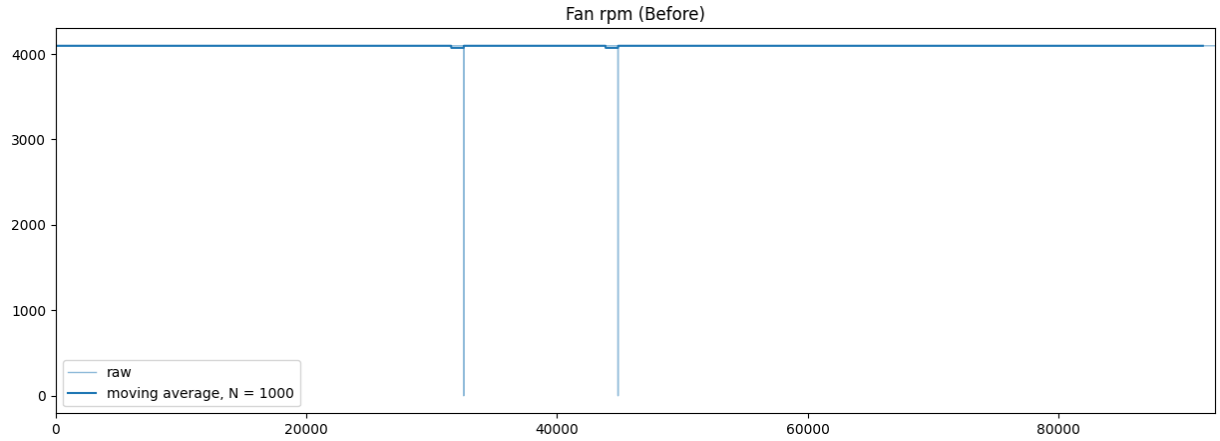

Figure 4.12: Fan speed before introducing the second fault

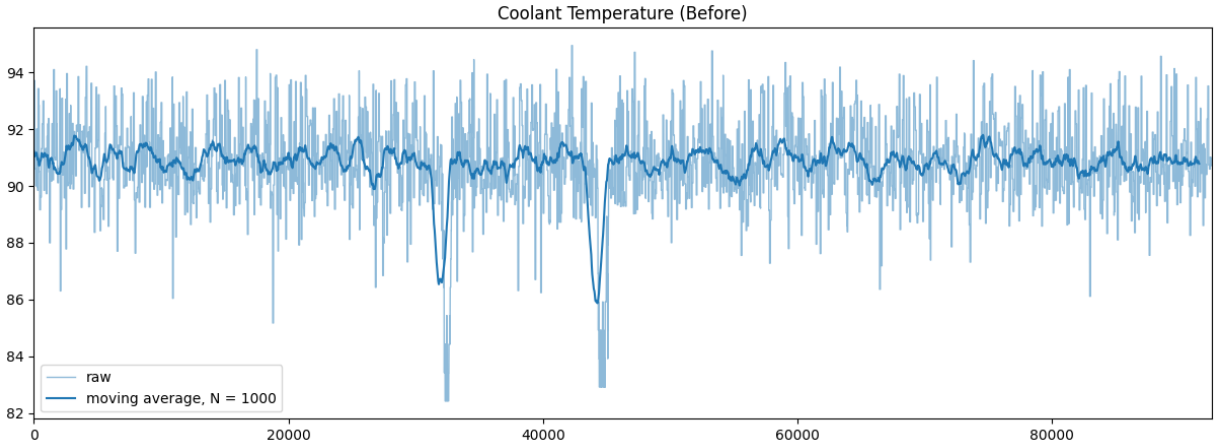

Figure 4.13: Coolant temperature before introducing the second fault

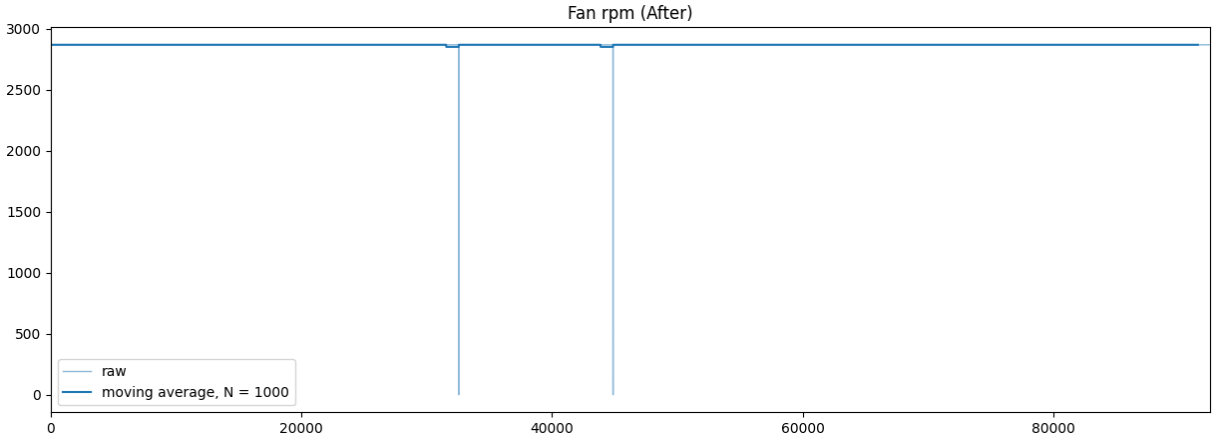

Figure 4.14: Fan speed after introducing the second fault

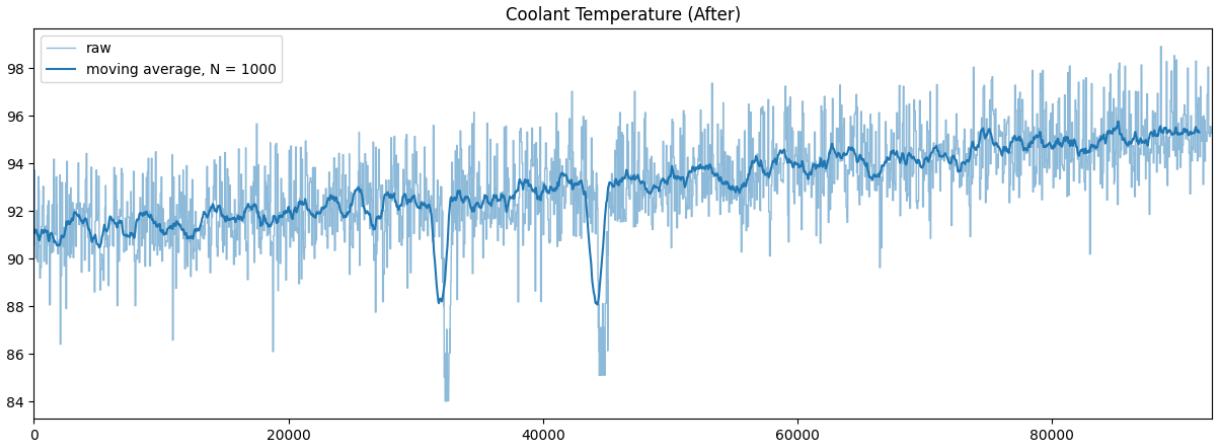

Figure 4.15: Coolant temperature after introducing the second fault

The third fault involves increasing the fan rpm of two buses by 10% and 15%, respectively, to indicate that the fan is rotating above the rated rpm of 4096 due to faulty wiring, a firmware glitch, or a similar issue, and it could cause wear-and-tear faster and lead to failure. Figure 4.16 shows the normal fan speed readings that were later modified to introduce the third type of fault. On the other hand, Figure 4.17 shows the fan speed readings containing the third type of gradual fault.

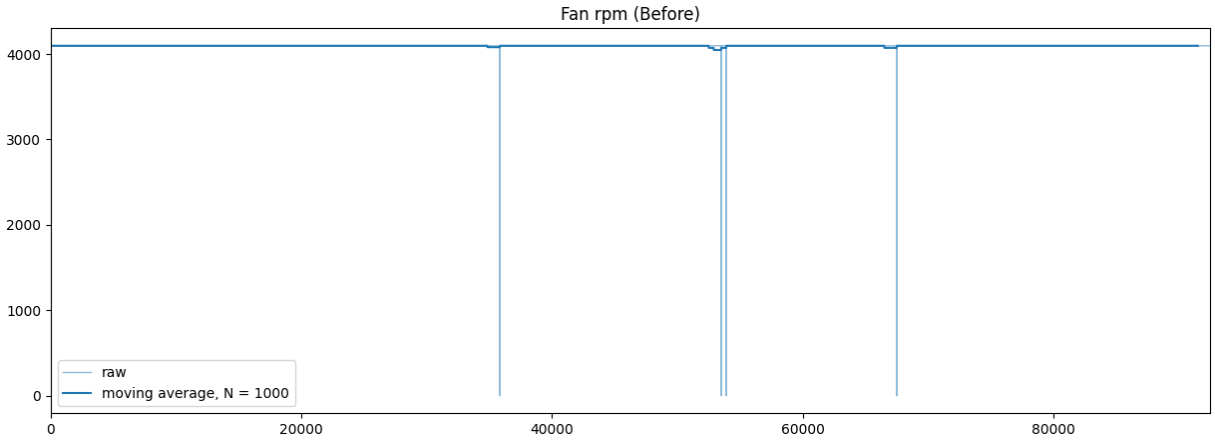

Figure 4.16: Fan speed before introducing the third fault

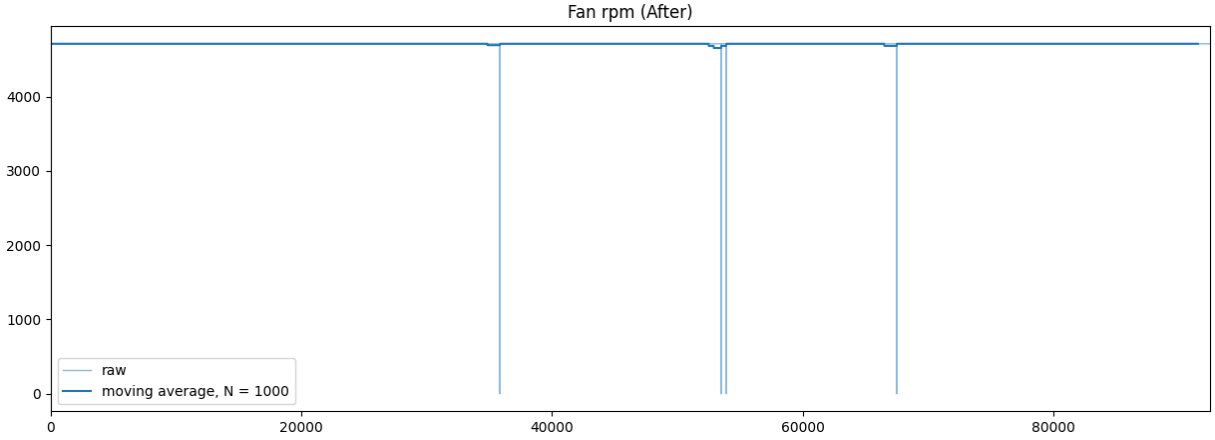

Figure 4.17: Fan speed after introducing the third fault

Lastly, the fourth fault involves increasing the coolant temperature by 10% and 15%, respectively, to indicate a cooling system issue that cannot be caused by the coolant level or fan performance. The purpose of this fault is to detect the temperature change that could have been caused by other cooling components for which the data could not be used in this work. Figure 4.18 shows the normal coolant temperature readings that were later modified to introduce the fourth gradual fault. On the other hand, Figure 4.19 shows the coolant temperature readings containing the fourth type of gradual fault.

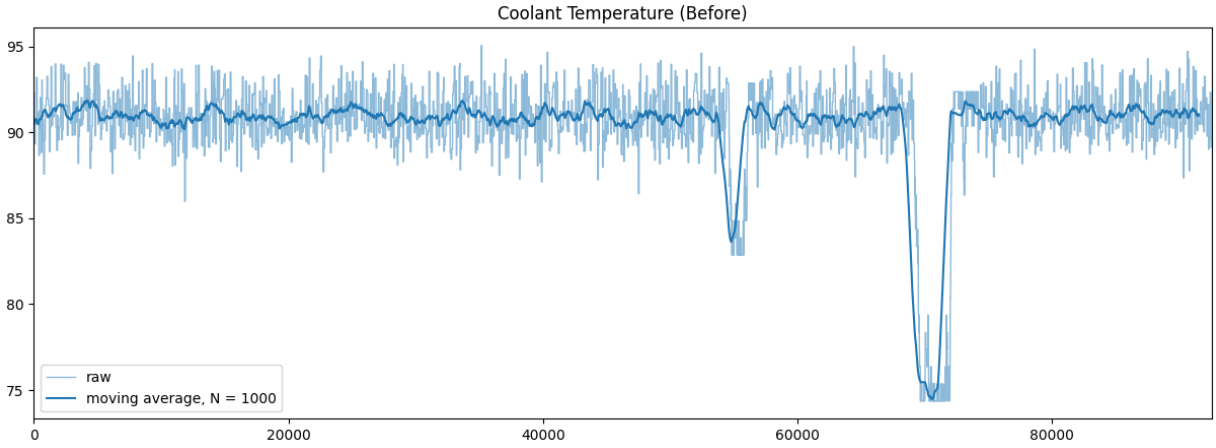

Figure 4.18: Coolant temperature before introducing the fourth fault

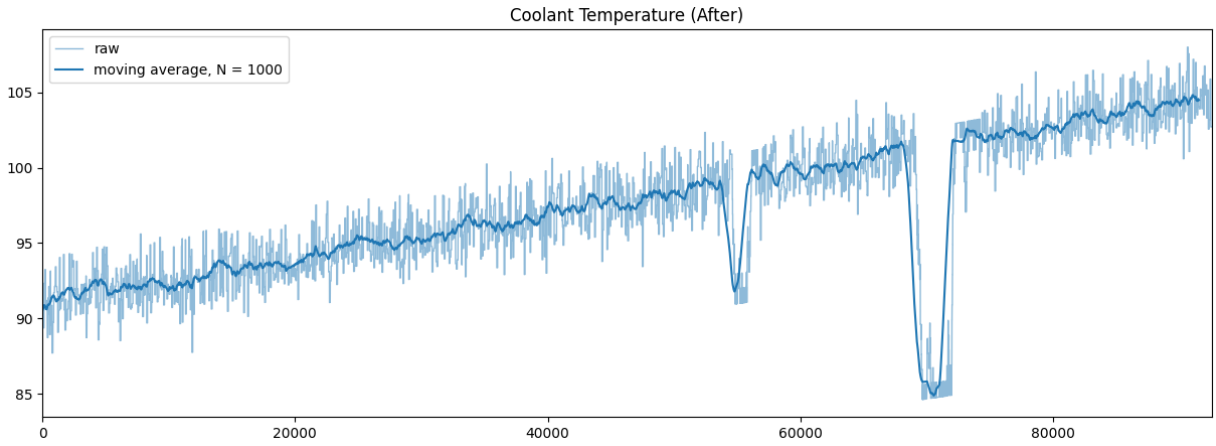

Figure 4.19: Coolant temperature after introducing the fourth fault

4.2 Engine Torque System Data

4.2.1 Data Processing and Generation

Three sensors were chosen for the engine torque system - Accelerator Pedal Position 1, Driver's Demand Engine - Percent Torque, and Actual Engine - Percent Torque as these were the only sensor readings recorded that had meaningful and accurate values. The sensor readings were first decoded from J1939 packets using the J1939 specification document and stored in a CSV file. The CSV file contained a timestamp associated with every reading along with the sensor's identifier (PGN-SPN). The data had to be restructured and reshaped for training the DGAN model, similar to the case with the cooling system data. The reshaping step included creating a dataset with each sensor as a column - [Accelerator Pedal Position, Driver's Demand Torque, Actual Engine Torque] - and each

column had the sensor readings for its respective sensor for every timestamp. Doing so eliminated all categorical variables and resulted in a dataset with only continuous values. As mentioned in Subsection 4.1.1, the rows in the dataset had only one reading per row and the other two columns had null values. This was caused because sensor readings were appended to the dataset one by one to preserve the order of the sensor readings recorded with respect to the timestamp. Additionally, the null values were filled using the forward fill method as mentioned earlier so as to preserve the trends in the time series. The final reshaped and processed dataset contained three columns, one for each sensor, and the columns contained sensor readings for their respective sensor.

The final dataset was used as the input to the DGAN model for training and generating synthetic data. As mentioned in Subsection 4.1.1, several hyperparameters needed to be tried and tested to generate similar patterns in the synthetic data. In comparison to the cooling system generation hyperparameters, the hyperparameters for the engine torque system required larger sequence lengths and batch sizes, and the structure of the generative network remained quite similar. The synthetic data generated from the best possible tried configuration of the generative model presented some similarities in trends to the original data, however, most synthetically simulated buses had unrealistic distributions and patterns. The following figures provide a comparison between the moving averages (window size 500) of the original sensor values and the synthetic sensor values generated using the DGAN model: a) Figure 4.20 provides a comparison between the real and synthetic accelerator pedal position values for a bus, b) Figure 4.21 provides a comparison between the real and synthetic driver's demand torque values for a bus, and c) Figure 4.22 provides a comparison between the real and synthetic actual engine torque values for a bus. The figures display high dependence and similarity among the real sensor values. A similar trend can be observed in the similarity between the synthetically generated actual engine torque values (Figure 4.22) and the synthetically generated driver's demand torque values (Figure 4.21). However, the synthetically generated accelerator pedal position values look entirely different from the previous two which is inaccurate and unrealistic. The trend of dependency among the three sensor readings is replicated but only to a certain degree which makes it unrealistic. Furthermore, the values generated by the DGAN model contain a lot of variance and fluctuations between consecutive readings, this does not hold true with the real data. The real data had most values concentrated around the mean and also many time periods when the accelerator was not pressed that is, at 0%. The pattern of the accelerator pedal position and its dependent sensors being at 0% was also absent in the synthetic data.

