# PREDICTIVE DIAGNOSTICS FOR ELECTRIC AND IC ENGINE VEHICLES

**M.Tech. Thesis**

By
**VIGNESH R**


**CENTER FOR ELECTRIC VEHICLE AND INTELLIGENT TRANSPORT SYSTEMS**
**INDIAN INSTITUTE OF TECHNOLOGY INDORE**
**MAY 2024**

# PREDICTIVE DIAGNOSTICS FOR ELECTRIC AND IC ENGINE VEHICLES

**A THESIS**

*Submitted in partial fulfillment of the requirements for the award of the degree*
***of***
**Master of Technology**

*by*
**VIGNESH R**


**CENTER FOR ELECTRIC VEHICLE AND INTELLIGENT TRANSPORT SYSTEMS**
**INDIAN INSTITUTE OF TECHNOLOGY INDORE**
**MAY 2024**

# INDIAN INSTITUTE OF TECHNOLOGY INDORE

## CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **PREDICTIVE DIAGNOSTICS FOR ELECTRIC AND IC ENGINE VEHICLES** in the partial fulfillment of the requirements for the award of the degree of **MASTER OF TECHNOLOGY** and submitted in the **CENTER FOR ELECTRIC VEHICLE AND INTELLIGENT TRANSPORT SYSTEMS, Indian Institute of Technology Indore**, is an authentic record of my own work carried out during the time period from July 2022 to May 2024 under the supervision of Prof. I A Palani, Professor, Department of Mechanical Engineering.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

R. Vignesh 29/05/2024

**Signature of the student with date**
**(VIGNESH R)**

---

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

29/05/2024

Signature of the Supervisor of
M.Tech. thesis (with date)

**(Dr. I A PALANI)**

---

**VIGNESH R** has successfully given his M.Tech. Oral Examination held on **27th May 2024**.

Signature(s) of Supervisor(s) of M.Tech. thesis
Date: 29/05/2024

---

# ACKNOWLEDGEMENTS

I extend my sincere gratitude to **Professor I A Palani**, whose expertise, guidance, and unwavering support have been the cornerstone of this endeavor. His mentorship not only shaped the direction of this research but also instilled a deeper understanding of the subject matter. I am indebted to him for his patience and dedication throughout this journey.

I am also grateful to the Head of the Department, **Dr. Amod Umarikar**, for allowing me to take up an industrial internship project. His encouragement and vision have provided the framework within which this project could thrive.

I am thankful to Volvo Eicher Commercial Vehicles Limited team **Mr. Kapil G Krishnan, Mr. Deepak Dubel, Mr. Harshit Garg** and **Mr. Dinesh Vijayan** for trusting me with this opportunity and providing practical insights throughout the project duration.

A heartfelt thank you is extended to the esteemed Lab Managers **Mr. Ashwin Wagh, Mr. Krishnpal Tomar,** Post-Doctoral Scholars **Dr. Nandini Patra, Dr. Jayachandran S, Dr. T. Geethapriyan, Dr. Anshu Sahu,** and PhD students **Mr. Kaushal Gangwar, Mr. Arpit Kumar Singh, Ms. Diksha Jaurker** whose invaluable contributions and constructive feedback have significantly enriched the quality of this work.

To my batchmates **Mr. Hrishikesh Meshram, Mr. Pavan Kumar Mangiri, Mr. Mohd Washique Ahemad, Mr. Sayan Doloi,** I express my appreciation for your collaboration, camaraderie, and shared commitment to academic excellence.

I am also grateful to the junior B. Tech students **Mr. Kailaash Pandiyan, Mr. Puneet Gupta,** who enthusiastically lent their support and assistance whenever needed. Their eagerness to engage with the project and willingness to contribute reflect the spirit of academic curiosity and collaboration that defines our institution.

Lastly, I extend my heartfelt thanks to my family and friends **Mr. Shriram S, Mr. Narasimmavinay N, Mr. Naveen L, Mr. Gopi Talluri,** whose unwavering support, encouragement, and companionship have been a source of strength and inspiration throughout this journey.

# Dedicated to

# My Beloved Family

# Abstract

This work explores integrating Machine Learning (ML) and logic-based models for predictive diagnostics in Electric Vehicles (EVs) and Internal Combustion (IC) engine vehicles, using data from Volvo Eicher Commercial Vehicles Limited (VECV). The objective of this work is to develop a predictive analytics model to forecast potential faults, optimize vehicle parameters, and enhance maintenance strategies. Key use cases include cell imbalance monitoring, temperature monitoring of cells and motors, and engine oil pressure warnings, aiming to improve fault detection systems' reliability. Despite challenges like the availability of labeled data for rare events in limited numbers and the computational demands of deep learning models for real-time applications, this thesis establishes a foundation for future advancements in automotive predictive analytics.

The methodology involves analyzing time-series data to monitor vehicles' dynamic performance, using ML models like Vector Auto Regression (VAR), Autoregressive Integrated Moving Average (ARIMA), Extreme Gradient Boosting (XGBoost), Light Gradient Boosting Machine (LGBM), and Long Short-Term Memory (LSTM) neural networks. These models capture temporal dependencies, while logic-based models provide interpretable rules for decision-making. The findings indicate that ML models excel in identifying complex patterns and nonlinear relationships, leading to highly accurate predictions for battery health and engine performance. ML models effectively predict battery cell imbalances and temperature variations, crucial for maintaining optimal battery performance and longevity. Logic-based models offer clear, interpretable rules essential for understanding vehicle behavior and regulatory compliance.

The research's key contributions include a comparative analysis of ML and logic-based models, highlighting their respective advantages and limitations. By combining ML's predictive power with the interpretability of logic-based models, the study suggests more robust predictive systems. Practical applications using real-world data show improvements in fault prediction accuracy and reduced false alarms. A hybrid model, advanced ML techniques, and enhance the real-time scalability of predictive models can be a way forward to improve performance optimization, and overall reliability.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# NOMENCLATURE

$TP$ – True Positive

$FP$ – False Positive

$TN$ – True Negative

$FN$ – False Negative

# ACRONYMS

IC – Internal Combustion

EV – Electric Vehicle

OEM – Original Equipment Manufacturer

VECV – Volvo Eicher Commercial Vehicles

ML – Machine Learning

ECU – Electronics Control Unit

TCU – Telematics Control Unit

VAR – Vector Auto Regression

ARIMA – Autoregressive Integrated Moving Average

XGBoost – Extreme Gradient Boosting

LGBM – Light Gradient Boosting Machine

LSTM – Long Short-Term Memory

RNN – Recurrent Neural Network

MACN – Multi Attention Collaborative Network

DFM – Dynamic Factor Model

DFML – Dynamic Factor Machine Learning

SOH – State of Health

AI – Artificial Intelligence

IoT – Internet of Things

BMS – Battery Management Systems

SOC – State of Charge

SEI – Solid Electrolyte Interphase

PMSM – Permanent Magnet Synchronous Motors

RPM – Rotations Per Minute

EDA – Exploratory Data Analysis

RMSE – Root Mean Square Error

MAE – Mean Absolute Error

CSV – Comma Separated Values

MSE – Mean Square Error

DOE – Design of Experiments

# Chapter 1

# Introduction

Electric vehicles have garnered a lot of attention in recent times due to their potential to reduce environmental impact and the dependency on traditional fuel sources. However, alongside their benefits, these vehicles present unique challenges such as range anxiety, charging infrastructure limitations, and frequent maintenance issues, particularly in the case of Electric Vehicles (EVs) due to the adaptation of new technologies. Traditional preventive maintenance strategies incur moderate to high costs, and unexpected failures between maintenance schedules further aggravate concerns for vehicle owners and fleet operators.