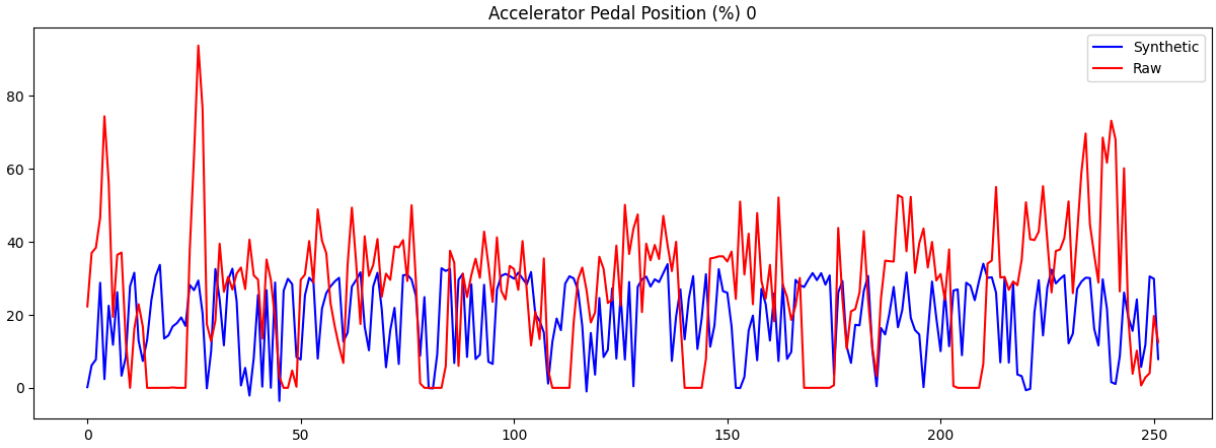

Figure 4.20: Comparison between real and synthetic accelerator pedal position values

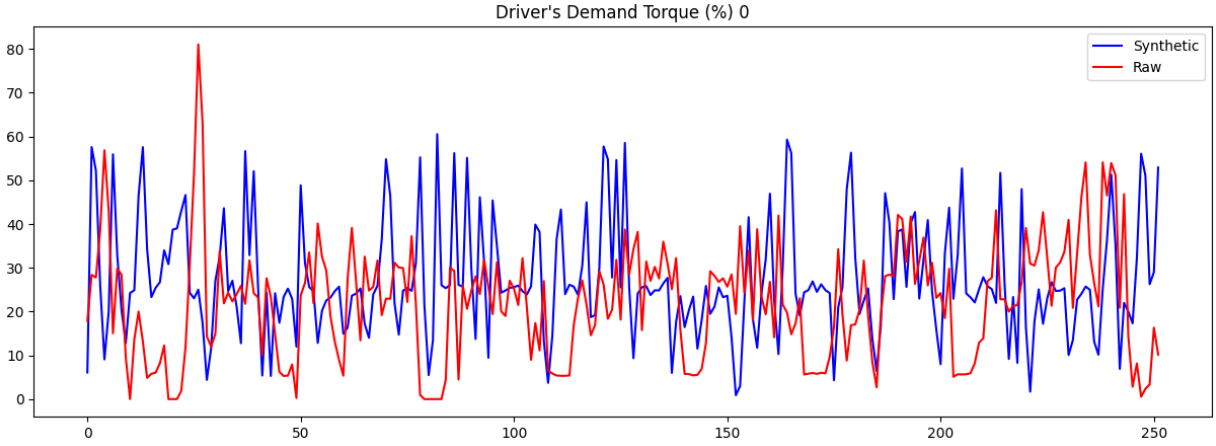

Figure 4.21: Comparison between real and synthetic driver's demand torque values

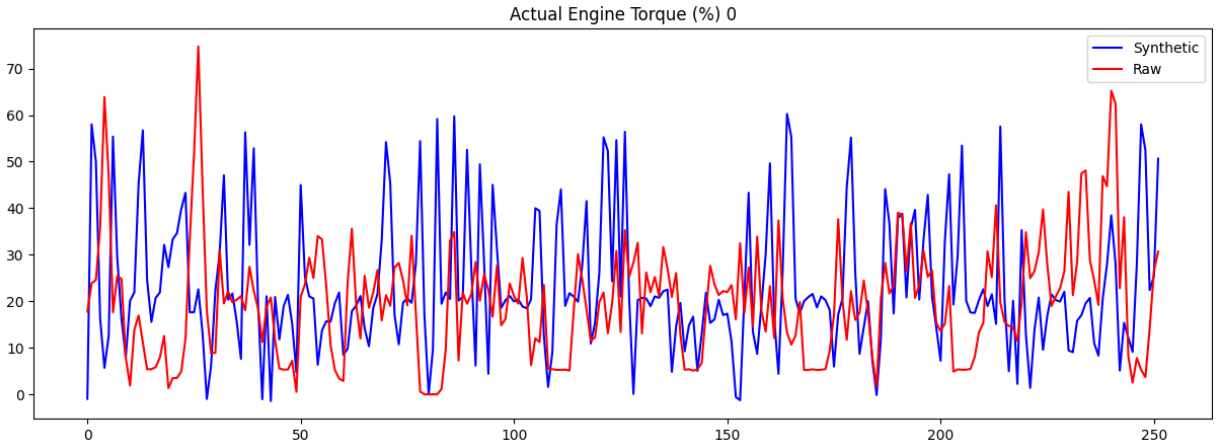

Figure 4.22: Comparison between real and synthetic engine torque values

The real data from the engine torque system was very rough along with a significant degree of randomness. This could be justified by a driver's behavior in pushing the accelerator pedal influencing all three sensors. Furthermore, the sensor readings were taken from a public bus that traveled around the city which would require constantly pressing and releasing the accelerator pedal with varying amounts of force. Additionally, there was a high dependence among the three sensor readings, that is, the driver's demand torque directly depended on the accelerator pedal position, and the actual engine torque directly depended on the driver's demand torque. Figure 4.23 contains three line plots with real values (moving average) from the three sensors; the high dependence mentioned earlier is visible in the figure with an overall (not strict) relation among the three sensor values as:

$$AcceleratorPedalPosition \geq DriverDemandTorque \geq EngineTorque$$

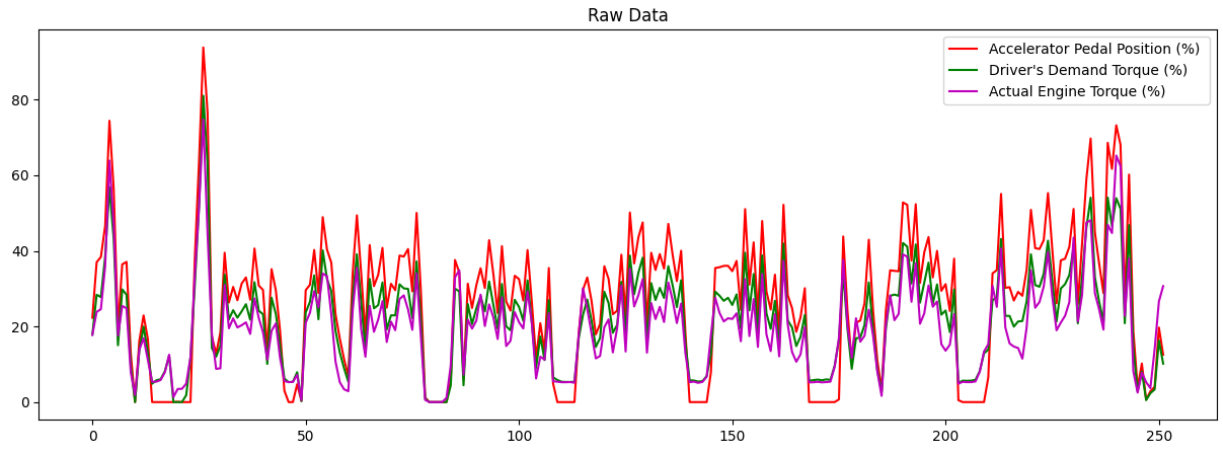

Figure 4.23: All three real sensor values plot

On the other hand, Figure 4.24 contains three line plots with synthetic values (moving average) from the three sensors; the overall relation between the three sensors mentioned earlier does not hold true consistently in the synthetic data making it unrealistic.

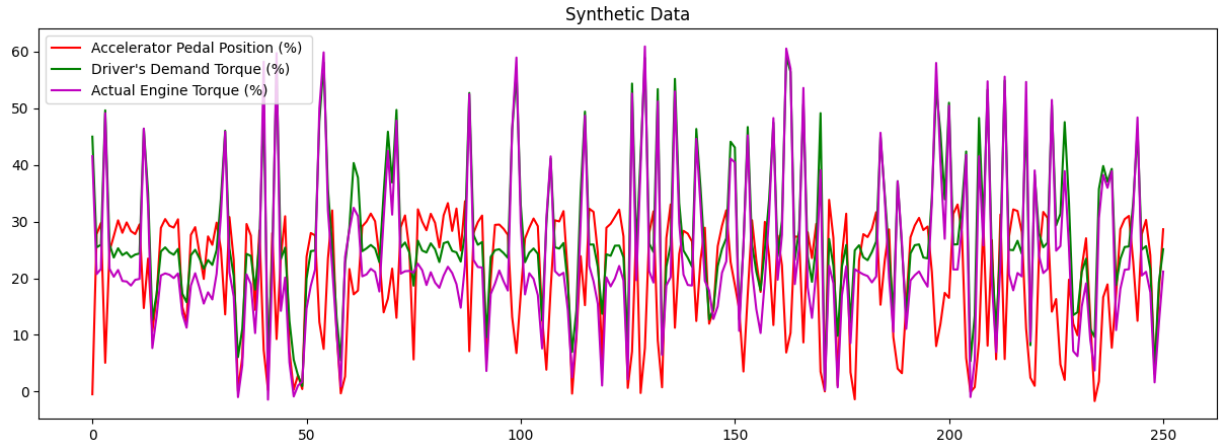

Figure 4.24: All three synthetic sensor values plot

In addition to the dependence among the three sensor values, the distribution of the data also does not appear to be realistic with constant irregular spikes in the values. A better visualization of the inaccurate distribution can be visible in the difference between Figure 4.25 and Figure 4.28. Figure 4.25 contains the synthetic raw accelerator pedal position sensor values along with the moving average and Figure 4.28 contains the original raw accelerator pedal position sensor values along with the moving average.

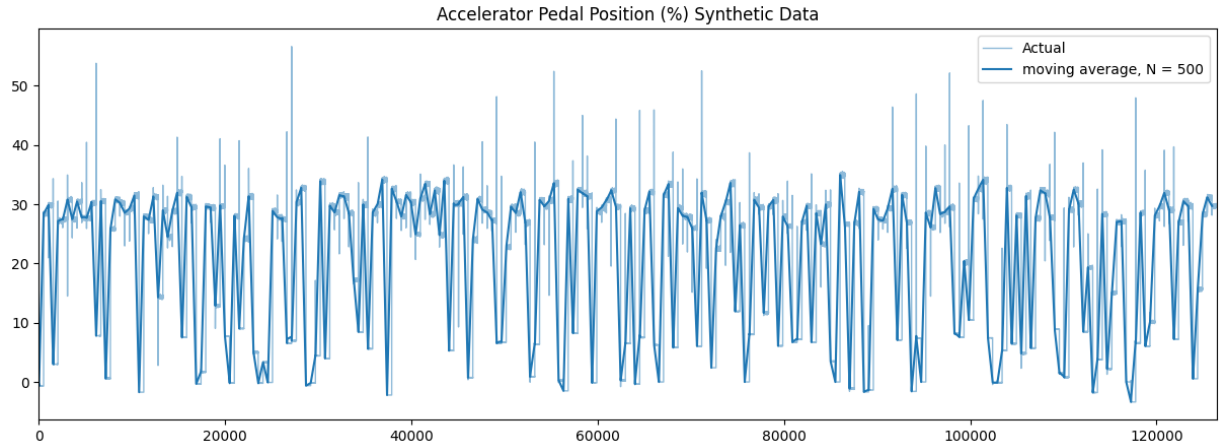

Figure 4.25: Synthetic accelerator pedal position values with moving average

In summary, the synthetic data generated using the DGAN model contained various inaccuracies making it unrealistic and unfit for predictive maintenance model training. The aim of using a generative model for synthetic data generation is to get as close to the distributions of real data as possible while maintaining realistic trends along with an acceptable degree of randomness; this would result in realistic simulations of normally functioning buses. Certain dependencies, trends, and value distributions must hold in normally functioning buses and cannot be ignored, however, they could not be achieved using the generative model. Although generating synthetic data to simulate buses using the DGAN model was not the most feasible choice, a few insights received from the effort of using the generative model are:

- The real data values are highly concentrated around the mean and negatively influence model performance by obstructing their interpretation of intricate trends, therefore, this must be taken into account when preprocessing the data for training predictive maintenance models.
- Although the similarity in values among the three sensors might appear clear to the eye, using raw data values without sufficient preprocessing might result in low model performance due to the noisy and varying structure of the data.

4.2.2 Synthetic Data Generation with Trends

The attempt to generate synthetic data using the DGAN model was unsuccessful as explained in Subsection 4.2.1. Although certain similarities between the real data and the synthetic data were visible, the overall distributions were inaccurate and not suitable for training machine learning models for the task of predictive maintenance. Therefore, the process of synthetic data generation needed to be done manually from scratch. The key constraints and dependencies that the synthetic data must exhibit are:

- The relation established among the three sensors must hold true in all cases, that is, the accelerator pedal position influences the driver's demand engine torque, which in turn influences the actual engine torque.
- The presence of "bins" where the accelerator pedal position is at 0% indicating that the vehicle was halted. The "bins" mentioned are ranges for which the accelerator pedal is not pressed at all and these ranges appear throughout the time series.
- The distribution of the non-zero values is concentrated around the mean value, that is, the accelerator pedal position is most often pressed or held around its mean of 35% to 40%, and therefore, the other two sensor readings also follow the same distribution due to the chain of influence mentioned earlier.

Using this knowledge, the generation of synthetic data was done using samples from the relevant distribution, using the current sensor reading to calculate the next sensor reading (by adding or subtracting random values from the current reading), and introducing "bins" throughout the time series.
The first step involved testing a variety of distributions to find the best fit for the real data. Some of the popular distributions tested were the normal distribution, the exponential distribution, the uniform distribution, the Cauchy distribution, the Weibull distribution, and many other distributions from [116]. The metric used to evaluate the best fit of distribution is the Akaike Information Criterion (AIC). AIC uses the maximum likelihood estimation to compare the information value of models while penalizing the number of parameters used by the models (complexity of the model) [22]. The maximum likelihood estimation (or log-likelihood estimate) is the process of estimating the parameters of a model in its ability to produce the observed results [97]. The model with the lowest AIC score is considered the best fit. Equation 4.1 contains the formula for calculating AIC, where K is the number of parameters used by the model and L is the likelihood estimate.

$$AIC = 2K - 2ln(L) \tag{4.1}$$

The fitting of distributions was done with and without the 0% values to find the most accurate representation of the range of values present in the real data. The reason behind this is that simulating the "bins" and introducing 0% values throughout the time series was relatively easier than simulating the accelerator pressing behavior and the resultant torque produced. The main aim was that the final torque produced by the engine should be realistic and accurate, this would automatically ensure that the preceding sensors would

have realistic values as well due to the dependency among them.
The exponential distribution fitted using non-zero values had the most similar statistics to the real data. The probability density function of the exponential distribution can be expressed as in Equation 4.2 where λ is the rate parameter [90]:

$$F(x, \lambda) = \begin{cases} \lambda e^{-\lambda x} & if\ x > 0 \\ 0 & otherwise \end{cases} \tag{4.2}$$

The exponential distribution and the normal distribution had very similar representations of the real data containing non-zero values, however, the exponential distribution was found to be more relevant as it contains a range of positive values, and smaller values are more likely to be sampled than larger ones. Figure 4.26 contains a histogram plot of the values in the real engine torque data and it is visible that small torque percentage values have a higher frequency than high torque percentage values.

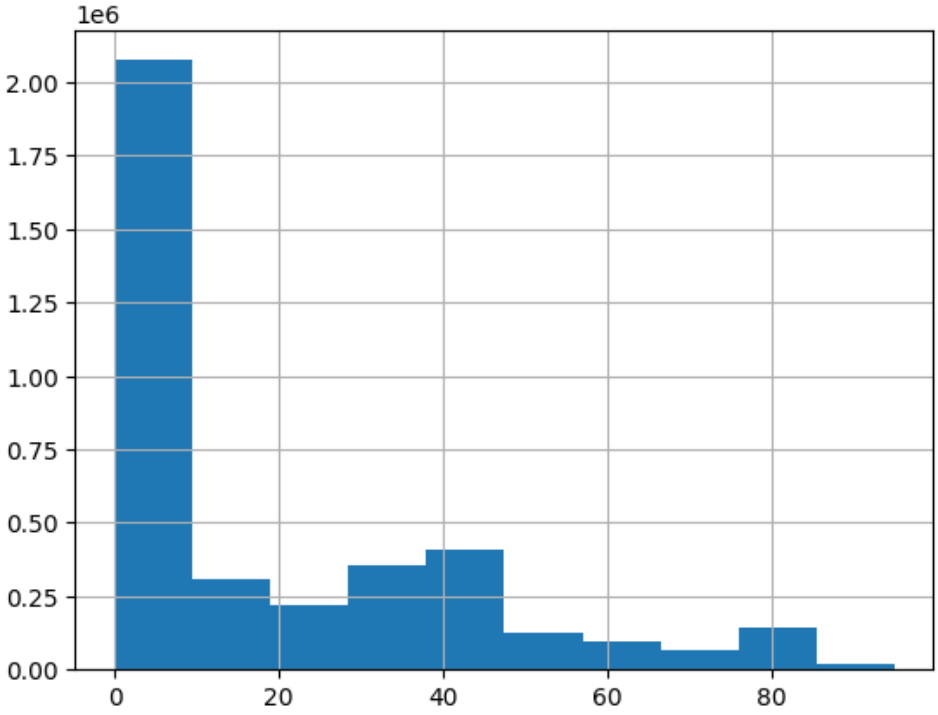

Figure 4.26: Histogram of the real engine torque values

Finding the best-fit distribution was the first step in generating synthetic data. However, randomly sampling data from the distribution is a small part of the entire synthetic data generation process. The simple random sampling of data would not produce a realistic time series that maintains all the dependencies and constraints mentioned earlier.
The idea was to introduce accelerator pedal pressing behavior throughout the time series as opposed to introducing "bins" to an existing time series, such as the one generated by the DGAN model. The reason for this is that it gives us more control over how the

non-zero values are generated so as to ensure that no constraints and dependencies are violated. Therefore, a dataset is initialized with only zeroes as a placeholder, that is, it contains only 0% values and could be considered as one large "bin".
In order to add non-zero values to the synthetic accelerator pedal position data, there is a small probability while iterating over the dataset that non-zero values will be added for a range. The reason behind the small probability is that the length of the dataset is large and choosing a small enough probability would result in the introduction of a significant amount of non-zero values (indicating the vehicle was in motion) while preserving a reasonable number of "bins" (indicating the vehicle was stationary). The range for which non-zero values will be added to the dataset is set to simulate a random time period between 20 minutes and 45 minutes based on the length of the dataset and the polling frequencies of the sensors. The first step in introducing a non-zero value is to take a random sample from the distribution created. There is a small probability that the current value being added to the dataset will be the random sample chosen. There is a large probability that the current value being added to the dataset will depend on the previous value in the dataset. Doing so will maintain temporal relationships between consecutive sensor readings in the dataset. On the other hand, using a small probability for the current value to be a random sample replicates the noisy behavior of the real data. The key ideas in using the previous reading to calculate the current value to be used as a reading are as follows:

- There is a large probability that the current reading will be greater than the previous reading by a small margin resulting in a slow upward trend.
- There is a small probability that the current reading will be smaller than the previous reading but by a larger margin resulting in a fast downward trend.

Following this pattern in generating synthetic sensor readings for the accelerator pedal position would result in a) slow increments in the pedal being pressed but over a longer period of time, and b) fast releases in the accelerator pedal over a short period of time. This approach falls in line with the usual real-world scenario where a driver would not abruptly press the accelerator pedal to 100%, on the other hand, a driver would release the accelerator pedal immediately to lower the speed or press the brake pedal. The values added to the previous readings during the upward trend to get the current readings are random values in the range of 0.4 and 2.5 with a step size of 0.1. The values deducted from the previous readings during the downward trend to get the current readings are random values in the range of 5.0 and 8.0 with a step size of 0.1. In addition to all the methods used to add non-zero values, there was still a 5% chance that the current value would be left as 0; this was to replicate the presence of noisy 0% values even during the motion of the vehicle.
The aforementioned methods are first used in the generation of synthetic sensor data for the accelerator pedal position. This is because the sensor readings of the other two sensors entirely depend on the values of the accelerator pedal position, therefore, being accurate in the synthetic generation of the accelerator pedal position data was necessary. The following figures provide a comparison between the real accelerator pedal position data and the synthetically generated accelerator pedal position data using the methods mentioned

in this subsection: a) Figure 4.27 contains a graph with two moving averages (window size = 500) comparing the real and synthetic accelerator pedal position data, b) Figure 4.28 contains a plot of the real accelerator pedal position data with a moving average (window size = 500), and c) Figure 4.29 contains a plot of the synthetically generated accelerator pedal position data with a moving average as well. The moving average window size for the engine subsystem data is smaller than that used for the cooling subsystem data for a better representation of the complex distribution of the raw data.