Predictive analytics offers a promising solution to mitigate these challenges by providing real-time alerts and early warnings, thereby minimizing breakdown situations and associated costs. For stakeholders such as customers, dealers, and Original Equipment Manufacturers (OEMs), predictive analytics translates into tangible benefits including cost savings from longer equipment life, increased revenue opportunities, improved customer experience, and reduced product recalls. By harnessing predictive analytics, it is possible to optimize vehicle parameters, predict component failures, identify root causes for faults, and enhance the overall driving experience.

The objective of this work, conducted in collaboration with Volvo Eicher Commercial Vehicles Limited (VECV), is to develop advanced predictive analytics models using machine learning (ML) and logic-based approaches. The specific use cases are cell imbalance monitoring, temperature monitoring of cell & motor, and engine oil pressure warnings. The main objective of this study is mentioned below:

- Identify root causes for potential faults and failures in Electric and Internal Combustion (IC) engine vehicles.

- Identify the right parameters and thresholds for accurate fault prediction.

- Address false alarms to improve the reliability of fault detection systems.

- Enhance fault prediction accuracy.

- Develop actionable plans based on failure modes and patterns.

## 1.1 Vehicle Data

Vehicles generate a lot of data to analyze, measure, compare, and correct themselves to function. These data help various Electronic Control Units (ECU) to decide as programmed. A few selected parameters share data from the vehicle to the cloud using a Telematics Control Unit (TCU). These parameters are used to visualize, analyze, and identify the location of the vehicle in real-time.

This data can also be used to plan vehicle maintenance as per the existing conditions of the parts instead of scheduled maintenance. VECV obtains vehicle data of various parameters every minute and stores it in its TCU. This unit has an in-built capability to upload data to the cloud server. VECV's uptime center accesses this data to filter out vehicles under breakdown to provide maintenance services. The parameters collected from the electric vehicle for the use cases are shown in Figure 1.

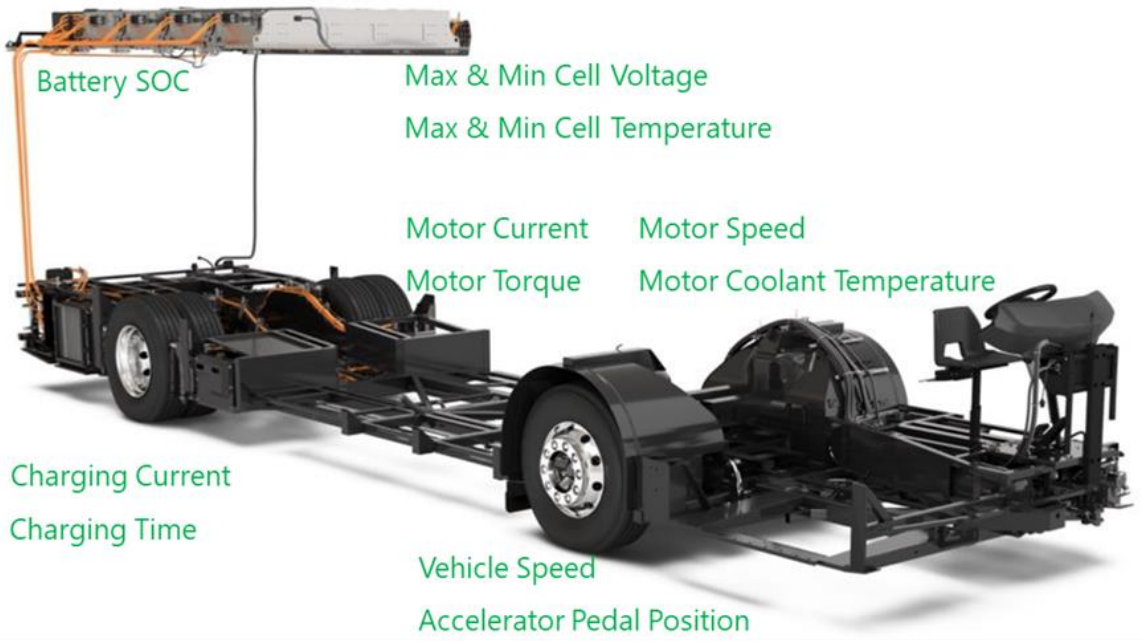



Figure 1 Parameters collected from vehicle

## 1.2 Predictive Maintenance

Predictive Maintenance helps to maintain the vehicle with lower downtime, and it is more critical to commercial vehicles as the vehicle availability is related to the total cost of ownership. The data generated by the vehicle is shared with the uptime center and based on the severity of the issue the maintenance activity takes place. Based on the

location of the vehicle and the probability of part failure, the service centers keep the parts readily available to reduce the maintenance time.

The historical data and the current vehicle data help to predict the future occurrences of the faults and the remaining useful life of parts by using machine learning. This helps in planning maintenance appointments, which is convenient and cost-effective. The flow of a vehicle predictive maintenance system is shown in Figure 2.

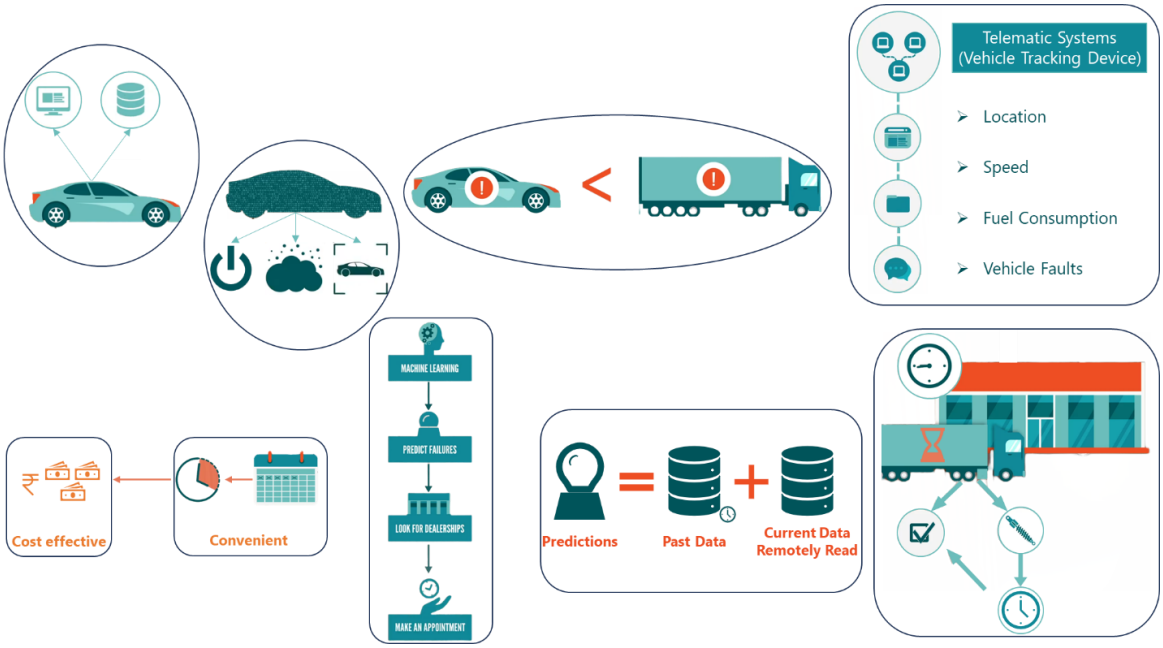



Figure 2 Vehicle Predictive Maintenance System

## 1.3 Machine Learning Methods

The dataset handled in this study is a time-series dataset which consists of sequential data collected every minute when the vehicle is being operated. A diverse array of machine learning methods tailored for time series datasets are experimented with to address the multifaceted challenges inherent in the use cases.

Vector Auto Regression (VAR) models offer a powerful framework for capturing the dynamic interdependencies among multiple time series variables, enabling accurate forecasting of future states based on past observations. Autoregressive Integrated Moving Average (ARIMA) models excel in capturing the temporal dynamics and seasonal patterns inherent in time series dataset, is suited for predicting tasks where stationary and differenced data are prevalent.

Extreme Gradient Boosting (XGBoost) and Light Gradient Boosting Machine (LGBM) algorithms leverage the power of ensemble learning to combine the predictive capabilities of multiple weak learners, resulting in robust and accurate predictions for complex time series data. Additionally, Long Short-Term Memory (LSTM) neural networks, is a recurrent neural network (RNN) model, excel in capturing long-term dependencies and sequential pattern in time series dataset, is suited for modeling complex temporal relationships in electric and IC engine vehicle datasets. This project aims to develop sophisticated predictive analytics models capable of accurately forecasting future states, diagnosing faults, and optimizing vehicle performance in real-time by choosing the best-performing model for these use cases.

## 1.4 Motivation of the Work

The motivation behind this project stems from the increase in significance of predictive analytics in the automotive industry, particularly in addressing the evolving challenges posed by electric and IC engine vehicles. With the growing adoption of EVs, the need for proactive maintenance strategies to mitigate vehicle-related issues has become paramount.

VECV electric buses observed a few battery and motor-related issues in their first batch of vehicles which impacted the vehicle performance. A predictive analytics model was needed to revolutionize maintenance practices by providing real-time alerts, early warnings, and proactive communication to stakeholders, thereby enhancing vehicle performance, prolonging equipment life, and ultimately improving customer satisfaction.

Through the development of advanced machine learning and logic-based models, the project endeavors to predict and diagnose faults accurately, identify root causes, and formulate actionable strategies for maintenance and repair, thereby guiding in a new era of efficiency and reliability in the automotive sector.

## 1.5 Organization of the Thesis

**Literature Review and Problem Formulation:**

This chapter delves into a comprehensive review of relevant literature, journals, and research articles pertinent to predictive analytics of vehicles using several machine learning methods. It also discusses the types of datasets used and delineates the

preprocessing steps undertaken for different ML models. The development of ML model for unique use cases by combining various tools and methods was also discussed.

**Machine Learning Models:**

This chapter provides exploration of the ML models experimented with in this project, including Vector Auto Regression (VAR), Autoregressive Integrated Moving Average (ARIMA), Extreme Gradient Boosting (XGBoost), Light Gradient Boosting Machine (LGBM), and Long Short-Term Memory (LSTM) Neural Networks. It discusses the principles behind each model, their applications in predictive analytics, and their respective strengths and weaknesses.

**Logic-Based Models:**

In this section, the logic-based models employed in the project are elucidated. It discusses the rationale behind logic-based modeling, the types of logic models utilized, and their applications in predictive analytics. Furthermore, it explores how logic-based models complement machine-learning approaches and contribute to the overall predictive analytics framework for electric and IC engine vehicles.

**Results and Discussions:**

The results and discussions section presents a comprehensive overview of the findings obtained from the developed models. It compares the performance of ML and logic-based models, discusses their strengths and weaknesses, and explores insights gleaned from logic models for optimizing machine learning models. Furthermore, it examines cross-validation and model validation techniques, scalability considerations, and interpretability aspects.

**Conclusions and Scope for Future Work:**

This concluding chapter summarizes the key findings of the research and evaluates the extent to which the project objectives were met. It discusses the contributions made by the project to the field of predictive analytics for electric and IC engine vehicles, acknowledges any limitations encountered, and outlines potential avenues for future research and improvement. Additionally, it offers concluding remarks on the significance of the research and its implications for the automotive industry.

**Appendix (Codes):**

The appendix contains code snippets, algorithms, and additional data analyses to support the findings presented in the thesis.

**References:**

The references section provides a comprehensive list of the sources cited in this thesis, with appropriate guidelines.

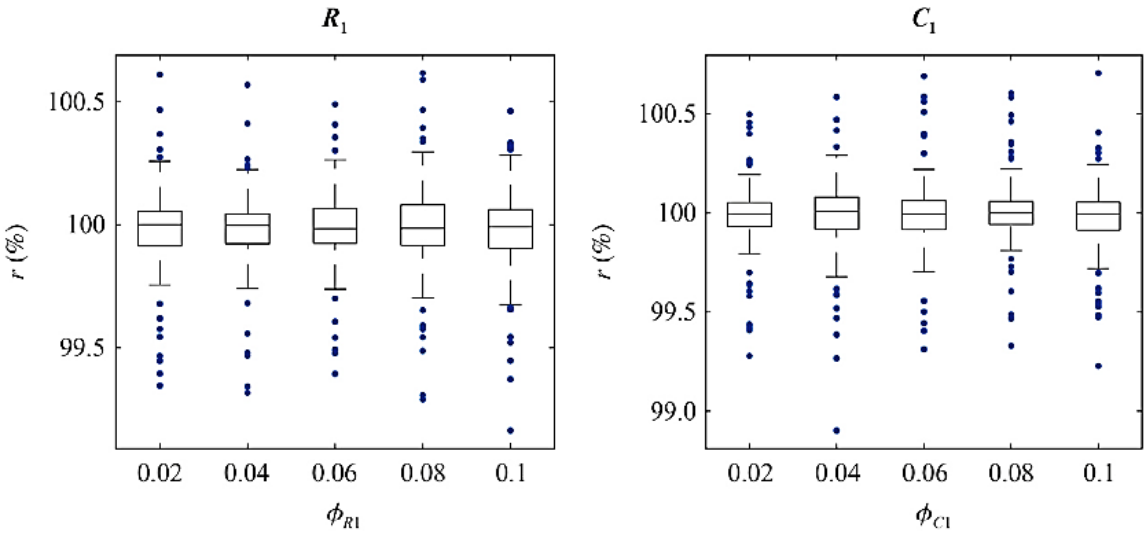
# Chapter 2

# Literature Review and Problem Formulation

## 2.1 Literature Review

The application of predictive analytics techniques in the domain of time series forecasting has garnered significant attention because of its applications in many fields such as weather forecasting, electrical power load forecasting, health monitoring, and intrusion detection [1]. Time series forecasting, as a fundamental aspect of predictive analytics, plays a critical role in understanding temporal data behavior and predicting future values, facilitating informed decision-making in diverse domains. The inherent sequential nature of time series dataset, where observations are captured over regular intervals of time, presents unique challenges and opportunities for predictive modeling. These challenges include capturing complex temporal patterns, handling missing data, and addressing non-stationarity and seasonality in the data.