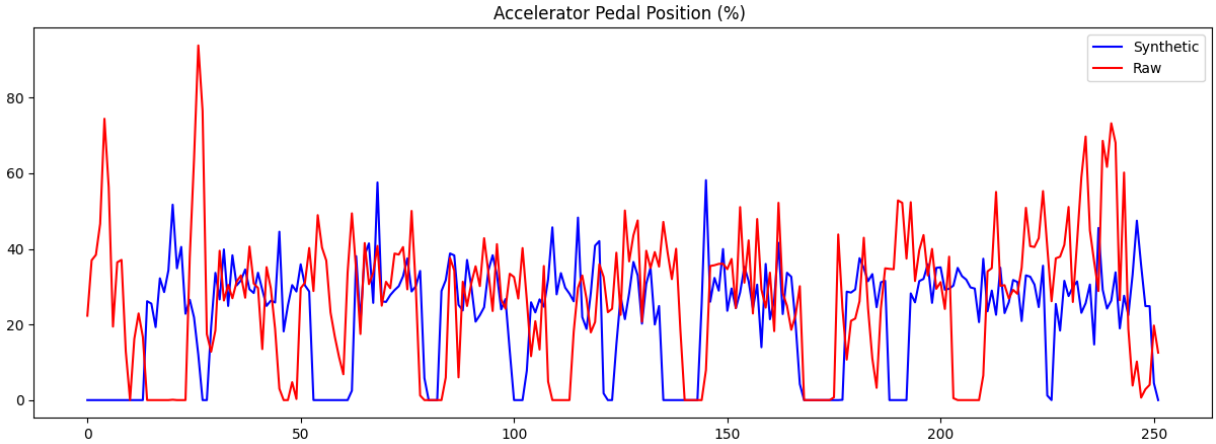

Figure 4.27: Comparison between the moving averages of real and synthetic accelerator pedal position data

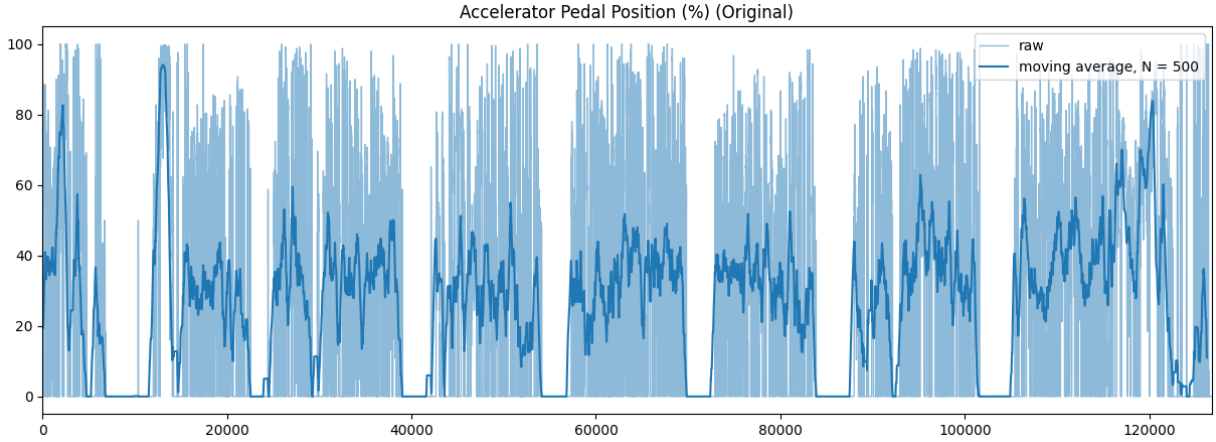

Figure 4.28: Real accelerator pedal position values with moving average

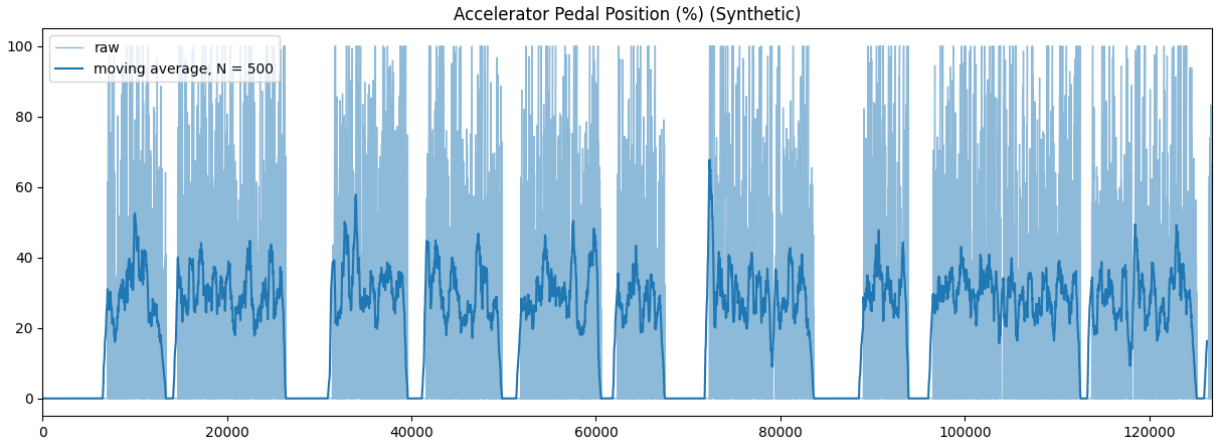

Figure 4.29: Synthetic accelerator pedal position values with moving average

The generation of synthetic driver's demand torque and actual engine torque was done based on the synthetic data of the accelerator pedal position. The accelerator pedal position's value is generally larger than the driver's demand torque's value for a given timestamp and the driver's demand torque's value is larger than the actual engine torque's value for a given timestamp as established in Subsection 4.2.1. Therefore, for a given timestamp, the driver's demand torque's value is calculated by subtracting a random value from the accelerator pedal position value. The random value is chosen from a range of 0.4 to 10.0 with a step size of 0.1. The following figures provide a comparison between the real driver's demand torque data and the synthetically generated driver's demand torque data using the methods mentioned earlier: a) Figure 4.30 contains a graph with two moving averages (window size = 500) comparing the real and synthetic driver's demand torque data, b) Figure 4.31 contains a plot of the real driver's demand torque data with a moving average (window size = 500), and c) Figure 4.32 contains a plot of the synthetically generated driver's demand torque data with a moving average as well.

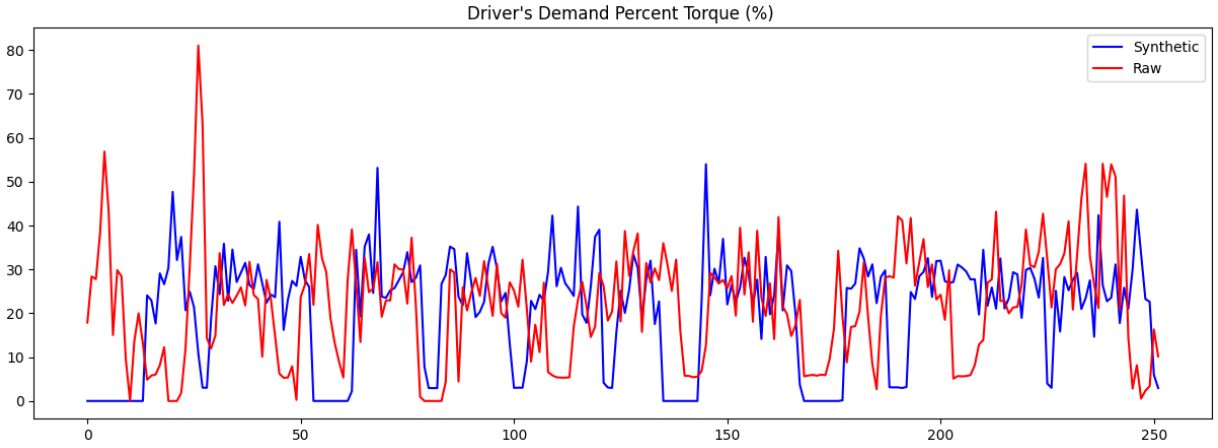

Figure 4.30: Comparison between the moving averages of real and synthetic driver's demand torque data

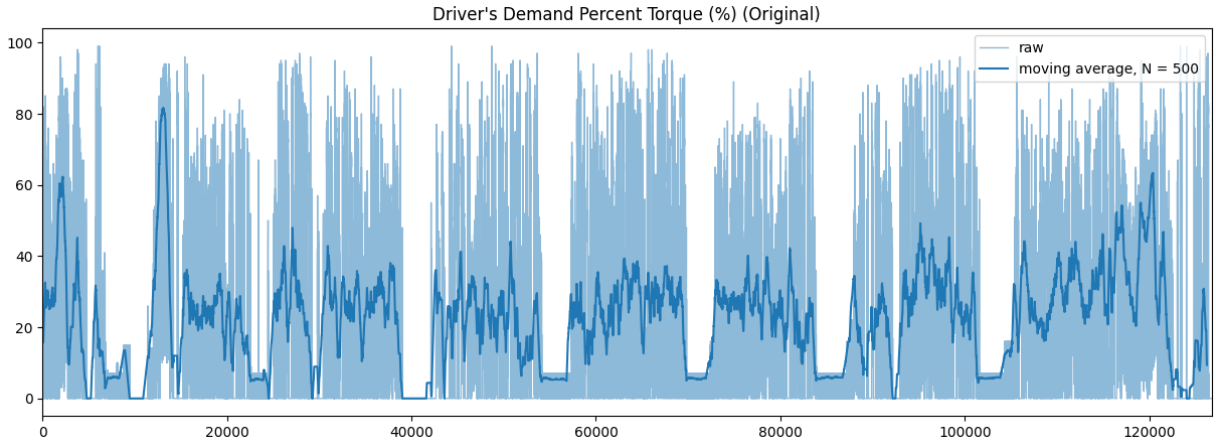

Figure 4.31: Real driver's demand torque values with moving average

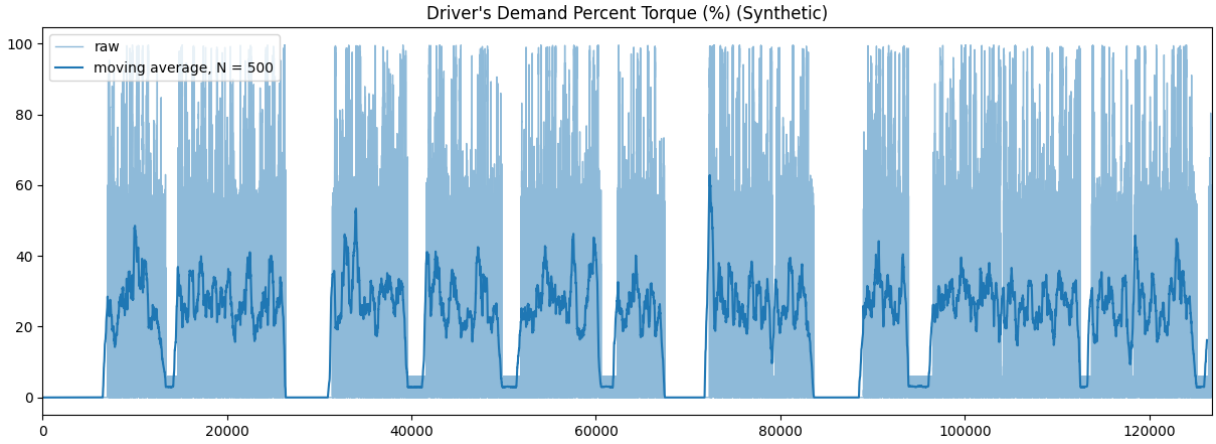

Figure 4.32: Synthetic driver's demand torque values with moving average

The actual engine torque's value is also calculated by subtracting a random value from the accelerator pedal position's value but the range of the random value is from 0.8 to 15.0 with a step size of 0.1. These ranges of random values would generally and probabilistically result in the values of the driver's demand torque being larger than the values of the actual engine torque. However, for each timestamp, there is a 5% chance that the value for either driver's demand torque or actual engine torque will be a random sample from the exponential distribution. Doing so will maintain the presence of noisy values across all the sensors that are independent of each other. The following figures provide a comparison between the real actual engine torque data and the synthetically generated actual engine torque data using the methods mentioned earlier: a) Figure 4.33 contains a graph with two moving averages (window size = 500) comparing the real and synthetic actual engine torque data, b) Figure 4.34 contains a plot of the real actual engine torque data with a moving average (window size = 500), and c) Figure 4.35 contains a plot of the synthetically generated actual engine torque data with a moving average as well.

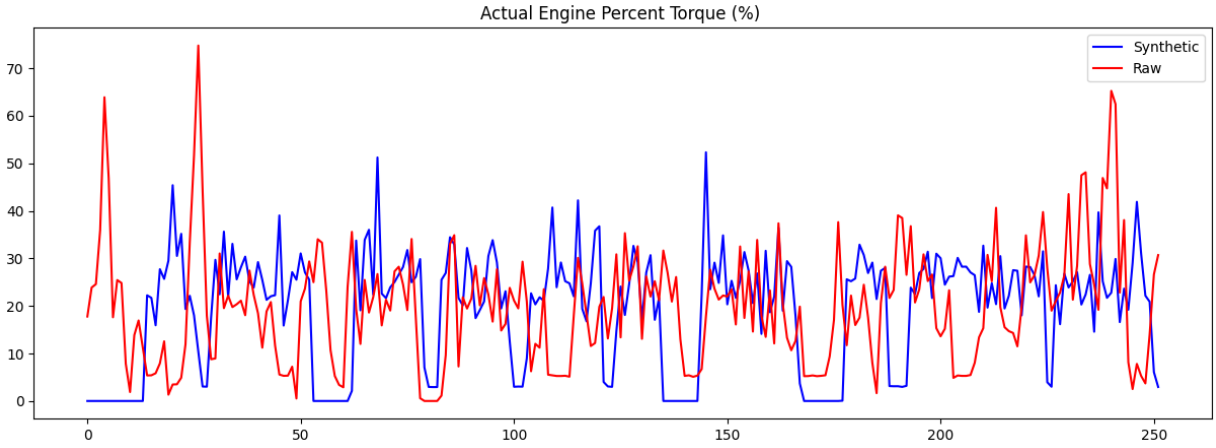

Figure 4.33: Comparison between the moving averages of real and synthetic actual engine torque data

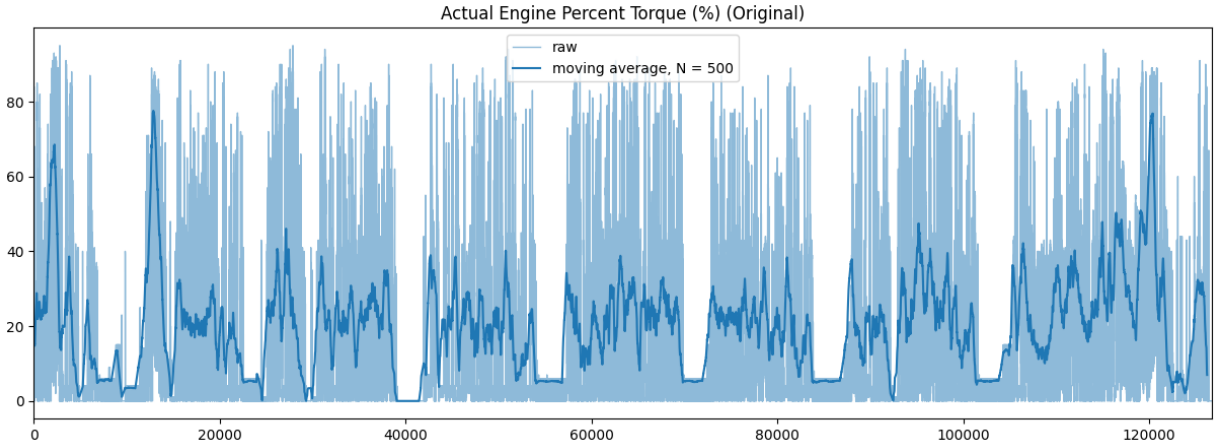

Figure 4.34: Real actual engine torque values with moving average

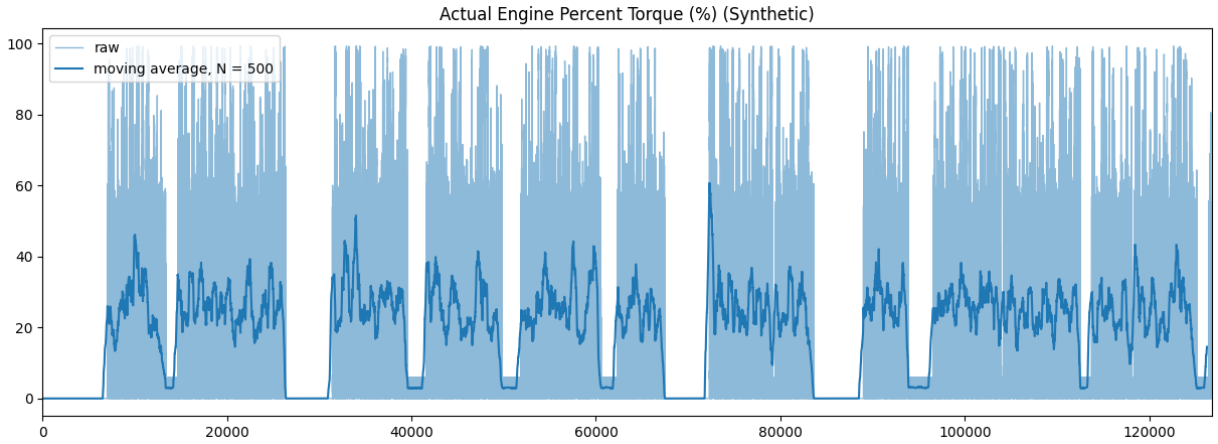

Figure 4.35: Synthetic actual engine torque values with moving average

There is another trend common in the driver's demand torque data and the actual engine torque data. When the size of a "bin" in the accelerator pedal position data is small, the corresponding readings in the driver's demand torque and the actual engine torque are non-zero. This indicates that the vehicle was not turned off, the engine was required to produce torque for miscellaneous reasons such as cruise control options, air conditioning, etc. The driver's demand torque values are also non-zero for either the previously mentioned reasons or some other mode setup. This trend was also introduced to the synthetic data by measuring the interval between two non-zero value addition phases (or a "bin") and adding small random values to the two sensor's data if the interval was small enough. The interval was set by estimating a range ("bin" size) that corresponds to 15 minutes. The methodology to generate synthetic engine torque system data described in this subsection appears to generate reasonably accurate and realistic synthetic data. This is visible in the figures that compare the real and synthetic data. Despite the generative model not being used, the manual approach to generate synthetic data based on probabilities and data analytics seems to be comprehensive and robust as the formation of the methodology required a rigorous amount of trial and error.