Figure 3 shows the simulation results of *Jun Chen et al* [1] using the box plot of variation in vehicle range due to variation in cell capacity level obtained by modelling an equivalent circuit.



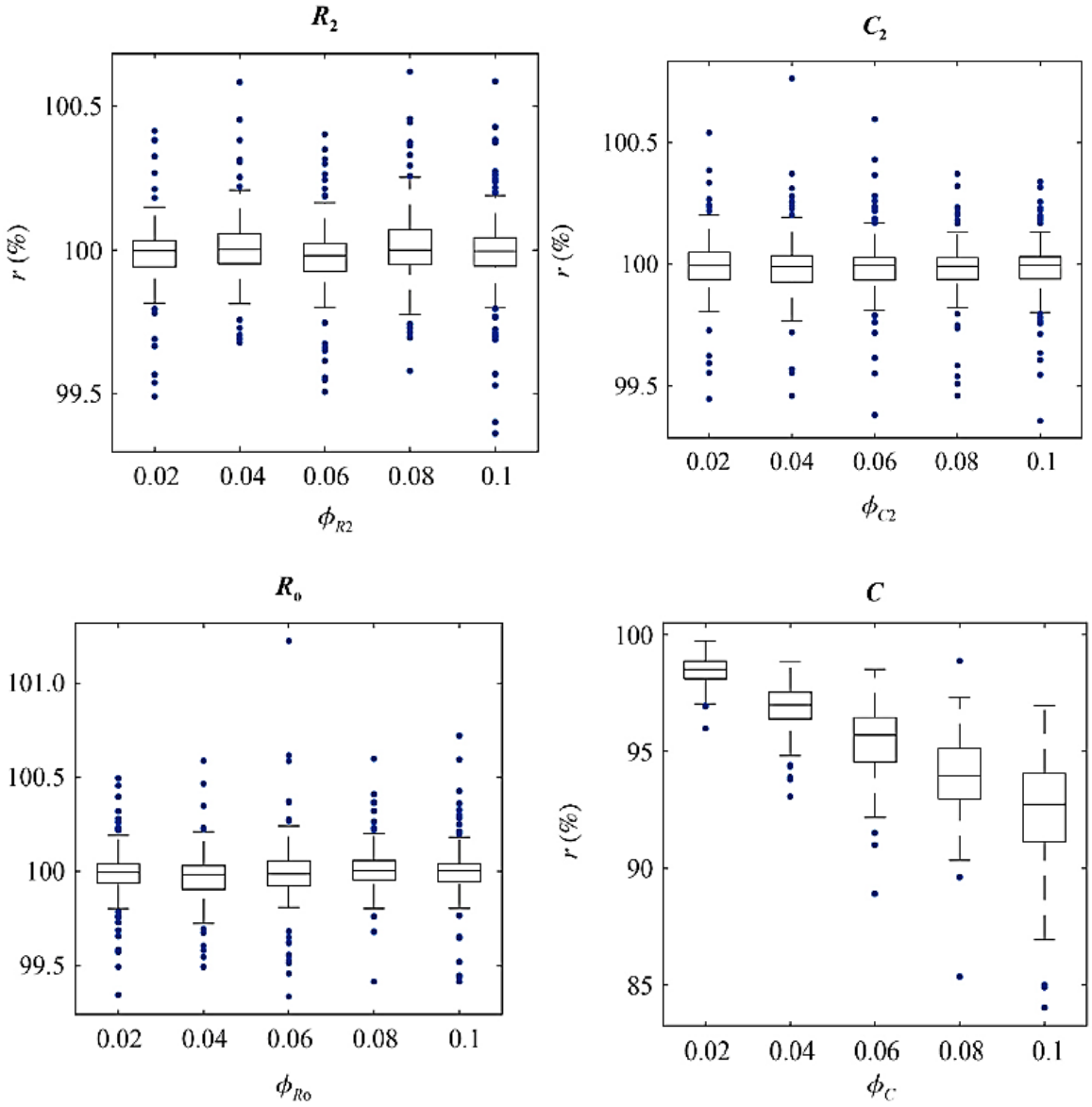



Figure 3 Range variation with respect to cell variation level

Notably, attention-based encoder-decoder frameworks have emerged as effective solutions for multivariate time series forecasting problems. *Du et al. (2020)* proposed a novel encoder-decoder model based on bi-directional LSTM networks (Bi-LSTM) with a temporal attention mechanism, demonstrating superior forecasting performance compared to baseline methods across multiple datasets [2]. This attention mechanism enables it to target applicable temporal information while generating forecasts, thereby capturing long-term dependencies and hidden correlation features in multivariate time series dataset. By leveraging the temporal attention mechanism, the model can adaptively learn and incorporate important temporal patterns, leading to more accurate and reliable forecasts.

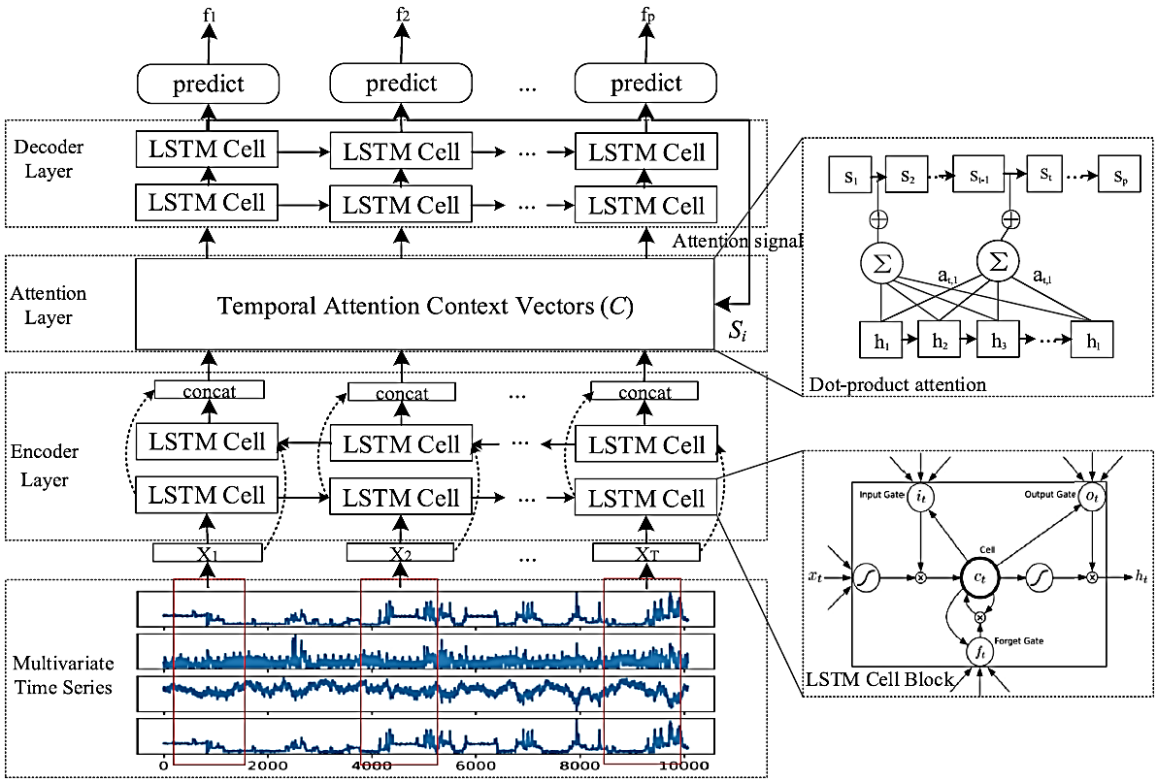



Figure 4 Framework of Multivariate Time Series Forecasting

Figure 4 shows a new model framework with attention towards temporal encoder and decoder developed by *Shengdong Du et al* [2] based on two-dimensional LSTM model which yielded better results to the multivariate time series data.

Furthermore, advancements in recurrent neural networks (RNNs) have addressed key limitations in current models for multivariate time series multi-step forecasting [3]. *He et al. (2023)* introduced Multi-Attention Collaborative Network (MACN) with a triangle-structure, incorporating an encoder-decoder framework with attention-based and a secondary hierarchical network to improve forecasting accuracy by capturing relevant variables and temporal dependencies [4]. This innovative approach enhances the interpretability and scalability of forecasting models for multivariate time series data, offering a robust solution for complex forecasting tasks. By integrating attention mechanisms at multiple levels of the model architecture, MACN can effectively capture short-term and long-term temporal dependencies together, with more accurate and reliable forecasts.

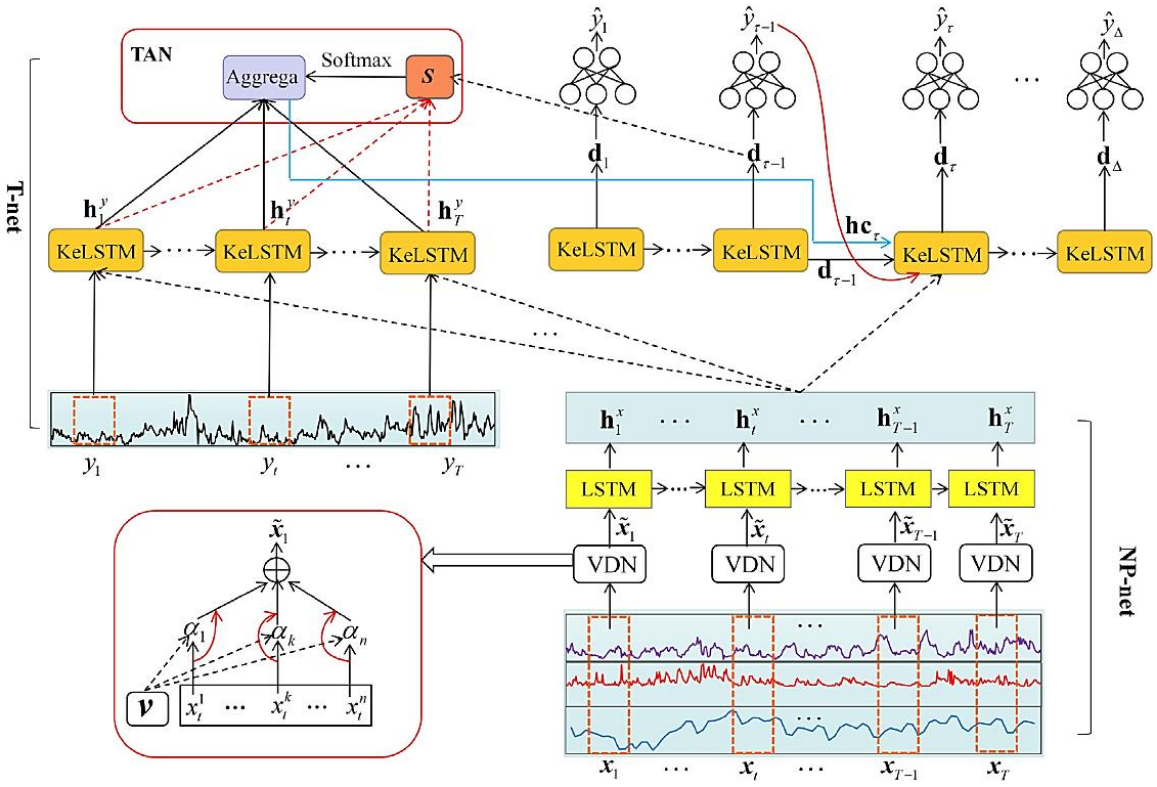
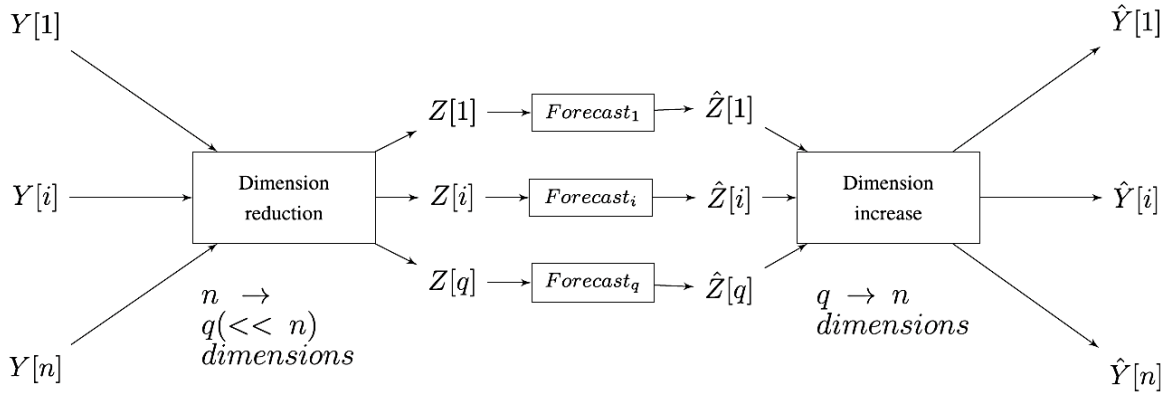



Figure 5 Architecture of Collaborative Network based on multi-attention

Figure 5 shows a framework model proposed by *Xiaoyu He et al* [4] based on multi-attention model collaborating with attention-based variables distillation network and LSTM model. Encoding and decoding of the network was done by a knowledge enhanced long short-term memory model and the results of the model outperformed existing state of the art models.

Despite the promise of deep learning models in multivariate forecasting tasks, they often face challenges in scalability, interpretability, and computational efficiency. To address these limitations, *De Stefani and Bontempi (2021)* proposed an approach with an extension to the Dynamic Factor Model (DFM), combining linear factor and non-linear factor evaluating techniques for large-scale multivariate forecasting tasks [5]. By leveraging the strengths of both linear factor and non-linear factor evaluating techniques, this approach enhances the interpretability and computational efficiency of multivariate time series forecasting models, making them suitable for real-world applications. Additionally, the DFM approach provides insights into the underlying factors driving the observed temporal patterns, enabling better understanding and interpretation of the forecasting results.




| | Dimensionality reduction | Forecast |
|---|---|---|
| **DFM** | PCA | VAR |
| **DFML** | PCA/Autoencoder | Statistical/ML-based |

Figure 6 Architecture of Dynamic Factor Model (DFM) and Machine Learning (DFML)

Figure 6 shows an extension model proposed by *Jacopo De Stefani et al* [5] based on Dynamic Factor Model framework to improve forecasting accuracy of the model. The experimental results also show an increase in computational efficiency and forecasting accuracy.