4.2.3 Gradual Fault Introduction

The engine torque system synthetic data generation process along with trends resulted in 38 datasets, each representing a normal functioning bus. The data generation process was based on the data from the real bus which is assumed to be functioning normally and without any faults. Hence, the next step is to introduce gradual faults to some of the simulated buses. Gradual faults were introduced to 6 out of 38 buses, thereby resulting in approximately 15% of the buses possessing underlying faults. The assumption for the number of buses chosen was similar to the one in the cooling system dataset, however, faults related to the engine performance system are slightly less likely compared to the faults in the cooling system. Additionally, lowering the percentage of faulty buses from 20% (as in the cooling system) to 15% provides an additional testing case for the proposed

algorithm in Section 3.1, which is almost an imbalance problem. Achieving accurate results in a situation containing imbalance would validate the algorithm's use in real-world situations where most of the buses function normally and only a few may contain potential faults.

The nature of the faults introduced is gradual, that is, the magnitude of the impact of the faults is small and may not be noticed by the driver or existing detection mechanisms until the fault grows to result in significant performance issues or component failure. Also, the faults are introduced by observing patterns in the original bus data and adding modifications to them by assuming that normally functioning buses would exhibit similar trends and patterns as the original bus. The faults introduced to the engine torque system primarily indicate that the bus's engine system is unable to deliver its full amount of torque or potential. Moreover, all the faults result in lower actual engine torque being produced as a fault in the accelerator pedal or the driver's demand would directly influence the final engine torque produced. Three types of gradual faults are introduced, and each type of fault is introduced to two buses.

The first type of fault involves lowering the driver's demand engine torque by approximately 13% and lowering the actual engine torque by approximately 14% throughout the time series. This fault indicates that the driver's demand engine torque system may be misreading the pedal position, or may be miscalculating the amount of torque to deliver due to malfunctioning sensors, faulty firmware, inefficient engine loads, and so on.

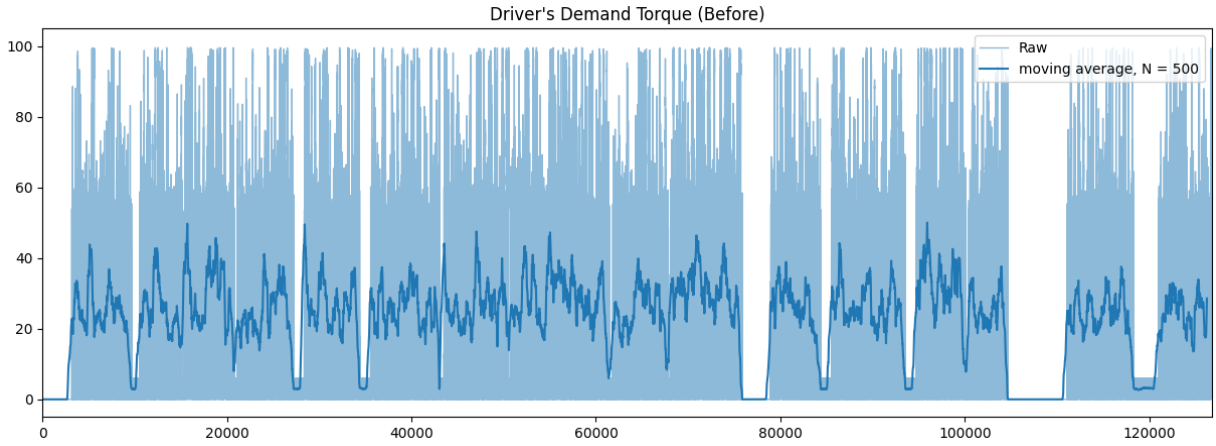

Figure 4.36: Driver's demand engine torque before introducing the first fault

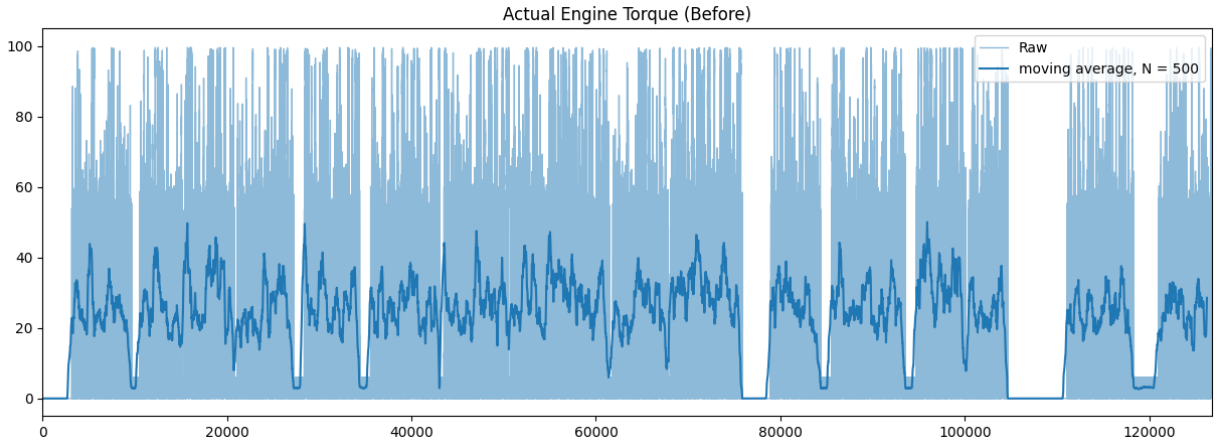

Figure 4.37: Actual engine torque before introducing the first fault

Figure 4.38 contains a plot of the driver's demand engine torque after faults were introduced to its readings and Figure 4.39 contains a plot of the actual engine torque readings after the first fault was introduced to it. Figure 4.36 and Figure 4.37 contain plots of the driver's demand engine torque and the actual engine torque readings before introducing faults to them for comparison.

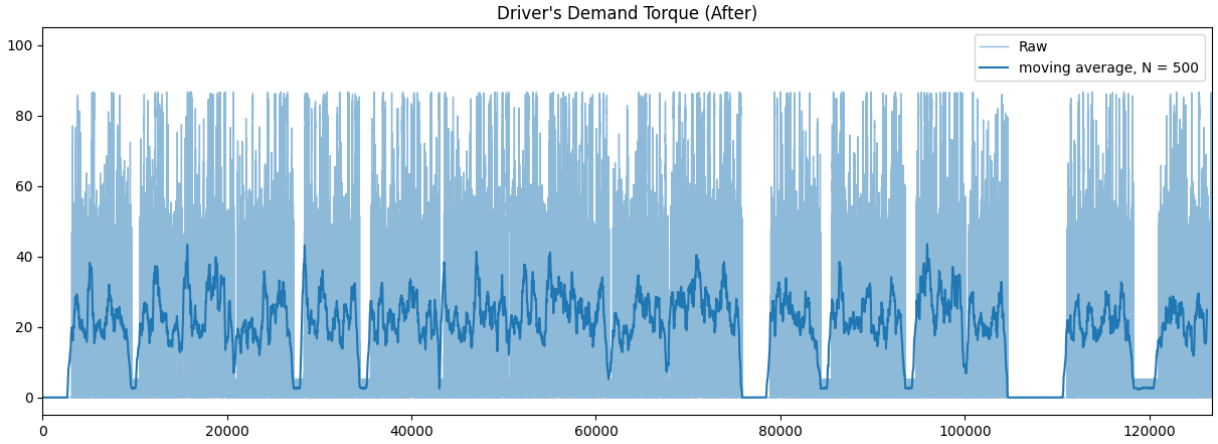

Figure 4.38: Driver's demand engine torque after introducing the first fault

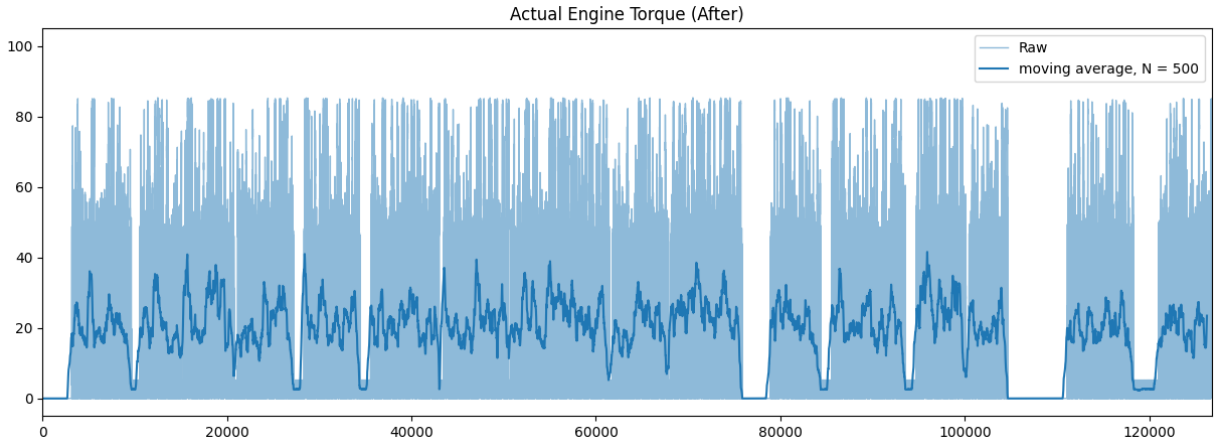

Figure 4.39: Actual engine torque after introducing the first fault

The second fault involves lowering the actual engine torque by approximately 15% across the time series. This fault indicates a potential fault in the engine and its inability to deliver its maximum potential. This fault affects only the actual engine torque reading and could be caused by any reason external to the other two sensors chosen for this work. The second fault also covers the possibility that the gradual faults in the engine are not caused by any of the sensors in this work and may be caused by factors that are not recognized in this work.

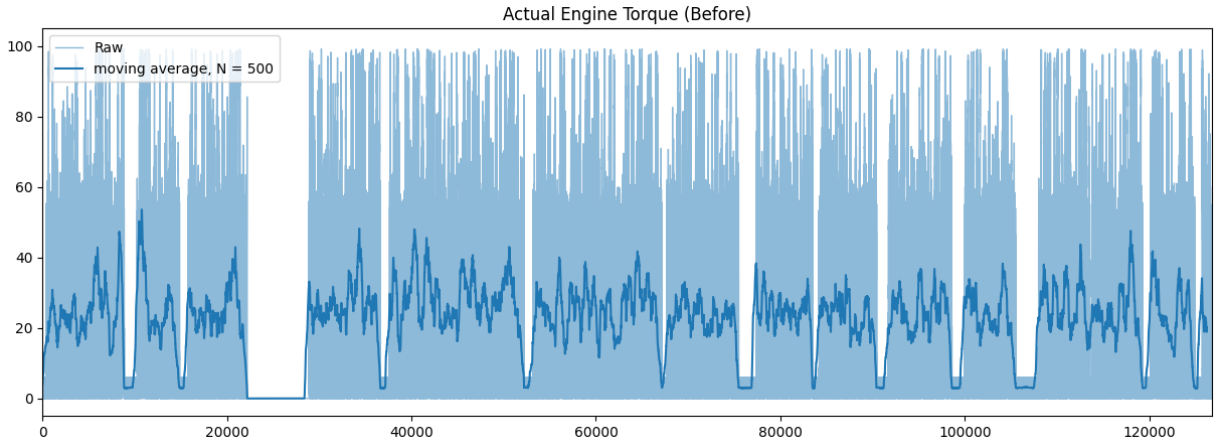

Figure 4.40: Actual engine torque before introducing the second fault

Figure 4.41 contains a plot of the actual engine torque readings after the second fault was introduced to it. Figure 4.40 contains a plot of the actual engine torque readings before introducing faults to it for comparison.

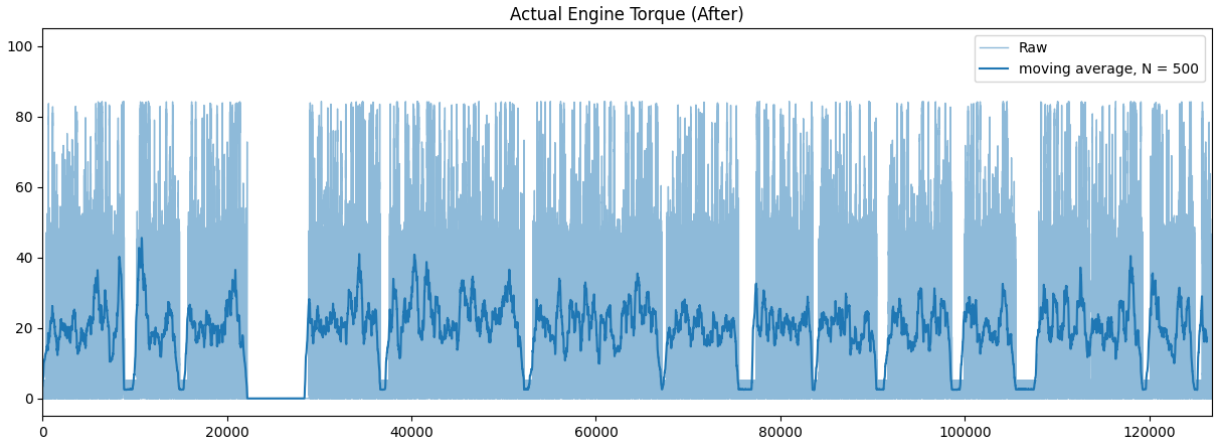

Figure 4.41: Actual engine torque after introducing the second fault

Lastly, the third fault involves gradually lowering the values of all three sensors throughout the length of the time series. The reduction in the values in this fault is not flat but instead is gradually increasing. This type of fault covers a situation in which a gradual fault is experienced or caused during a bus's up-time. For the third type of fault, the accelerator pedal position was lowered by approximately 15% throughout the time series, the driver's demand torque was lowered by a little bit more (approximately 16-17%), and the actual engine torque was lowered even more (approximately 18-20%) throughout the time series. The primary cause of this type of fault would lie in the accelerator pedal position being misread, or if the accelerator pedal cannot be pressed to its maximum possible position due to mechanical or electrical malfunctioning. The faulty behavior of the accelerator pedal results in a decrease in the readings of the other two sensors due to them being directly dependent on the accelerator pedal.

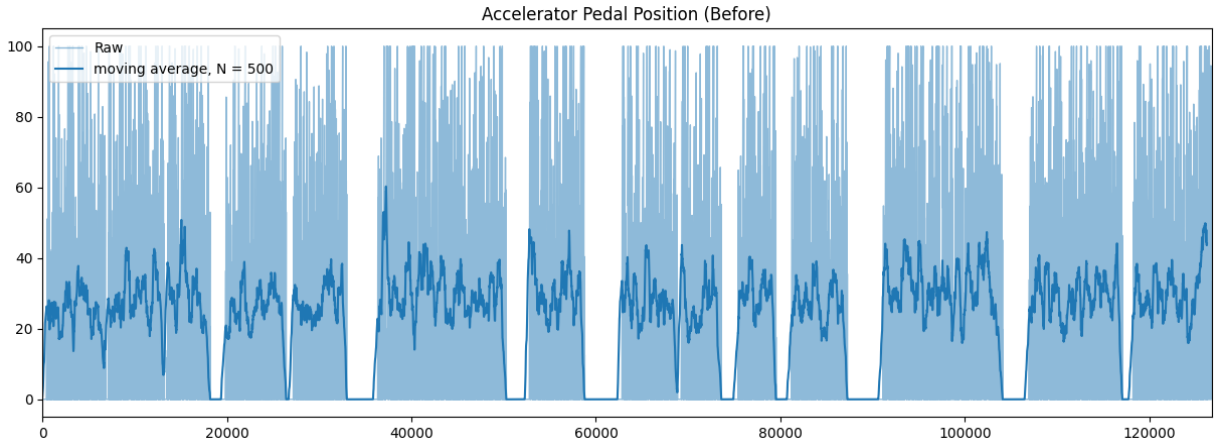

Figure 4.42: Accelerator pedal position before introducing the third fault

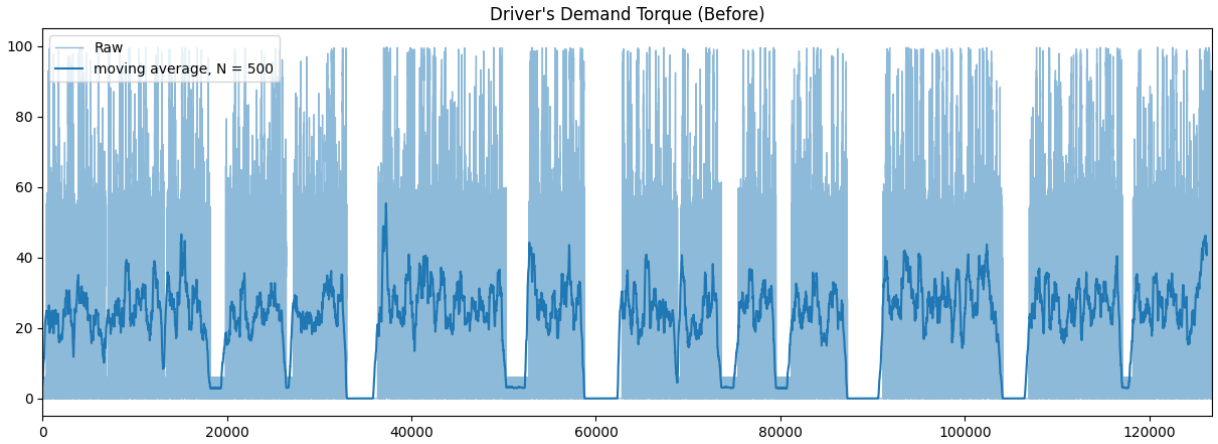

Figure 4.43: Driver's demand engine torque before introducing the third fault

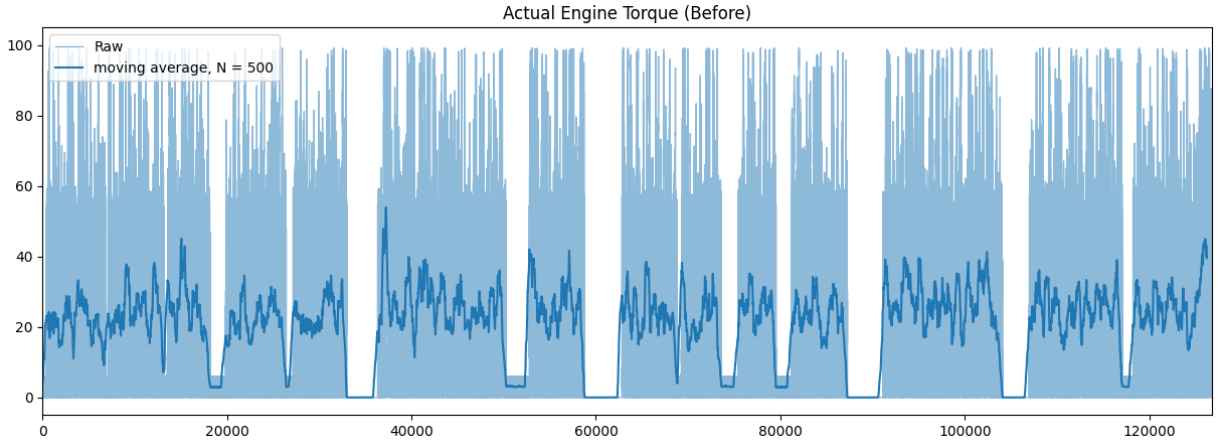

Figure 4.44: Actual engine torque before introducing the third fault

Figure 4.45 contains a plot of the accelerator pedal position readings after the third type of fault was introduced to it, Figure 4.46 contains a plot of the driver's demand engine torque after fault introduction to its readings, and Figure 4.47 contains a plot of the actual engine torque readings after the same type of fault was introduced to it. Figure 4.42, Figure 4.43, and Figure 4.44 contain plots of the accelerator pedal position, driver's demand engine torque, and the actual engine torque readings before introducing faults to them for comparison.

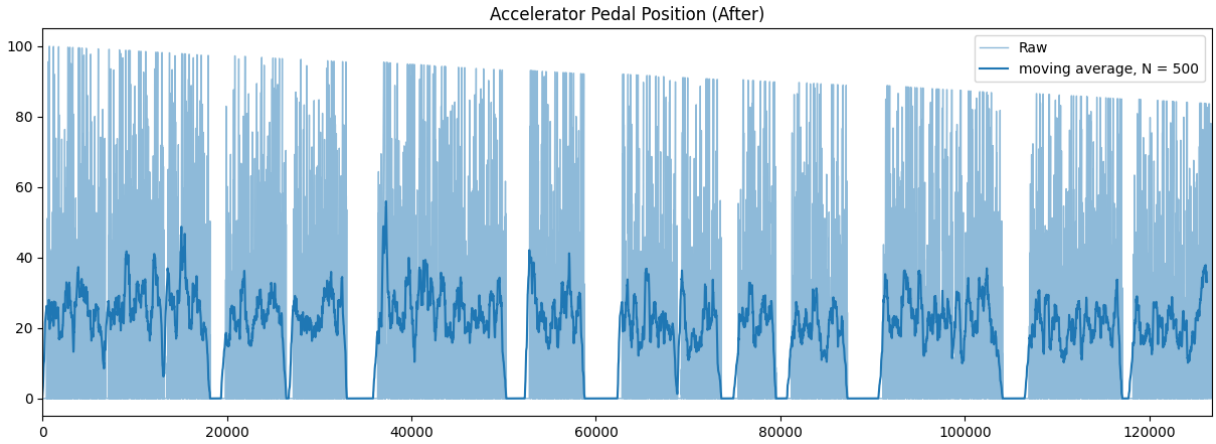

Figure 4.45: Accelerator pedal position after introducing the third fault

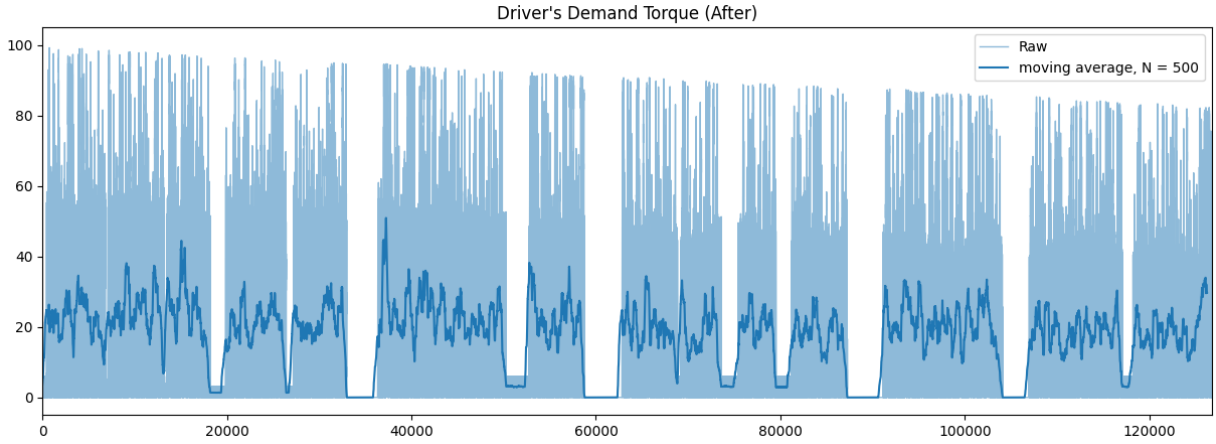

Figure 4.46: Driver's demand engine torque after introducing the third fault

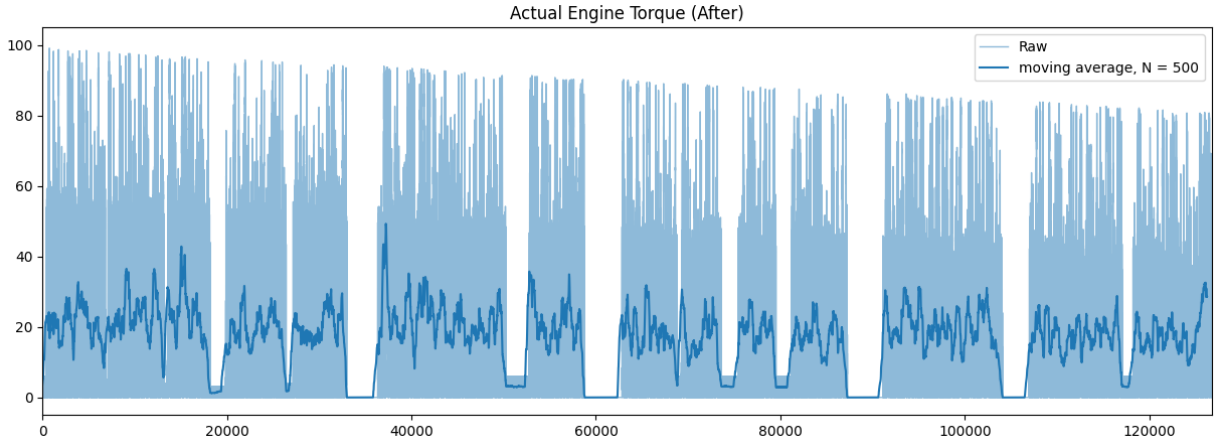

Figure 4.47: Actual engine torque after introducing the third fault

After introducing gradual faults to the engine torque system data, the entire process of generating synthetic data was complete. 40 buses were simulated for the cooling system out of which 8 buses contained gradual faults, and 38 buses were simulated for the engine torque system out of which 6 buses contained gradual faults.

In total, 78 datasets were generated belonging to two subsystems of an IoT-enabled bus which reasonably and accurately compare to data from a real bus. Note that the data generated in this work is completely independent of the proposed algorithm, in essence, the synthetic data generation process was not influenced by the proposed algorithm to encourage convenience and facilitate model performance. The synthetic data generated as well as the faults introduced were based entirely on the trends, patterns, relations, and distributions in the original data. Furthermore, the faults introduced were based on heuristics and the most common complaints and faults experienced by users on the internet across numerous transportation blogs and forums. Additionally, the figures in Subsection 4.1.3 and Subsection 4.2.3 that contain plots of sensor readings "before" faults were added to them are included intentionally to provide an additional visual aid to demonstrate the results of the synthetic data generation process for both the bus subsystems. Moreover, the figures also provide a contrast between normal functioning buses and buses with gradual faults and how a faulty subsystem may appear in the exact same circumstance.

Chapter 5

Experiments and Results

This chapter contains the experiments conducted on the data generated in Chapter 4 using the proposed predictive maintenance algorithm in Section 3.1; and the results from the experiments are also discussed. The experiments are conducted on the individual subsystems (cooling and engine torque) of the bus separately due to the difference in the structure of data requiring different preprocessing techniques. Moreover, this work has grouped sensors into subsystems based on relevance to and dependence on each other since the beginning so conducting experiments on individual subsystems and discussing the results seems appropriate. Furthermore, faults become easier to identify by isolating the faulty subsystem with faulty sensor readings. For example, finding out that the cooling system of a bus was flagged due to irregular fan speed and coolant temperature readings could help easily identify the fault pertaining to the cooling system of a bus instead of flagging the whole bus as faulty due to irregular sensor readings; the latter may require consideration of factors in addition to the cooling system and result in inefficiency. Hence, grouping the sensors based on relevance makes the fault identification step easier, so individual subsystems can be flagged as faulty instead of the whole bus. Furthermore, the clustering is performed for each sensor individually within a subsystem to identify any unusual patterns in the component that may or may not have any relation to the other components. The clustering results for each sensor in a subsystem of a bus can be combined to identify the type of fault and the cause of deterioration by experts.
The reason behind choosing a clustering approach is mainly the lack of real data with labels. Deep learning approaches have been rather popular amongst other methods due to the increasing computational powers of systems and the possibility of performing heavy computations at the cloud level instead of individual computers embedded in each vehicle [133]. However, in cases of lack of labeled data, traditional machine learning algorithms perform incredibly well. Deep learning methods largely rely on the quality of data and its being labeled, furthermore, there is usually an imbalance in such tasks because most of the vehicles function normally and only a few have potential problems - it is difficult for deep learning methods to tackle such imbalance problems, especially without accurate and sufficient data. Therefore, choosing an unsupervised clustering approach favors efficiency and accuracy.

5.1 Cooling System

The preprocessing of the cooling system data and the application of the predictive maintenance algorithm on it are discussed in this section.

5.1.1 Preprocessing

An initial preprocessing step for all cooling system sensor data includes converting all the data from float to integer type for memory efficiency, doing so results in no loss of information because the sensor recordings are whole numbers. Data from individual sensors was selected separately for all the buses and grouped into three time series datasets - a) coolant level data for all buses, b) fan speed for all buses, and c) coolant temperature for all buses. Essentially, the data for each sensor is combined and formatted into time series datasets containing 40 buses. Additional preprocessing steps were applied to the created time series based on the characteristics and requirements of individual sensors.
Firstly, the readings in which the coolant level value is 0% are removed, as they are noise. The coolant level values are fairly constant (in normal buses) throughout the up-time of a single bus, that is, coolant level values are not expected to change abruptly and then return to normal as it is practically improbable. Hence, using the entire duration of coolant level recordings for a bus was not required so the size of the coolant level time series was reduced by sampling sensor recordings at a fixed interval of 100. This increased efficiency without potentially losing important data or trends. Additionally, the coolant level dataset contained various values that are not relevant to any faults, for example, one bus may have a coolant level of 85% and another with just 50%, but that does not mean that there is anything wrong with either of them. Therefore, the shape of the time series was relevant to the problem at hand and not the magnitude of the values. Hence, the coolant level data was scaled to a distribution with zero mean and unit variance, doing so scaled all the normal data to have the same value (of zero) and trends could be observed within a standard deviation of approximately 1 and -1. The fan speed time series did not require a lot of preprocessing due to its small domain of values, this implies that there were only a few unique values present in the fan speed data due to an electric fan's ability to operate at a fixed rpm. Similar to the coolant level data, due to the limited value possibilities, the entire length of the fan speed data was not necessary. So, the size of the time series was reduced by sampling at a fixed interval of 10. The chosen value of the interval for the fan speed (10) is deliberately smaller than the interval for the coolant level (100). This is because choosing a larger interval resulted in almost eliminating the fan-turning-off (0% values) trend from the time series. Although no faults have been introduced in this work directly based on the fan's ability to turn off and on, the behavior may be relevant in other situations so those trends are preserved for experiment and algorithm testing purposes. Lastly, the coolant temperature data is noisy and has fluctuations so the moving average of the time series was calculated (with a window size of 1000) and used for smoothing the data. Using the moving average instead of raw data values helped emphasize the underlying trends while eliminating noisy and inconsistent readings. The coolant temperature time series was also reduced in size to smooth the data further and to increase efficiency. The

time series was sampled at a fixed interval of 200, which is larger than the values chosen for the previous sensors because a larger sampling value was needed to make the noisy coolant temperature data more interpretable. Furthermore, it was observed that reducing the size of all three time series by fixed interval sampling did not negatively impact model fitting and did not reduce the quality of the clusters created but only increased efficiency.

5.1.2 Model Hyperparameters

The choice of K (number of clusters) is based on the types of faults introduced for each sensor. The number of clusters for the coolant level is chosen to be 2, the number of clusters for the fan speed is chosen to be 3, and the number of clusters for the coolant temperature is chosen to be 2. The chosen values of K are also validated by using the elbow method mentioned in Chapter 3 for K values 2 to 7. The results show that the aforementioned chosen values of K based on the type of faults are optimal, because the sum of squared distances does not decrease significantly when increasing the value of K beyond the optimal value. Also, if the sum of squared distances continues to decrease, the clustering quality also decreases beyond the chosen values of K. The quality of the clusters is determined by using the silhouette score. The data was clustered using two distance metrics - the Euclidean distance and DTW. Additional hyperparameter tuning was performed if the baseline models failed to perform accurately.

5.1.3 Coolant Level Results

The results of the Euclidean clustering and DTW clustering for the coolant level dataset are the same. They both perform incredibly well at identifying the decline in the coolant level so the Euclidean K-Means clustering model can be used for this problem for efficiency. Moreover, the clustering models did not require any additional hyperparameter tuning and were able to perform accurately using the default parameters. The silhouette scores of the K=2 Euclidean clustering and the DTW clustering for the coolant level were equal to 0.99, which is extremely accurate. Figure 5.1 shows the two clusters using the Euclidean distance, one contains all the normal buses (left) and the other (right) contains the two buses with a decline in the coolant level. The thick red line in all the following cluster figures depicts the cluster centroid, while the less opaque gray lines depict individual time series.

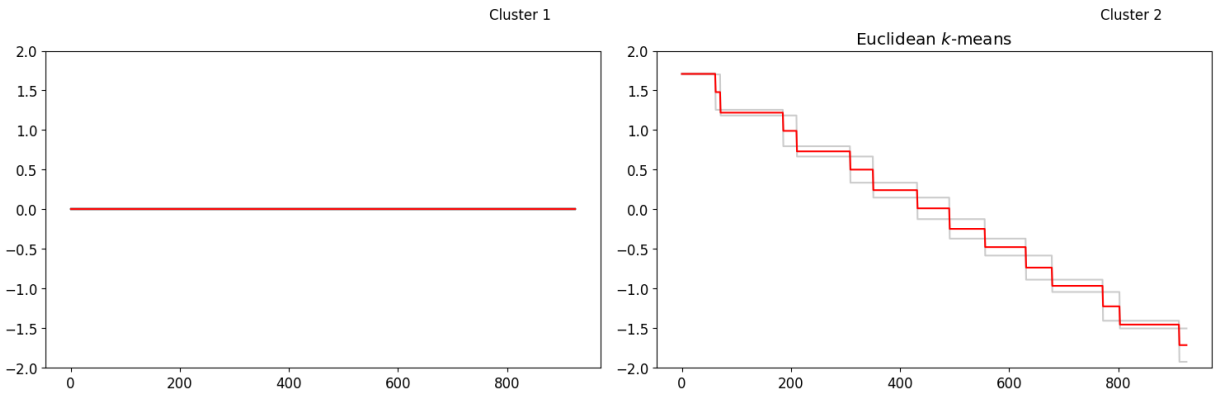

Figure 5.1: K=2 Clusters (Euclidean) for Coolant Level

5.1.4 Fan Speed Results

As for the fan speed time series, the Euclidean-based clustering without hyperparameter tuning fails to detect the slightly higher rpm in the fan speed dataset and results in a silhouette score of -0.24, which implies poor clustering quality. Figure 5.2 contains the clusters from the initial Euclidean clustering model and it is visible that the first cluster contains normal buses along with faulty buses (high fan speed rpm) as well as a normal bus being clustered separately in the third cluster. The reason for this might be that the 0 rpm values in one bus might result in a large distance for congruent non-zero points in other buses (normal without faults), so the slight increase does not result in a large enough distance for it to be significant enough to be identified as a separate cluster. However, increasing the number of centroid computations and the number of cluster initializations enabled the Euclidean clustering model to detect the faulty fan speed readings accurately and achieve a silhouette score of 0.88. On the other hand, the DTW-based clustering was able to identify the two faults introduced to the fan speed and cluster them correctly without any additional hyperparameter tuning, resulting in the same silhouette score of 0.88. Figure 5.3 shows the clusters using DTW and it is visible that all the faults are identified and clustered correctly - the first cluster contains all the normally functioning buses, the second cluster contains all the faulty buses with a lower fan speed, and the third cluster contains all the faulty buses with a higher than normal fan speed. The Euclidean-based clustering after performing hyperparameter tuning forms the same clusters as the DTW model with there being some differences in the structure of the centroids.

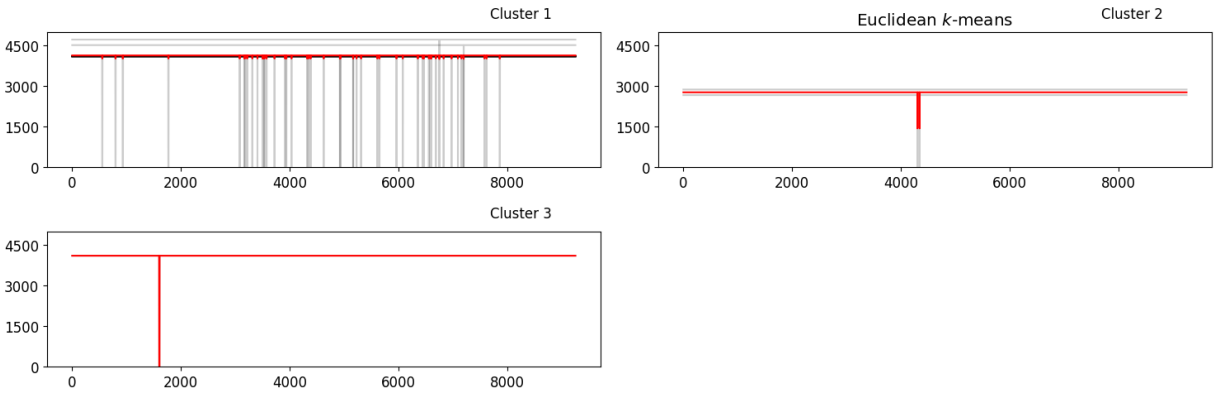

Figure 5.2: K=3 Clusters (Euclidean) for Fan Speed

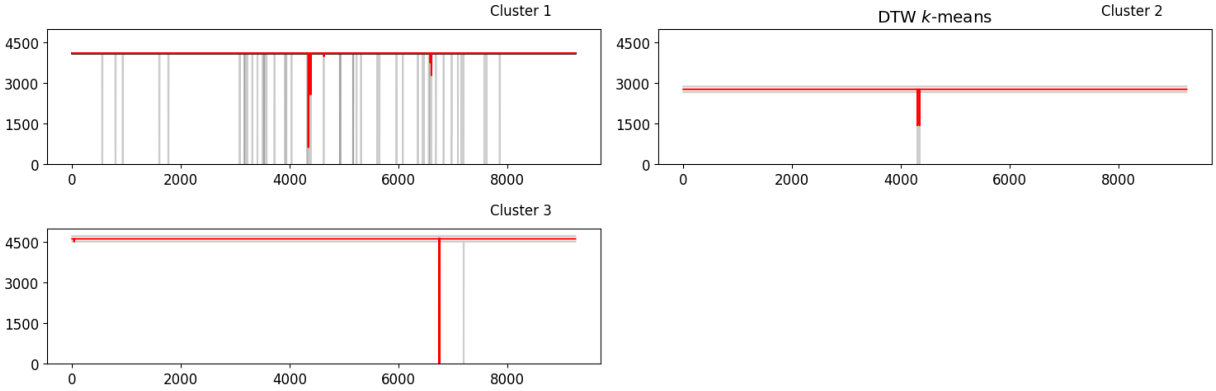

Figure 5.3: K=3 Clusters (DTW) for Fan Speed

5.1.5 Coolant Temperature Results

Lastly, for the coolant temperature time series, the Euclidean-based approach was initially unable to detect any faults and also formed two clusters containing normal and faulty buses jumbled up with a silhouette score of -0.10, indicating poor clustering quality. However, this was expected due to the complex trends and patterns present in the coolant temperature data. The noisy time series along with inconsistent temperature dips could negatively affect the distance calculations between points that are not related or temporally aligned. Figure 5.4 contains the clusters formed using Euclidean-based clustering without any hyperparameter tuning, the results show inaccurate clusters. Although the initial Euclidean model failed to perform well, performing hyperparameter tuning in the same way as for the fan speed Euclidean model (increasing the number of centroid computations and the number of cluster initializations) resulted in all the faulty buses being clustered accurately. The model formed clusters with a silhouette score of 0.62 which is significantly higher than the initial Euclidean model and Figure 5.5 shows the resultant clusters formed

that are accurate. The first cluster contains all the normal buses and the second cluster contains all the faulty buses in which the coolant temperature rises throughout the up-time. Lastly, The DTW-based clustering model was able to identify the faults and cluster them appropriately, although it failed to detect the slight rise in temperature in one bus. The rise in temperature is very subtle and is not detected by the model. Additional hyperparameter tuning also did not result in any change in the clusters. Figure 5.6 shows the clusters using DTW and it is visible that there is a time series in cluster 2 in which the temperature increases but it is clustered along with the normal buses. The silhouette score of the model is 0.64, which is the highest among the three models for coolant temperature despite the inaccurate clustering of one bus.