In the context of electric vehicle (EV) battery health prediction, machine learning-based approaches have gained traction for developing sturdy state-of-health (SOH) prediction models. *Akbar et al. (2022)* utilized a data-driven modeling strategy, incorporating Big Data, Artificial Intelligence (AI), and the Internet of Things (IoT) to develop an accurate and dependable SOH prediction model, demonstrating high accuracy in real-world scenarios [6]. Additionally, *Li et al. (2020)* proposed a method for battery life estimation based on cloud data, utilizing charging data to forecast battery cell capacity and its impedance, with errors of less than 4% [7]. These advancements in machine learning-based approaches enable proactive maintenance and optimization of battery performance, enhancing the reliability and efficiency of EVs.

Figure 7 shows the proposed model by *Kai Li et al* [7] based on optimizing the estimated results by Kalman filter and usage of Fuzzy logic to control the noise observed to increase the accuracy of the model. The simulation results of the cloud data shows that the estimated battery life has less than 4% error based on this new approach.

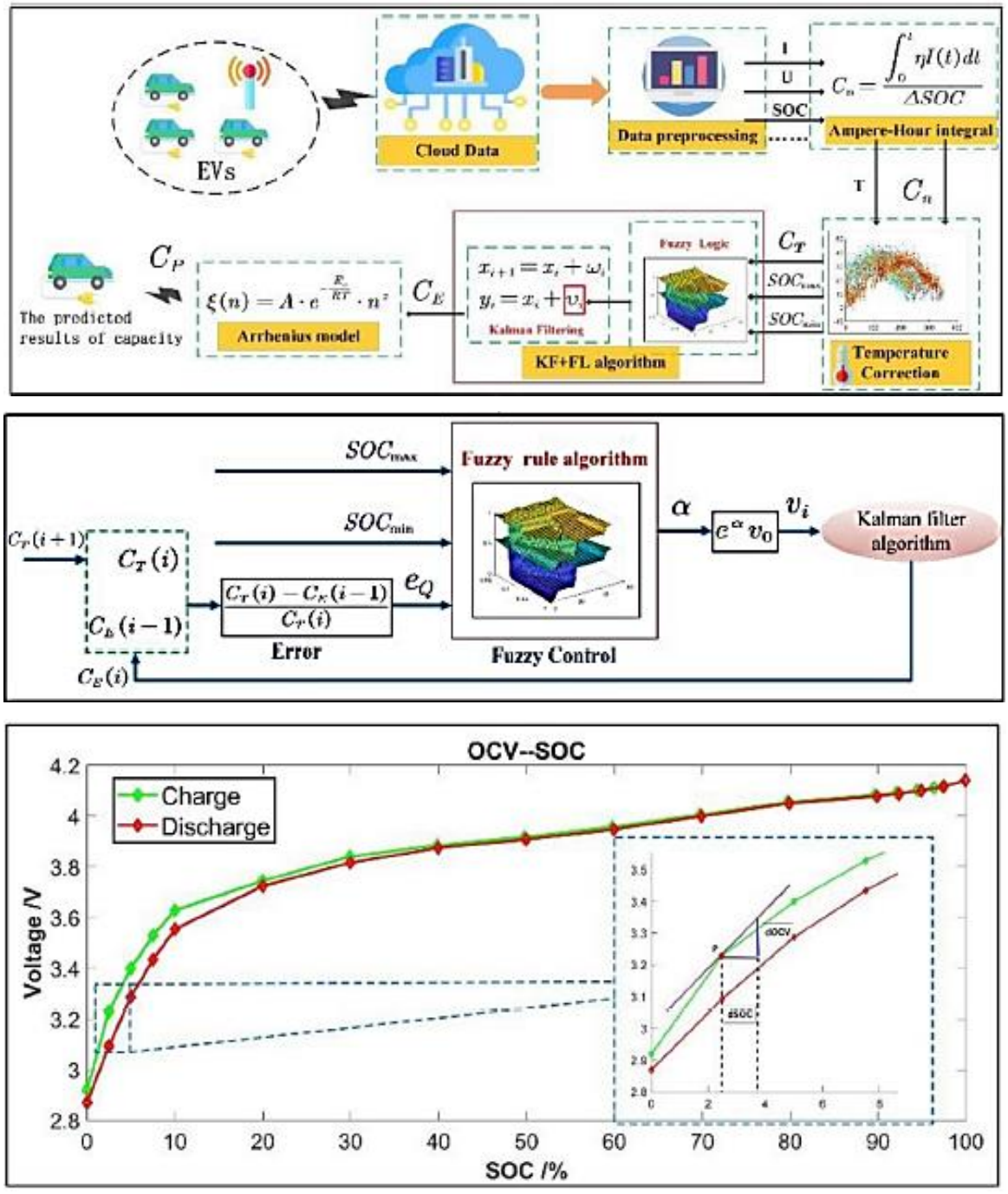



Figure 7 Analysis flowchart of Fuzzy Logic and Kalman Filter model

Moreover, ML techniques were used to monitor and predict the performance of internal combustion engines. *Kulkarni et al. (2021)* developed an ML model for detecting and live monitoring of engine oil aeration with a single high-speed oil pressure sensor, achieving high prediction accuracy [8]. By leveraging machine learning algorithms, such as Gaussian process regression, this approach enables live monitoring and detection of engine oil aeration, facilitating timely maintenance and optimization of engine

performance. These machine learning-based solutions contribute to improving the reliability, efficiency, and longevity of internal combustion engines, ultimately leading to cost savings and enhanced operational performance.