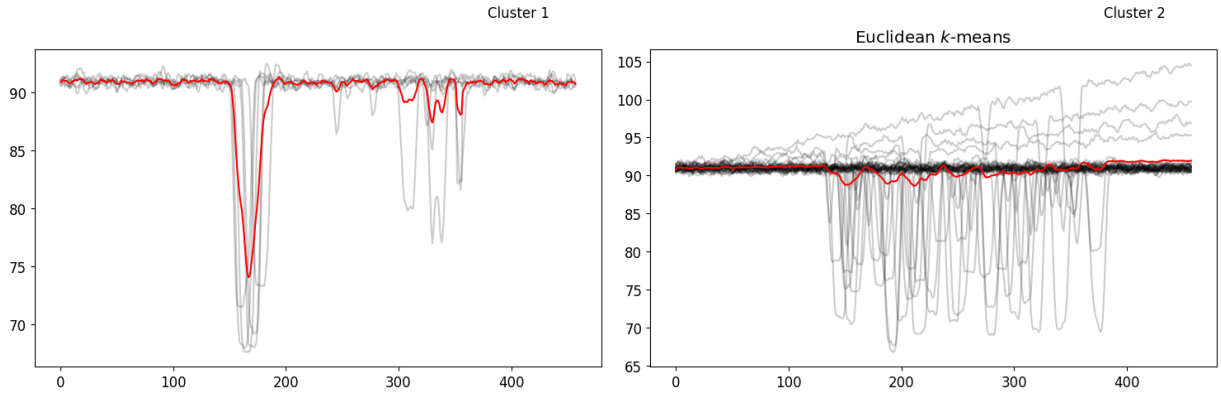

Figure 5.4: K=2 Clusters (Euclidean – No hyperparameter tuning) for Coolant Temperature

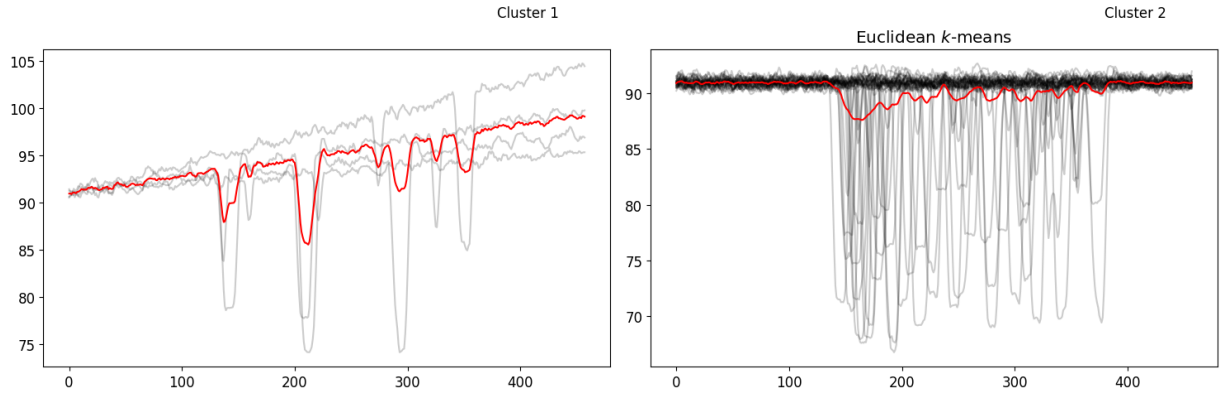

Figure 5.5: K=2 Clusters (Euclidean – With hyperparameter tuning) for Coolant Temperature

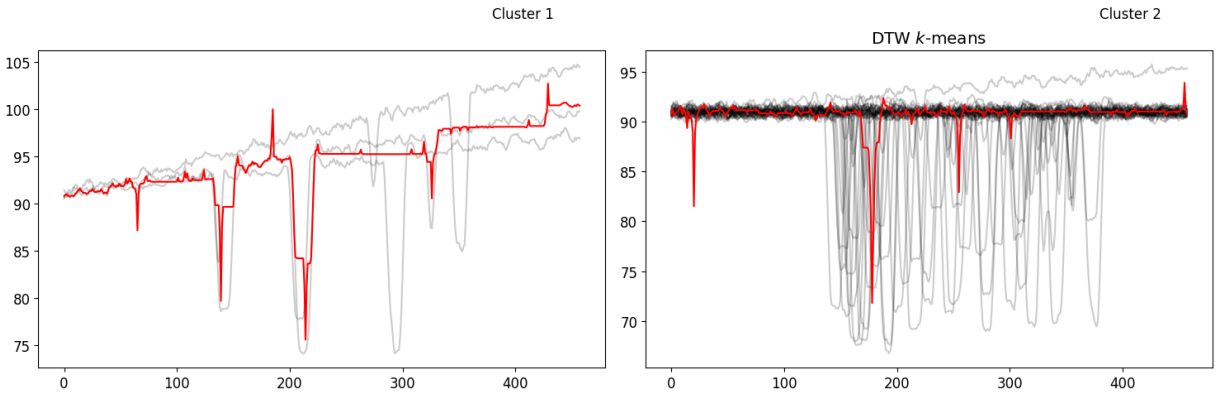

Figure 5.6: K=2 Clusters (DTW) for Coolant Temperature

The second Euclidean model and the DTW model both perform well in terms of accurate clustering and good clustering quality, therefore, additional data and testing are required to select the most efficient and accurate model for coolant temperature data. Eight additional test cases were generated specifically for the coolant temperature models to be evaluated on. The generation of the test cases was done using the same steps mentioned in Section 4.1 – a) cooling temperature sensor data was first generated using the DGAN model, b) missing trends were manually inserted into the data, and c) faults were added to two out of the eight buses. The two well-performing models – DTW-based and Euclidean-based with hyperparameter tuning – were used to cluster the test data. The models were only fit (or trained) on the initial data and were used to make clusters directly on the test data, in essence, the models were not retrained and re-initialized to perform clustering on the additional test data. The results show that both models perform the same with a silhouette score of 0.58; they are both able to cluster the normal buses accurately in one cluster and the two faulty buses separately in another cluster. Hence, using the Euclidean-based clustering model with hyperparameter tuning can be used for the coolant temperature due to its accurate results and low complexity.

5.1.6 Summary of Results

In summary, all the models perform as expected and are able to identify and cluster the faults appropriately while achieving high clustering quality. Table 5.1 contains the silhouette scores from all the clustering models implemented for each sensor. Based on the results, the Euclidean distance (with hyperparameter tuning) can be used to perform clustering on the data belonging to the cooling system due to the simplicity of most of the trends present in the cooling system sensor data. When considering the coolant temperature, the Euclidean approach seems to be performing slightly better than the DTW-based approach which is contrary to the expected result, however, the DTW-based approach performs better when only considering the evaluation metrics. However, the Euclidean-based model maintained the accuracy of its clustering with the test cases resulting in an overall higher

accuracy of clustering than the DTW-based model. In situations like this where there is a discrepancy between the clustering quality of one model and the clustering accuracy of another, it could be beneficial to continue to use both models for clustering to avoid any inaccuracy or performance loss from changes in the structure and complexity of data.

Sensor	Euclidean Clustering	Euclidean Clustering (Hyperparameter Tuning)	DTW Clustering
Coolant Level	0.99	–	0.99
Fan Speed	−0.24	0.88	0.88
Coolant Temperature	−0.10	0.62	0.64
Coolant Temperature (Test)	–	0.58	0.58

Table 5.1: Silhouette scores of the clustering approaches for the coolant system

5.2 Engine Torque System

This section contains the preprocessing steps used on the engine torque system data generated in Section 4.2 to test the proposed algorithm in Section 3.1.

5.2.1 General Preprocessing

The engine torque system sensor data contains values concentrated highly around the mean. Hence, the introduced gradual faults become difficult to observe due to the overall distribution of the data being very similar to the normal data, that is, concentrated around the mean. The primary effect of the faults introduced to the engine torque systems data is the delivery of lower-than-normal performance, or the system being unable to deliver its maximum possible performance. Therefore, the main preprocessing step of the engine torque system data involves using only values above a high percentile range. Doing so eliminates the large number of values around the mean that do not provide information about the decreased performance and it also results in smaller datasets, thereby increasing efficiency. The percentile values are retrieved from the original bus data that was physically collected from a public bus as this is considered the ground truth of a normally functioning bus. The percentile value ranges were chosen for each sensor based on repeated testing to find the optimal threshold beyond which relevant and informative data could be used. The percentile values chosen for each sensor are – a) greater than the 90^{th} percentile value for the accelerator pedal position, b) greater than the 95^{th} percentile value for the driver's demand eng torque, and c) greater than the 95^{th} percentile value for the actual engine torque. Considering only the higher percentile ranges of values allowed the models to perform clustering on the data in which the changes in performance could be observed. Furthermore, this resulted in time series of varying lengths as different buses have different

numbers of values in the aforementioned ranges due to the location and time of their operation. This implies that buses operating in peak hours in the city may tend to have a higher demand for high torque due to traffic (and frequently having to move the vehicle from standstill) as opposed to buses operating in the outskirts of the city with less traffic and fewer traffic lights as torque is then only needed to overcome wind resistance and friction. In addition to the buses demanding lower torque, the faulty buses may also not have as many values in the higher percentile ranges due to their inability to deliver the demanded torque. Hence, the mentioned percentile values were considered optimal as they contained reasonable sensor data from all buses and could be used to train models. After selecting the relevant data based on percentile values, the second preprocessing step involved taking the moving average of all the sensor data to smooth it and emphasize on the underlying trend and distribution. This was essential as the sensor data from the engine torque system was noisy and rough. Lastly, the size of the datasets was reduced by sampling at a fixed interval of 100 for all sensors. Doing so further reduced the size of the datasets to increase efficiency while avoiding any loss of useful information and trends. After all the preprocessing steps were conducted on the sensor data, the individual datasets were ready to be reshaped into time series for each sensor. In essence, three time series were created, one for each sensor, and each time series contained data from all buses for that sensor. This is the same reshaping method that was also used in Section 5.1. Moreover, the data was clustered using two distance metrics - the Euclidean distance and DTW. Additional hyperparameter tuning was performed if the baseline models failed to perform accurately. However, one major limitation of using the Euclidean distance is that all the time series needed to be of the same length. This was not possible due to the preprocessing steps mentioned earlier as they resulted in time series of varying lengths. Hence, to evaluate the performance of Euclidean-based models, all of the sensor data was selected, that is, no percentile ranges and thresholds were used to select only part of the data. This resulted in all the time series being of the same length (as they were generated in Section 4.2) so they could be used with the Euclidean-based clustering models. Preprocessing steps for the data to be used for the Euclidean-based models included taking the moving average to smooth the data and eliminate roughness and noise, and then sampling the data at a fixed interval of 100 to reduce the size of the datasets to increase efficiency. It is important to note that reducing the sizes of the datasets by fixed interval sampling was tested thoroughly to ensure that no information was lost and that there was no negative impact on the fitting of the clustering models and their consequent performance. On the other hand, the reduction only increased efficiency. Lastly, after the simple preprocessing steps for Euclidean clustering, the datasets were reshaped into three time series − one for each sensor.

5.2.2 Model Hyperparameters

Similar to the experiments conducted for the cooling system, the choice of K (number of clusters) is based on the types of faults introduced for each sensor. The number of clusters for the actual engine torque is chosen to be 3, the number of clusters for the driver's

demand engine torque is chosen to be 3, and the number of clusters for the accelerator pedal position is chosen to be 2. The chosen values of K are also validated by using the elbow method mentioned in Section 3.3.2 for K values 2 to 7. The results show that the chosen values of K based on the type of faults, are optimal, because the sum of squared distances does not decrease significantly when increasing the value of K beyond the optimal value. Also, if the sum of squared distances continues to decrease, the clustering quality also decreases beyond the chosen values of K. The quality of the clusters is determined by using the silhouette score.

5.2.3 Actual Engine Torque Results

DTW-based clustering was able to accurately cluster the actual engine torque data with a silhouette score of 0.88. The silhouette score is considerably high, especially when forming clusters for noisy data. Figure 5.7 shows the results from the DTW clustering, the first cluster contains the faulty buses with a low maximum performance, the second cluster contains all the normal buses, and the third cluster contains the faulty buses with a slow decline in performance throughout the duration of the buses' up-time. It is visible that all the buses were clustered accurately. Also, no additional hyperparameter tuning was necessary to achieve accurate clusters. On the other hand, the Euclidean distance model failed to cluster the data accurately and resulted in a silhouette score of 0.37. The silhouette score is not extremely low but the clusters formed are entirely inaccurate as visible in Figure 5.8. Furthermore, the Euclidean-based model was implemented with additional hyperparameter tuning to increase the model's capacity to compare a large number of centroids to find the optimal ones. The hyperparameters changed were the number of initializations of the cluster and the number of computations of centroids.

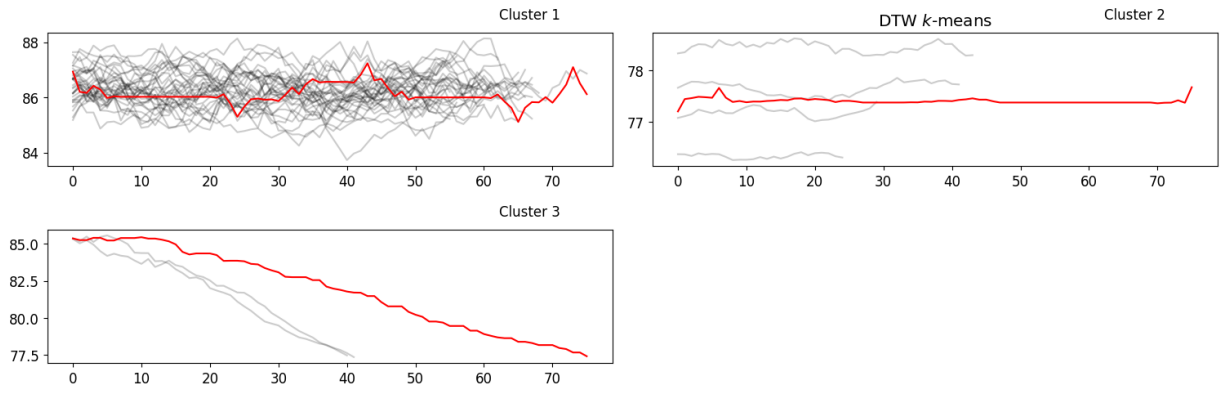

Figure 5.7: K=3 Clusters (DTW) for Actual Engine Torque

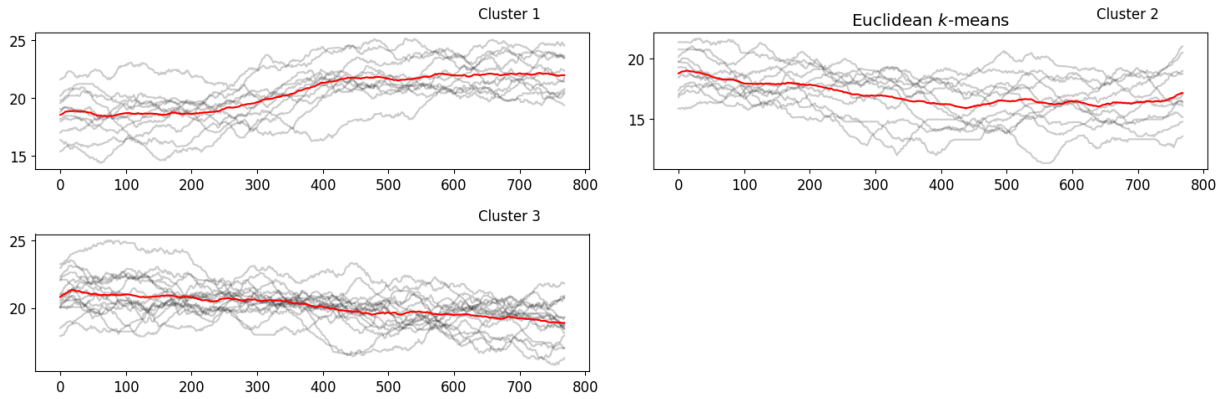

Figure 5.8: K=3 Clusters (Euclidean) for Actual Engine Torque

5.2.4 Driver's Demand Engine Torque Results

DTW-based clustering was also able to accurately cluster the driver's demand engine torque data and achieve a silhouette score of 0.84 which indicates very good quality clustering. Figure 5.9 contains the results from the DTW-based clustering model for the driver's demand engine torque, it is visible that all the buses were clustered appropriately – the first cluster contains all the normal buses, the second cluster contains the faulty buses in which the demanded torque declines throughout the buses' up-time, and the third cluster contains the faulty buses with a lower-than-normal demanded torque. On the other hand, Euclidean-based clustering was not able to cluster the data accurately despite the additional hyperparameter tuning and increased model complexity. The silhouette score achieved by the Euclidean model was 0.32 which is low and implies bad quality clusters. This is visible in Figure 5.10 which contains the results from the Euclidean-based clustering model for the driver's demand engine torque time series.

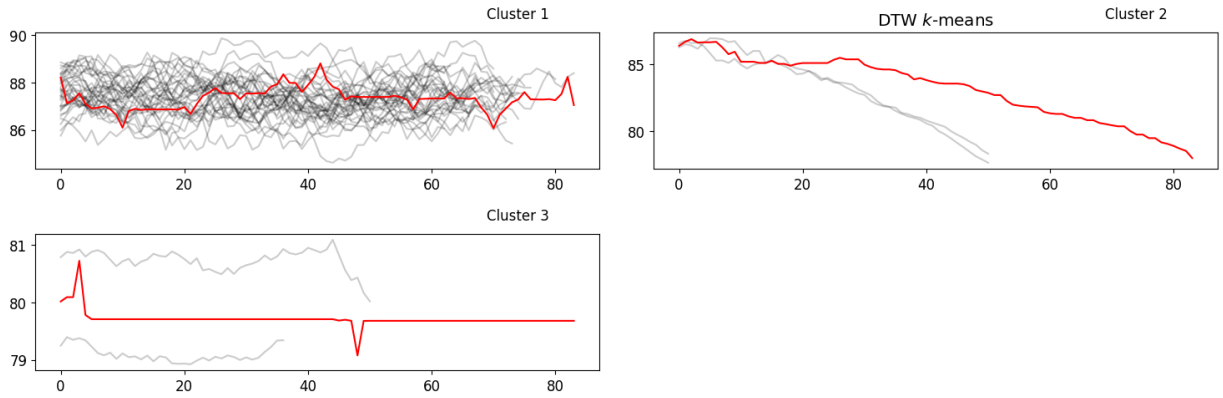

Figure 5.9: K=3 Clusters (DTW) for Driver's Demand Engine Torque

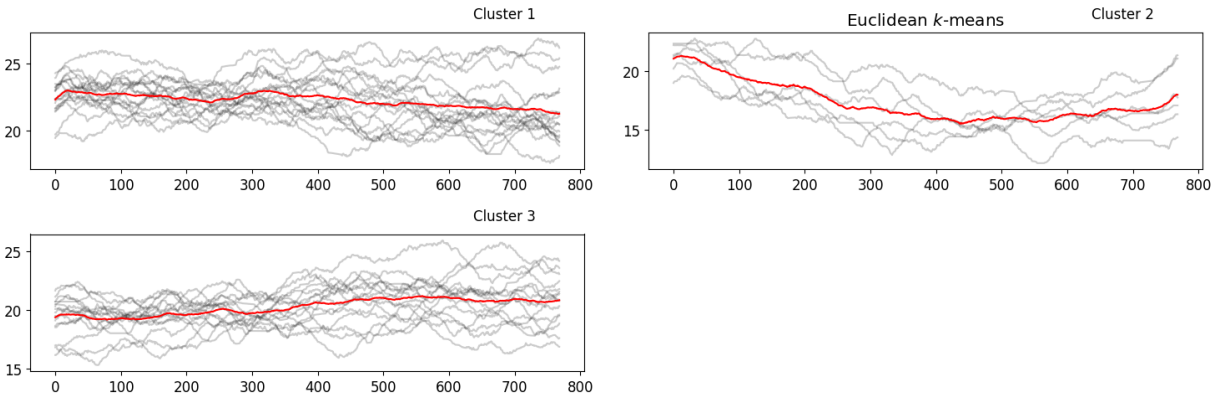

Figure 5.10: K=3 Clusters (Euclidean) for Driver's Demand Engine Torque

5.2.5 Accelerator Pedal Position Results

Lastly, successful and accurate clustering for the accelerator pedal position was also achieved using DTW-based clustering with a silhouette score of 0.79. This score is very high, especially when considering the fact that only 2 out of the 38 buses contained faults in the accelerator pedal, that is, there is an imbalance of approximately 5% in the accelerator pedal position time series and achieving a high clustering quality score along with accurate results strongly validates the efficiency and performance the proposed algorithm. Figure 5.11 shows the results from the DTW-based clustering model for the accelerator pedal position and it is visible that the first cluster contains all the normal buses and the second cluster contains the two faulty buses in which the maximum possible position of the accelerator pedal keeps declining throughout the buses' up-time. Euclidean-based clustering was also not able to achieve good results for the accelerator pedal position data. The silhouette score achieved by the Euclidean-based clustering mode was 0.39 which is low and indicates poor clustering. The results from the Euclidean-based model are visible in Figure 5.12. The Euclidean-based model for the accelerator pedal position was also tuned to increase the number of centroid computations and the number of cluster initializations but failed to achieve accurate clustering.

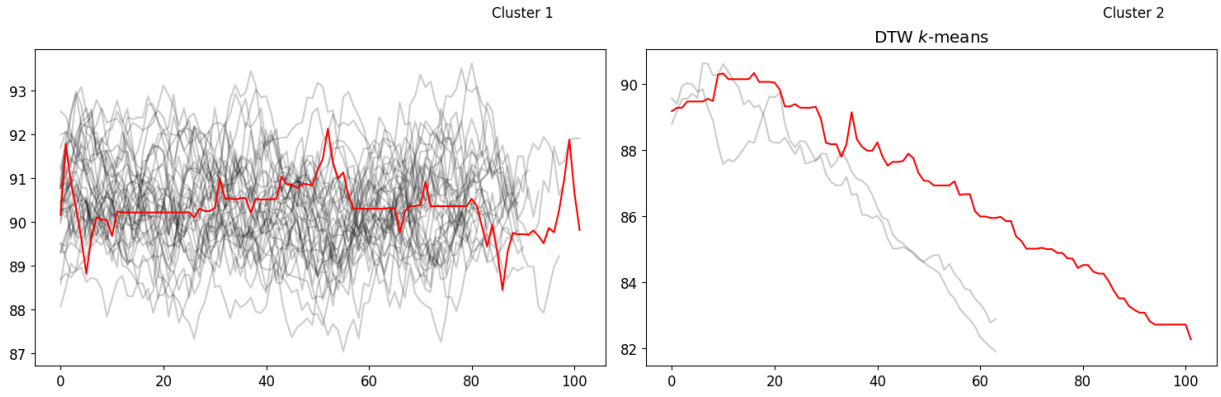

Figure 5.11: K=2 Clusters (DTW) for Accelerator Pedal Position

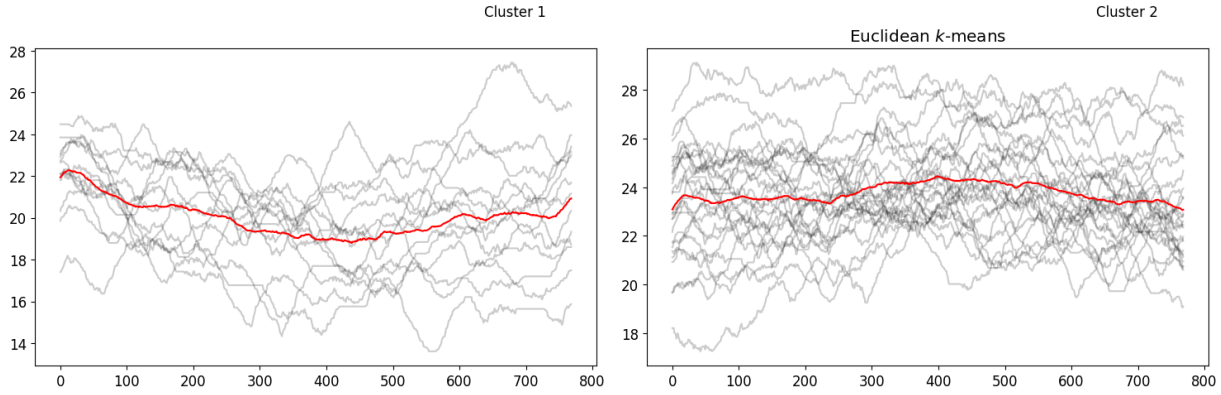

Figure 5.12: K=2 Clusters (Euclidean) for Accelerator Pedal Position

5.2.6 Summary of Results

To summarize the results from the experiments conducted for the engine torque systems, the DTW-based clustering model performed incredibly well in accurately clustering the buses based on functioning and faults, especially when considering the imbalance in the number of faulty and normal sensor readings across the three time series. Euclidean-based clustering was unsuccessful for all three sensors, however, this was expected due to the complexity of trends and distributions of the data belonging to the engine torque system. The limitation of having equal-sized time series results in inadequate preprocessing for the Euclidean-based models. Furthermore, due to the randomness and varying trends in the data, simply calculating the distance between points one by one based on indices does not seem sufficient to compare the time series. Attempting to use percentile ranges for the Euclidean-based models, followed by equalizing the lengths of the time series while ensuring no information is lost, and also while maintaining temporal correlations unnecessarily complicates the process and seems impractical as opposed to using DTW-based clustering

models for the engine subsystem. Moreover, the DTW-based models did not even require additional hyperparameter tuning as opposed to the Euclidean-based models, thereby increasing the simplicity of using DTW as the metric for the engine torque system. Table 5.2 contains the silhouette scores of clusters formed by the DTW-based models and Euclidean-based models with additional hyperparameter tuning for the three sensors belonging to the engine torque system. The DTW-based models outperform the Euclidean-based models in terms of accuracy, clustering quality, and simplicity.

Sensor	DTW Clustering	Euclidean Clustering (Hyperparameter Tuning)
Actual Engine Torque	0.88	0.37
Driver's Demand Engine Torque	0.84	0.32
Accelerator Pedal Position	0.79	0.39

Table 5.2: Silhouette scores of the clustering approaches for the engine torque system

Chapter 6

Conclusion

6.1 Summary

The work in this thesis addresses the issue of common maintenance strategies being costly and inefficient. Traditional maintenance methods, often relying on scheduled check-ups, are not only resource-intensive but can also lead to unnecessary downtime and maintenance activities when no issues are present. This thesis shines a light on an alternative strategy that is cost and time-efficient - predictive maintenance - and presents an algorithm to implement it. Predictive maintenance leverages data analytics and machine learning to anticipate potential faults before they result in significant failures. The work in this thesis presents an innovative clustering-based algorithm designed to implement predictive maintenance effectively. The presented algorithm is thoroughly tested for the cooling system and the engine torque system of a public bus to validate its performance, effectiveness, and practicality.

To address the challenge of insufficient and low-quality data, this thesis also generates two synthetic time series datasets to simulate two scenarios: a) buses that operate normally, and b) buses with underlying faults and gradual deterioration that would not be identified by standard maintenance systems, because the faults have not crossed a threshold after which they would result in component failure and damage. The aim of the datasets was to enable the training of models that could detect these underlying faults accurately and distinguish them from the buses that are normal. Doing so could allow the user to take preemptive measures to have the problem rectified without resulting in any failures and breakdowns. Such an approach ensures efficiency in terms of cost, resources, and time, and most importantly the safety of the user.

The clustering models in this work were trained mainly using synthetic data and focused on only six sensors in total. This was because other sensor data from the cooling system and the engine torque system could not be used due to them being just noise and unavailable. Despite these limitations, the models demonstrated promising results in accurately identifying potential faults.

The data and models in this thesis belong to the cooling system and the engine torque system, however, the approach presented in this thesis could be extended to any subsystem

of a bus such as the braking system or the engine speed delivery system. Creating models for these subsystems and combining them altogether could be used to perform predictive maintenance comprehensively for the entire bus so that no faults could go undetected.

6.2 Future Work

In conclusion, the work in this thesis addresses critical challenges in traditional maintenance strategies by presenting a cost-effective, time-efficient predictive maintenance approach. By leveraging synthetic data generation, advanced machine learning algorithms, and a focus on specific subsystems, this research provides a robust framework for improving maintenance practices. However, there is still room for improvement and advancement in the proposed approach.

A direction forward would be to gather additional real data from multiple buses of different models under various weather conditions over an extended period of time. This real-world data collection would provide a more diverse and robust dataset, enhancing the accuracy and generalizability of the predictive models. Furthermore, any buses experiencing faults could be labeled and the previous trips of those buses could be analyzed and used in the training of the model to find and predict the trends that resulted in failure.

Collecting sufficient data can permit the use of advanced and complex frameworks such as deep learning models that rely heavily on the amount of data they are trained on. Several trips of a single bus can be fed to models like transformers to expand the duration between the onset of deterioration and its detection. This would enable even earlier detection of potential issues, further improving maintenance strategies and enhancing the overall safety and efficiency of the transportation system.

Moreover, integrating IoT devices with advanced data analytics platforms can facilitate real-time monitoring and predictive maintenance. This integration would allow for continuous data collection and immediate analysis, providing instant alerts and recommendations for maintenance actions. The use of such an IoT architecture for a fleet of vehicles can enable and enhance scalability to handle large volumes of data and conduct predictive maintenance for an extensive fleet.

References

[1] Francesco Abate, Victor KL Huang, Gustavo Monte, Vincenzo Paciello, and Antonio Pietrosanto. A comparison between sensor signal preprocessing techniques. *IEEE Sensors Journal*, 15(5):2479–2487, 2014.

[2] Hervé Abdi, Dominique Valentin, and Betty Edelman. *Neural networks*. Number 124. Sage, 1999.

[3] John M Abowd and Lars Vilhuber. How protective are synthetic data? In *International Conference on Privacy in Statistical Databases*, pages 239–246. Springer, 2008.

[4] Srisairam Achuthan, Rishov Chatterjee, Sourabh Kotnala, Atish Mohanty, Supriyo Bhattacharya, Ravi Salgia, and Prakash Kulkarni. Leveraging deep learning algorithms for synthetic data generation to design and analyze biological networks. *Journal of Biosciences*, 47(3):43, 2022.

[5] Alankrita Aggarwal, Mamta Mittal, and Gopi Battineni. Generative adversarial network: An overview of theory and applications. *International Journal of Information Management Data Insights*, 1(1):100004, 2021.

[6] Mohiuddin Ahmed, Raihan Seraj, and Syed Mohammed Shamsul Islam. The k-means algorithm: A comprehensive survey and performance evaluation. *Electronics*, 9(8):1295, 2020.

[7] Moustafa Alzantot, Supriyo Chakraborty, and Mani Srivastava. Sensegen: A deep learning architecture for synthetic sensor data generation. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 188–193. IEEE, 2017.

[8] Fabio Arena, Mario Collotta, Liliana Luca, Marianna Ruggieri, and Francesco Gaetano Termine. Predictive maintenance in the automotive sector: A literature review. *Mathematical and Computational Applications*, 27(1):2, 2021.

[9] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[10] David Arthur, Sergei Vassilvitskii, et al. k-means++: The advantages of careful seeding. In *Soda*, volume 7, pages 1027–1035, 2007.

[11] Gholamreza Asadollahfardi and Gholamreza Asadollahfardi. Artificial neural network. *Water Quality Management: Assessment and Interpretation*, pages 77–91, 2015.

[12] Samuel A Assefa, Danial Dervovic, Mahmoud Mahfouz, Robert E Tillman, Prashant Reddy, and Manuela Veloso. Generating synthetic data in finance: opportunities, challenges and pitfalls. In *Proceedings of the First ACM International Conference on AI in Finance*, pages 1–8, 2020.

[13] Yang Bao, Guosheng Rui, and Song Zhang. A unsupervised learning system of aeroengine predictive maintenance based on cluster analysis. In *Proceedings of the 2020 International Conference on Aviation Safety and Information Technology*, pages 187–191, 2020.

[14] José Manuel Benítez, Juan Luis Castro, and Ignacio Requena. Are artificial neural networks black boxes? *IEEE Transactions on neural networks*, 8(5):1156–1164, 1997.

[15] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd international conference on knowledge discovery and data mining*, pages 359–370, 1994.

[16] James C Bezdek. A review of probabilistic, fuzzy, and neural models for pattern recognition. *Journal of Intelligent & Fuzzy Systems*, 1(1):1–25, 1993.

[17] Eric Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, 99(suppl_3):7280–7287, 2002.

[18] Peter Bühlmann. Bagging, boosting and ensemble methods. *Handbook of computational statistics: Concepts and methods*, pages 985–1022, 2012.

[19] Zaharah Allah Bukhsh, Aaqib Saeed, Irina Stipanovic, and Andre G Doree. Predictive maintenance using tree-based classification techniques: A case of railway switches. *Transportation Research Part C: Emerging Technologies*, 101:35–54, 2019.

[20] Stefan Byttner, Thorsteinn Rögnvaldsson, and Magnus Svensson. Consensus self-organized models for fault detection (cosmo). *Engineering applications of artificial intelligence*, 24(5):833–839, 2011.

[21] Gregory Caiola and Jerome P Reiter. Random forests for generating partially synthetic, categorical data. *Trans. Data Priv.*, 3(1):27–42, 2010.

[22] Joseph E Cavanaugh and Andrew A Neath. The akaike information criterion: Background, derivation, properties, application, interpretation, and refinements. *Wiley Interdisciplinary Reviews: Computational Statistics*, 11(3):e1460, 2019.

[23] Rachel SL Chan, Paul Gordon, and Michael R Smith. Evaluation of dynamic time warp barycenter averaging (dba) for its potential in generating a consensus nanopore signal for genetic and epigenetic sequences. In *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 2821–2824. IEEE, 2018.

[24] Kang-Tsung Chang. *Introduction to geographic information systems*, volume 4. McGraw-Hill Boston, 2008.

[25] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[26] Chong Chen, Ying Liu, Xianfang Sun, Carla Di Cairano-Gilfedder, and Scott Titmus. Automobile maintenance prediction using deep learning with gis data. *Procedia CIRP*, 81:447–452, 2019.

[27] Alice Consilvio, Angela Di Febbraro, and Nicola Sacco. A rolling-horizon approach for predictive maintenance planning to reduce the risk of rail service disruptions. *IEEE Transactions on Reliability*, 70(3):875–886, 2020.

[28] Juan Antonio Cortés-Ibáñez, Sergio González, José Javier Valle-Alonso, Julián Luengo, Salvador García, and Francisco Herrera. Preprocessing methodology for time series: An industrial world application case study. *Information sciences*, 514:385–401, 2020.

[29] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018.

[30] Mengyao Cui et al. Introduction to the k-means clustering algorithm based on the elbow method. *Accounting, Auditing and Finance*, 1(1):5–8, 2020.

[31] Jessamyn Dahmen and Diane Cook. Synsys: A synthetic data generation system for healthcare applications. *Sensors*, 19(5):1181, 2019.

[32] Jovani Dalzochio, Rafael Kunst, Edison Pignaton, Alecio Binotto, Srijnan Sanyal, Jose Favilla, and Jorge Barbosa. Machine learning and reasoning for predictive maintenance in industry 4.0: Current status and challenges. *Computers in Industry*, 123:103298, 2020.

[33] Ilesanmi Daniyan, Khumbulani Mpofu, Moses Oyesola, Boitumelo Ramatsetse, and Adefemi Adeodu. Artificial intelligence for predictive maintenance in the railcar learning factories. *Procedia Manufacturing*, 45:13–18, 2020.

[34] Fida K Dankar and Mahmoud Ibrahim. Fake it till you make it: Guidelines for effective synthetic data generation. *Applied Sciences*, 11(5):2158, 2021.

[35] Narjes Davari, Bruno Veloso, Gustavo de Assis Costa, Pedro Mota Pereira, Rita P Ribeiro, and João Gama. A survey on data-driven predictive maintenance for the railway industry. *Sensors*, 21(17):5739, 2021.

[36] Scott De Marchi and Scott E Page. Agent-based models. *Annual Review of political science*, 17:1–20, 2014.

[37] Xenofontas Dimitropoulos, Paul Hurley, Andreas Kind, and Marc Ph Stoecklin. On the 95-percentile billing method. In *Passive and Active Network Measurement: 10th International Conference, PAM 2009, Seoul, Korea, April 1-3, 2009. Proceedings 10*, pages 207–216. Springer, 2009.