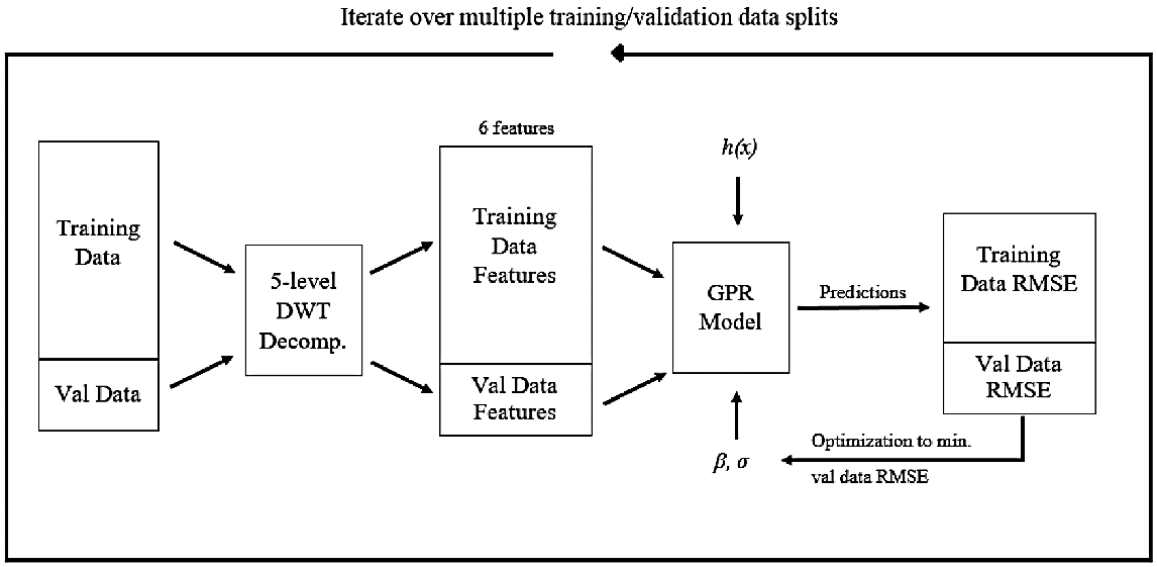



Figure 8 Gaussian Process Regression model training flowchart for prediction

Figure 8 shows the proposed model by *Vainatey Kulkarni et al* [8] based on a five-level discrete wavelet transform (DWT) and gaussian process regression (GPR) machine learning model. The predicted results show that uncertainty of the oil aeration values is under ± 0.02.

## 2.2 Problem Formulation

The results reported in the literatures underscore the importance of predictive analytics in addressing various challenges related to time series forecasting, battery health prediction, and engine performance monitoring in the context of electric and IC engine vehicles. Building upon these advancements, the present project aims to develop predictive analytics models to address specific use cases identified in collaboration with VECV.

By leveraging machine learning and logic-based approaches, the project seeks to enhance the reliability, efficiency, and performance of electric and IC engine vehicles, contributing to the advancement of predictive maintenance strategies in the automotive industry.

# Chapter 3

# Methodology

## 3.1 Understanding the Dataset

Time-series dataset represents a collection of parameters observed over a consistent time interval, typically in chronological order. In the context of this project, time-series data encompasses various parameters relevant to Electric and IC engine vehicles, such as battery voltage, temperature readings, motor performance metrics, vehicle performance parameters, and engine diagnostics. These observations are gathered at regular intervals, providing insights into the vehicle's operational state and performance over time. Unlike static datasets commonly encountered in traditional machine learning tasks, time-series data introduces unique challenges and opportunities due to its temporal nature.

Time-series data necessitates specialized handling and analysis techniques to extract meaningful insights and patterns. Trend analysis, for example, involves identifying long-term patterns or tendencies in the data, such as overall growth or decline over time. Seasonality decomposition aims to separate the data into seasonal components, allowing for the isolation and analysis of recurring patterns or cycles within the dataset. Autocorrelation analysis examines the correlation between observations at different time lags, helping to identify temporal dependencies and predictability in the data.

One of the key distinctions between time-series data and traditional tabular datasets lies in the sequential dependencies and temporal trends inherent in the former. Static dataset observations are independent of each other, and time-series dataset exhibits sequential relationships, where the value of a given observation may depend on its past values. This temporal structure necessitates the adoption of specialized modeling approaches tailored to capture and exploit these sequential dependencies effectively. In summary, understanding time-series data involves recognizing its temporal nature, identifying patterns and trends through specialized analysis techniques, and acknowledging the sequential dependencies inherent in the data. By leveraging appropriate modeling approaches and analysis tools, researchers and practitioners can extract valuable insights and make informed decisions in various domains, including vehicle performance monitoring and predictive maintenance.

## 3.2 Understanding the Use Cases

### 3.2.1 Battery Cell Imbalance

Cell balancing of a battery is a fundamental aspect of Battery Management Systems (BMS), particularly in EVs, where multiple cells are interconnected to make a battery pack. The primary objective of cell balancing is to establish that all the cells in the battery pack have similar voltage levels, thereby optimizing overall performance, capacity, and lifespan. Imbalances in cell voltage can lead to various detrimental effects, including reduced energy capacity, accelerated degradation, and safety risks.

Several factors contribute to cell voltage imbalances within a battery pack. Cell aging is a significant factor, where variations in cell chemistry and internal resistance occur over time, resulting in discrepancies in voltage levels among cells. Additionally, differences in manufacturing tolerances, temperature gradients within the battery pack, and variations in charging and discharging rates can also contribute to cell imbalances.

The effects of cell voltage imbalances on battery performance and longevity are profound. During the charging process, overcharged cells may lead to capacity loss, overheating, and safety hazards such as thermal runaway, while undercharged cells may experience reduced energy capacity and premature aging during discharge. To mitigate these imbalances, various balancing techniques are employed. Passive balancing methods involve dissipating excess energy from overcharged cells using shunt resistors or bypass diodes, thereby equalizing cell voltages. In contrast, active balancing methods redistribute charge between cells through external circuitry or balancing circuits, using techniques such as charge transfer, energy transfer, or voltage conversion.

The process of cell balancing occurs iteratively during the charging process, where balancing systems monitor individual cell voltages and activate mechanisms to equalize cell voltages. This iterative process continues until all cells are within the desired voltage range, ensuring optimal performance and longevity of the battery pack. Understanding balance and balancing in battery systems is crucial for maximizing performance and longevity. *Tom Wicker* highlights the significance of considering differences in charging/discharging due to variations in cell internal resistance, emphasizing the need for larger balancing currents to address such imbalances [9]. Balancing primarily focuses on equalizing State of Charge (SOC) levels among cells and

compensating for cell-to-cell variations in leakage but may be hindered by variations in cell resistance.