[38] Norman R Draper and Harry Smith. *Applied regression analysis*, volume 326. John Wiley & Sons, 1998.

[39] Jörg Drechsler. Some clarifications regarding fully synthetic data. In *Privacy in Statistical Databases: UNESCO Chair in Data Privacy, International Conference, PSD 2018, Valencia, Spain, September 26–28, 2018, Proceedings*, pages 109–121. Springer, 2018.

[40] Harris Drucker, Christopher J Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. Support vector regression machines. *Advances in neural information processing systems*, 9, 1996.

[41] Eclipse. Eclipse mosquitto. https://mosquitto.org/, 2024. [Online, Accessed: 2024-06-06].

[42] Eclipse. Eclipse paho. https://eclipse.dev/paho/, 2024. [Online, Accessed: 2024-06-06].

[43] Abdellatif El Afia and Malek Sarhani. Particle swarm optimization for model selection of aircraft maintenance predictive models. In *Proceedings of the 2nd international Conference on Big Data, Cloud and Applications*, pages 1–12, 2017.

[44] Khaled El Emam, Lucy Mosquera, and Richard Hoptroff. *Practical synthetic data generation: balancing privacy and the broad availability of data.* O'Reilly Media, 2020.

[45] CSS Electronics. J1939 explained - a simple intro [2023]. https://www.csselectronics.com/pages/j1939-explained-simple-intro-tutorial, 2023. [Online, Accessed: 2024-05-6].

[46] Jay J Ely, Sandra V Koppen, Truong X Nguyen, Kenneth L Dudley, George N Szatkowski, Cuong C Quach, Sixto L Vazquez, John J Mielnik, Edward F Hogge, Boyd L Hill, et al. Radiated emissions from a remote-controlled airplane-measured in a reverberation chamber. Technical report, 2011.

[47] Shuai Fu and Nicolas P Avdelidis. Prognostic and health management of critical aircraft systems and components: An overview. *Sensors*, 23(19):8124, 2023.

[48] Pedro Cesar Lopes Gerum, Ayca Altay, and Melike Baykal-Gürsoy. Data-driven predictive maintenance scheduling policies for railways. *Transportation Research Part C: Emerging Technologies*, 107:137–154, 2019.

[49] Danilo Giordano, Flavio Giobergia, Eliana Pastor, Antonio La Macchia, Tania Cerquitelli, Elena Baralis, Marco Mellia, and Davide Tricarico. Data-driven strategies for predictive maintenance: Lesson learned from an automotive use case. *Computers in Industry*, 134:103554, 2022.

[50] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[51] M Goyal and S Kumar. Improving the initial centroids of k-means clustering algorithm to generalize its applicability. *Journal of The Institution of Engineers (India): Series B*, 95:345–350, 2014.

[52] Gretel.ai. Gretel dgan - gretel.ai. https://docs.gretel.ai/create-synthetic-data/models/synthetics/gretel-dgan, 2024. [Online, Accessed: 2024-05-6].

[53] Manbir Gulati and Paul Roysdon. Tabmt: Generating tabular data with masked transformers. *Advances in Neural Information Processing Systems*, 36, 2024.

[54] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[55] Yanfeng He, Yali Liu, Shuai Shao, Xuhang Zhao, Guojun Liu, Xiangji Kong, and Lu Liu. Application of cnn-lstm in gradual changing fault diagnosis of rod pumping system. *Mathematical Problems in Engineering*, 2019:1–9, 2019.

[56] Dirk Helbing. Agent-based modeling. In *Social self-organization: Agent-based simulations and experiments to study emergent social behavior*, pages 25–70. Springer, 2012.

[57] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*. John Wiley & Sons, 2013.

[58] VA Jane et al. Survey on iot data preprocessing. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, 12(9):238–244, 2021.

[59] Na Jiang, Hamdi Kavak, William G Kennedy, and Andrew T Crooks. Generation of reusable synthetic population and social networks for agent-based modeling. In *2021 Annual Modeling and Simulation Conference (ANNSIM)*, pages 1–12. IEEE, 2021.

[60] James Jordon, Lukasz Szpruch, Florimond Houssiau, Mirko Bottarelli, Giovanni Cherubin, Carsten Maple, Samuel N Cohen, and Adrian Weller. Synthetic data–what, why and how? *arXiv preprint arXiv:2205.03257*, 2022.

[61] Ali Furkan Kalay. Generating synthetic data with the nearest neighbors algorithm. *arXiv preprint arXiv:2210.00884*, 2022.

[62] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine Piatko, Ruth Silverman, and Angela Y Wu. The analysis of a simple k-means clustering algorithm. In *Proceedings of the sixteenth annual symposium on Computational geometry*, pages 100–109, 2000.

[63] Shruti Kapil and Meenu Chawla. Performance evaluation of k-means clustering algorithm with various distance metrics. In *2016 IEEE 1st international conference on power electronics, intelligent control and energy systems (ICPEICES)*, pages 1–4. IEEE, 2016.

[64] Mehmet Karakose and Orhan Yaman. Complex fuzzy system based predictive maintenance approach in railways. *IEEE Transactions on Industrial Informatics*, 16(9):6023–6032, 2020.

[65] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. ieee, 1995.

[66] Patrick Killeen, Iluju Kiringa, and Tet Yeap. Unsupervised dynamic sensor selection for iot-based predictive maintenance of a fleet of public transport buses. *ACM Transactions on Internet of Things*, 3(3):1–36, 2022.

[67] Hyunjoong Kim, Han Kyul Kim, and Sungzoon Cho. Improving spherical k-means for document clustering: Fast initialization, sparse centroid projection, and efficient cluster labeling. *Expert Systems with Applications*, 150:113288, 2020.

[68] Oleg Kobylin and Vyacheslav Lyashenko. Time series clustering based on the k-means algorithm. *Journal La Multiapp*, 2020.

[69] Panagiotis Korvesis, Stephane Besseau, and Michalis Vazirgiannis. Predictive maintenance in aviation: Failure prediction from post-flight reports. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1414–1422. IEEE, 2018.

[70] Akash Kothare, Shridhara Chaube, Yash Moharir, Gaurav Bajodia, and Snehlata Dongre. Syngen: synthetic data generation. In *2021 International Conference on Computational Intelligence and Computing Applications (ICCICA)*, pages 1–4. IEEE, 2021.

[71] Dhananjay Kumar and Bengt Klefsjö. Proportional hazards model: a review. *Reliability Engineering & System Safety*, 44(2):177–188, 1994.

[72] Mark Last, Alla Sinaiski, and Halasya Siva Subramania. Predictive maintenance with multi-target classification models. In *Intelligent Information and Database Systems: Second International Conference, ACIIDS, Hue City, Vietnam, March 24-26, 2010. Proceedings, Part II 2*, pages 368–377. Springer, 2010.

[73] Hongfei Li, Dhaivat Parikh, Qing He, Buyue Qian, Zhiguo Li, Dongping Fang, and Arun Hampapur. Improving rail network velocity: A machine learning approach to predictive maintenance. *Transportation Research Part C: Emerging Technologies*, 45:17–26, 2014.

[74] Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. Using gans for sharing networked time series data: Challenges, initial promise, and open questions. In *Proceedings of the ACM Internet Measurement Conference*, pages 464–483, 2020.

[75] Gianfranco Lombardo, Mattia Pellegrino, Agostino Poggi, et al. Unsupervised continual learning from synthetic data generated with agent-based modeling and simulation: a preliminary experimentation. In *WOA*, pages 116–126, 2022.

[76] Yingzhou Lu, Minjie Shen, Huazheng Wang, Xiao Wang, Capucine van Rechem, and Wenqi Wei. Machine learning for synthetic data generation: a review. *arXiv preprint arXiv:2302.04062*, 2023.

[77] Miro Mannino and Azza Abouzied. Is this real? generating synthetic data that looks real. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, pages 549–561, 2019.

[78] Oded Maron and Tomás Lozano-Pérez. A framework for multiple-instance learning. *Advances in neural information processing systems*, 10, 1997.

[79] Fausto Pedro Garcia Marquez. An approach to remote condition monitoring systems management. In *2006 IET international conference on railway condition monitoring*, pages 156–160. IET, 2006.

[80] Dhendra Marutho, Sunarna Hendra Handaka, Ekaprana Wijaya, et al. The determination of cluster number at k-mean using elbow method and purity evaluation on headline news. In *2018 international seminar on application for technology of information and communication*, pages 533–538. IEEE, 2018.

[81] Larry R Medsker, Lakhmi Jain, et al. Recurrent neural networks. *Design and Applications*, 5(64-67):2, 2001.

[82] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

[83] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine learning*, 4:227–243, 1989.

[84] Douglas C Montgomery, Elizabeth A Peck, and G Geoffrey Vining. *Introduction to linear regression analysis*. John Wiley & Sons, 2021.

[85] Fatemeh Mostofi, Onur Behzat Tokdemir, and Vedat Toğan. Generating synthetic data with variational autoencoder to address class imbalance of graph attention network prediction model for construction management. *Advanced Engineering Informatics*, 62:102606, 2024.

[86] John Moubray. *Reliability-centered maintenance*. Industrial Press Inc., 2001.

[87] Lingxia Mu, Wenzhe Sun, Youmin Zhang, and Nan Feng. Incipient gradual fault detection via transformed component and dissimilarity analysis. In *2023 IEEE 6th International Conference on Industrial Cyber-Physical Systems (ICPS)*, pages 1–6. IEEE, 2023.

[88] Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

[89] Hajra Murtaza, Musharif Ahmed, Naurin Farooq Khan, Ghulam Murtaza, Saad Zafar, and Ambreen Bano. Synthetic data generation: State of the art in health care domain. *Computer Science Review*, 48:100546, 2023.

[90] Saralees Nadarajah. The exponentiated exponential distribution: a survey. *AStA Advances in Statistical Analysis*, 95:219–251, 2011.

[91] Daniel A Newman. Missing data: Five practical guidelines. *Organizational Research Methods*, 17(4):372–411, 2014.

[92] Sergey I Nikolenko. *Synthetic data for deep learning*, volume 174. Springer, 2021.

[93] P Nunes, J Santos, and E Rocha. Challenges in predictive maintenance–a review. *CIRP Journal of Manufacturing Science and Technology*, 40:53–67, 2023.

[94] Society of Automotive Engineers. Sae j1939 standards collection - sae j1939 standards collection on the web. https://www.sae.org/standards/development/ground-vehicle/sae-j1939-standards-collection-on-the-web, 2024. [Online, Accessed: 2024-05-6].

[95] OpenAI. Generative models. https://openai.com/index/generative-models/, 2016. [Online, Accessed: 2024-05-6].

[96] Eva Ostertagová. Modelling using polynomial regression. *Procedia Engineering*, 48:500–506, 2012.

[97] Jian-Xin Pan, Kai-Tai Fang, Jian-Xin Pan, and Kai-Tai Fang. Maximum likelihood estimation. *Growth curve models and statistical diagnostics*, pages 77–158, 2002.

[98] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr, 2013.

[99] Diego J Pedregal, Fausto P García, and Felix Schmid. Rcm2 predictive maintenance of railway systems based on unobserved components models. *Reliability Engineering & System Safety*, 83(1):103–110, 2004.

[100] Yaling Pei and Osmar Zaïane. A synthetic data generator for clustering and outlier analysis. Technical report, University of Alberta Libraries, 2006.

[101] Duc Truong Pham, Stefan S Dimov, and Chi D Nguyen. Selection of k in k-means clustering. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 219(1):103–119, 2005.

[102] Derek A Pisner and David M Schnyer. Support vector machine. In *Machine learning*, pages 101–121. Elsevier, 2020.

[103] Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[104] Trivellore E Raghunathan. Synthetic data. *Annual review of statistics and its application*, 8:129–140, 2021.

[105] Henri Ralambondrainy. A conceptual version of the k-means algorithm. *Pattern Recognition Letters*, 16(11):1147–1157, 1995.

[106] MN Norazian Ramli, Ahmad Shukri Yahaya, NA Ramli, Noor Faizah Fitri Md Yusof, and MMA Abdullah. Roles of imputation methods for filling the missing values: A review. *Advances in Environmental Biology*, 7(12 S2):3861–3870, 2013.

[107] Ananth Ranganathan. The levenberg-marquardt algorithm. *Tutoral on LM algorithm*, 11(1):101–110, 2004.

[108] Steven J Rigatti. Random forest. *Journal of Insurance Medicine*, 47(1):31–39, 2017.

[109] Raul Rojas and Raúl Rojas. The backpropagation algorithm. *Neural networks: a systematic introduction*, pages 149–182, 1996.

[110] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[111] Alvida Mustika Rukmi and Ikhwan Muhammad Iqbal. Using k-means++ algorithm for researchers clustering. In *AIP Conference Proceedings*, volume 1867. AIP Publishing, 2017.

[112] Andrei Rykov, Renato Cordeiro De Amorim, Vladimir Makarenkov, and Boris Mirkin. Inertia-based indices to determine the number of clusters in k-means: an experimental evaluation. *IEEE Access*, 2024.

[113] Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007.

[114] Bernhard Schölkopf. The kernel trick for distances. *Advances in neural information processing systems*, 13, 2000.

[115] scikit learn. 2.3. clustering. https://scikit-learn.org/stable/modules/clustering.html, 2024. [Online, Accessed: 2024-05-08].

[116] SciPy. Statistical functions (scipy.stats). https://docs.scipy.org/doc/scipy/reference/stats.html, 2024. [Online, Accessed: 2024-05-6].

[117] Sanskar Shah, Darshan Gandhi, and Jil Kothari. Machine learning based synthetic data generation using iterative regression analysis. In *2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pages 1093–1100. IEEE, 2020.

[118] Ketan Rajshekhar Shahapure and Charles Nicholas. Cluster quality analysis using silhouette score. In *2020 IEEE 7th international conference on data science and advanced analytics (DSAA)*, pages 747–748. IEEE, 2020.

[119] Xiao-Sheng Si, Wenbin Wang, Chang-Hua Hu, and Dong-Hua Zhou. Remaining useful life estimation–a review on the statistical data driven approaches. *European journal of operational research*, 213(1):1–14, 2011.

[120] Kristina P Sinaga and Miin-Shen Yang. Unsupervised k-means clustering algorithm. *IEEE access*, 8:80716–80727, 2020.

[121] Michael Sivak and Brandon Schoettle. Recent changes in the age composition of drivers in 15 countries. *Traffic injury prevention*, 13(2):126–132, 2012.

[122] Vijender Kumar Solanki and Rohit Dhall. An iot based predictive connected car maintenance approach. *International Journal of Interactive Multimedia and Artificial Intelligence*, 2017.

[123] Padmanaba Srinivasan and William J Knottenbelt. Time-series transformer generative adversarial networks. *arXiv preprint arXiv:2205.11164*, 2022.

[124] Zhe Sun, Huaqiang Jin, Jiangping Gu, Yuejin Huang, Xinlei Wang, and Xi Shen. Gradual fault early stage diagnosis for air source heat pump system using deep learning techniques. *International Journal of Refrigeration*, 107:63–72, 2019.

[125] Shan Suthaharan and Shan Suthaharan. Decision tree learning. *Machine Learning Models and Algorithms for Big Data Classification: Thinking with Examples for Effective Learning*, pages 237–269, 2016.

[126] Shan Suthaharan and Shan Suthaharan. Support vector machine. *Machine learning models and algorithms for big data classification: thinking with examples for effective learning*, pages 207–235, 2016.

[127] Muhammad Ali Syakur, B Khusnul Khotimah, EMS Rochman, and Budi Dwi Satoto. Integration k-means clustering method and elbow method for identification of the best customer profile cluster. In *IOP conference series: materials science and engineering*, volume 336, page 012017. IOP Publishing, 2018.

[128] Romain Tavenard. An introduction to dynamic time warping. https://rtavenar.github.io/blog/dtw.html, 2021. [Online, Accessed: 2024-05-08].

[129] Amal Tawakuli, Daniel Kaiser, and Thomas Engel. Synchronized preprocessing of sensor data. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 3522–3531. IEEE, 2020.

[130] Andreas Theissler, Judith Pérez-Velázquez, Marcel Kettelgerdes, and Gordon Elger. Predictive maintenance enabled by machine learning: Use cases and challenges in the automotive industry. *Reliability engineering & system safety*, 215:107864, 2021.

[131] Wim J.C. Verhagen and Lennaert W.M. De Boer. Predictive maintenance for aircraft components using proportional hazard models. *Journal of Industrial Information Integration*, 12:23–30, 2018.

[132] Wlamir Olivares Loesch Vianna and Takashi Yoneyama. Predictive maintenance optimization for aircraft redundant systems subjected to multiple wear profiles. *IEEE Systems Journal*, 12(2):1170–1181, 2017.

[133] Gautam Vira, Patrick Killeen, Tet Yeap, and Iluju Kiringa. Predictive maintenance by detection of gradual faults in an iot-enabled public bus. In *2024 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 570–576, 2024.

[134] Anupriya Vysala and Dr Joseph Gomes. Evaluating and validating cluster results. *arXiv preprint arXiv:2007.08034*, 2020.

[135] Todd L Walton. Fill-in of missing data in univariate coastal data. *Journal of applied Statistics*, 23(1):31–40, 1996.

[136] Zhiqiang Wan, Yazhou Zhang, and Haibo He. Variational autoencoder based synthetic data generation for imbalanced learning. In *2017 IEEE symposium series on computational intelligence (SSCI)*, pages 1–7. IEEE, 2017.

[137] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[138] Peter Whittle. Restless bandits: Activity allocation in a changing world. *Journal of applied probability*, 25(A):287–298, 1988.

[139] CL Wu, Kwok Wing Chau, and C Fan. Prediction of rainfall time series using modular artificial neural networks coupled with data-preprocessing techniques. *Journal of hydrology*, 389(1-2):146–167, 2010.

[140] Jiawei Xie, Jinsong Huang, Cheng Zeng, Shui-Hua Jiang, and Nathan Podlich. Systematic literature review on data-driven models for predictive maintenance of railway track: Implications in geotechnical engineering. *Geosciences*, 10(11):425, 2020.

[141] Hong Yang, Aidan LaBella, and Travis Desell. Predictive maintenance for general aviation using convolutional transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 12636–12642, 2022.

[142] Liu Yi-ting, Xu Xiao-su, Liu Xi-xiang, Zhang Tao, Li Yao, Yao Yi-qing, Wu Liang, and Tong Jin-wu. A fast gradual fault detection method for underwater integrated navigation systems. *The Journal of Navigation*, 69(1):93–112, 2016.

[143] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270, 2019.

[144] Yang Yue, Ying Li, Kexin Yi, and Zhonghai Wu. Synthetic data approach for classification and regression. In *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 1–8. IEEE, 2018.

[145] L.A. Zadeh. Fuzzy logic. *Computer*, 21(4):83–93, 1988.

[146] Zhaoyu Zhang, Mengyan Li, and Jun Yu. On the convergence and mode collapse of gan. In *SIGGRAPH Asia 2018 Technical Briefs*, pages 1–4. 2018.

[147] Zheng Zhang, Romain Tavenard, Adeline Bailly, Xiaotong Tang, Ping Tang, and Thomas Corpetti. Dynamic time warping under limited warping path length. *Information Sciences*, 393:91–107, 2017.

[148] Jun Hua Zhao, ZhaoYang Dong, and Zhao Xu. Effective feature preprocessing for time series forecasting. In *International conference on advanced data mining and applications*, pages 769–781. Springer, 2006.