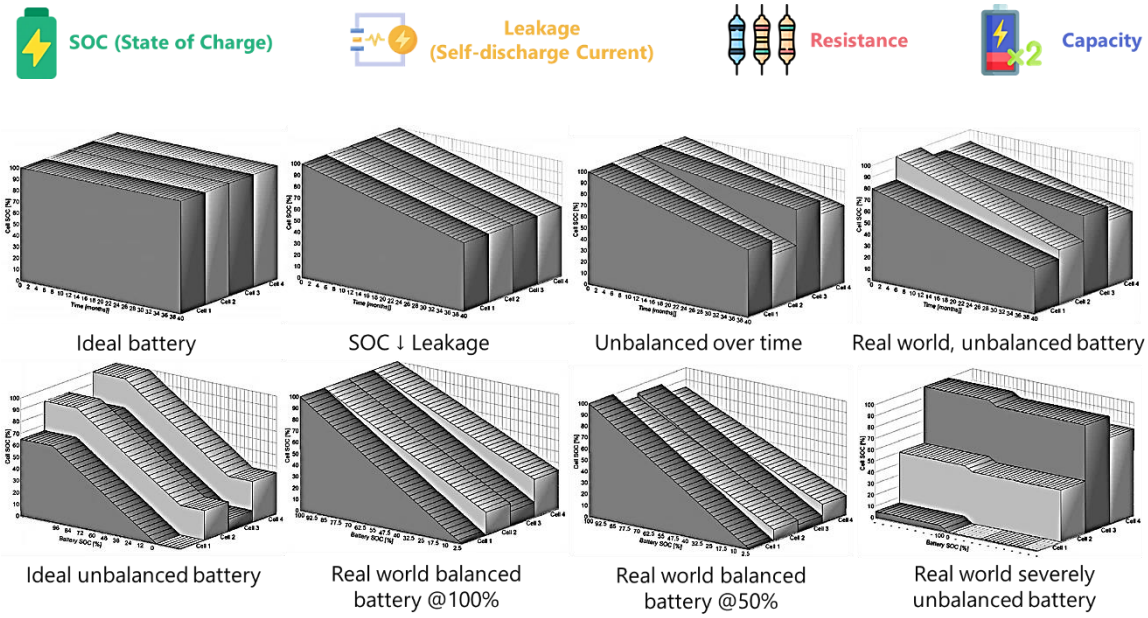



Figure 9 Cell Imbalance in a Battery Pack and its causes

Figure 9 shows various types of imbalances in a battery pack in real-world conditions and the possible reasons leading to imbalances such as SOC, leakage current, internal resistance (impedance), and cell capacity. An ideal battery will have full capacity for an infinite period if left unused, but current leakage happens due to impedance. The leakage is not uniform and varies from cell to cell. A battery can be balanced at 50% SoC or 100% SoC based on the program in the BMS. The common standard in the automotive industry is balancing at 100% SoC.

### 3.2.2 Battery Cell Temperature

Battery cell temperature management is a vital aspect of lithium-ion battery systems, especially in automotive applications, where it directly impacts performance, safety, and longevity. Several influential parameters contribute to cell temperature variations, with environmental conditions being one of the primary factors. Environmental factors such as ambient temperature, humidity levels, and altitude can significantly influence cell temperature. In extreme temperatures, whether hot or cold, the performance and life of the battery can be compromised. Cold temperatures increase internal resistance and reduce battery capacity, leading to decreased energy output, particularly during cold starts in electric vehicles (EVs). Conversely, high temperatures

accelerate chemical reactions within the cells, resulting in accelerated degradation mechanisms such as electrode corrosion, electrolyte decomposition, and formation of Solid-Electrolyte Interphase (SEI) layers. This degradation leads to capacity fade, reduced energy efficiency, and ultimately diminishing the reliability of the battery pack.

Moreover, high temperatures can trigger thermal runaway events, causing safety hazards such as cell venting, fire, or explosion. Conversely, low temperatures increase internal resistance and reduce ion mobility, limiting charge and discharge rates, which affects power output, energy capacity, and regenerative braking performance, particularly in cold climates. In addition to environmental conditions, high discharge currents common in EVs during acceleration or heavy loads can generate significant heat within the battery cells. The rapid flow of current increases internal resistance and heat dissipation, elevating cell temperatures. Similarly, overcharge or over-discharge events, often caused by charging or discharging beyond recommended levels, result in excessive heat generation, posing safety risks and accelerating battery aging.

To mitigate the adverse effects of temperature extremes on battery performance and lifecycle, battery thermal management systems are employed. These systems utilize heating and cooling mechanisms controlled by the BMS to maintain cells within an optimal temperature range, typically between -20°C to +60°C for lithium-ion cells. Heating mechanisms ensure cells remain above the minimum operating temperature, preventing performance degradation and enhancing energy efficiency, particularly in cold climates. Cooling mechanisms prevent cells from exceeding their maximum operating temperature, mitigating safety risks, and extending battery life.

Effective thermal management strategies tailored to specific environmental conditions are crucial for the battery pack. By maintaining cells within the recommended temperature range, automotive battery systems can operate efficiently, ensuring reliable performance, enhanced safety, and prolonged lifespan, ultimately contributing to the overall sustainability and viability of electric vehicles.

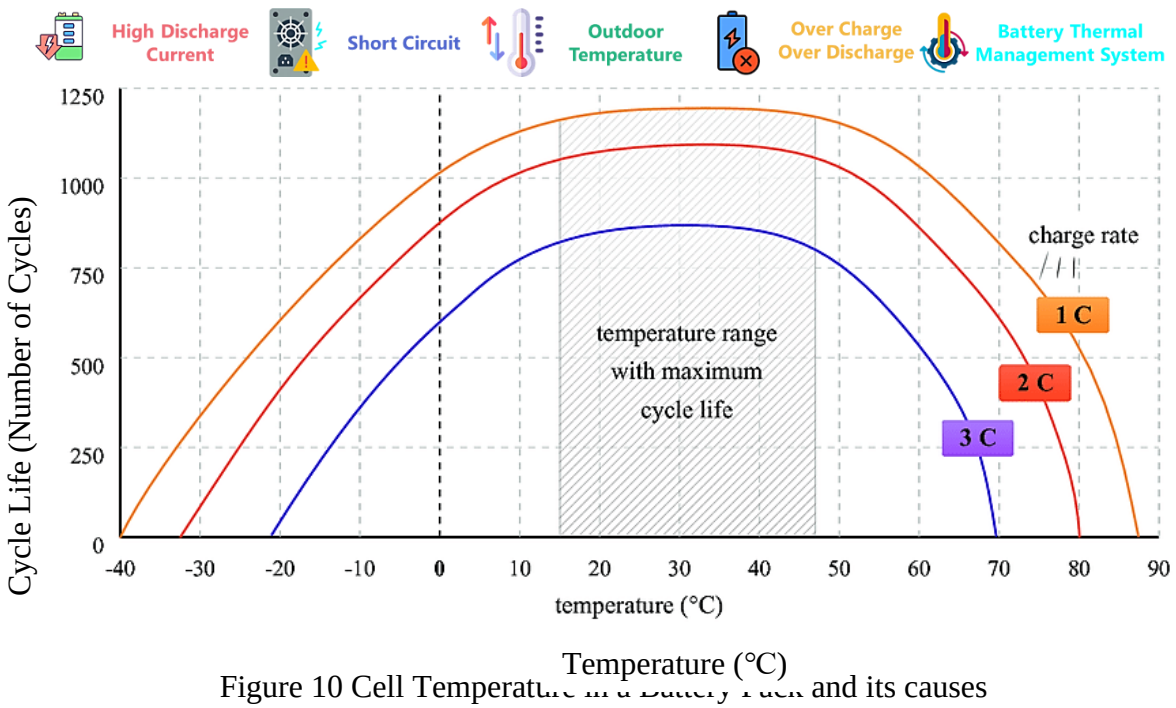



Figure 10 Cell Temperature in a Battery Pack and its causes

Figure 10 shows the possible reasons leading to cell temperatures such as overcharge, over-discharge, high discharge current, and thermal management system. The optimum temperature operating range for the maximum life cycle of the battery is between 15℃ to 45℃. The reduction in capacity retention against the number of cycles is shown in the graph for different cell temperatures with high temperatures affecting the capacity at a higher rate.

Figure 11 shows the data transfer from the battery to the Vehicle Control Unit (VCU). The octillion battery pack has 208 cells (8 columns) connected in series and the cells have individual voltage and temperature measurement sensors. The individual column data are shared with its slave Battery Management System (BMS) and all slave BMS data are consolidated in Master BMS. The necessary actions are taken based on the condition of the individual cell's voltage and temperature. Maximum and Minimum temperature and voltage values are shared with the Telematics Control Unit (TCU) to upload on the server. The data of both cell voltage and temperature are shared through the BMS.

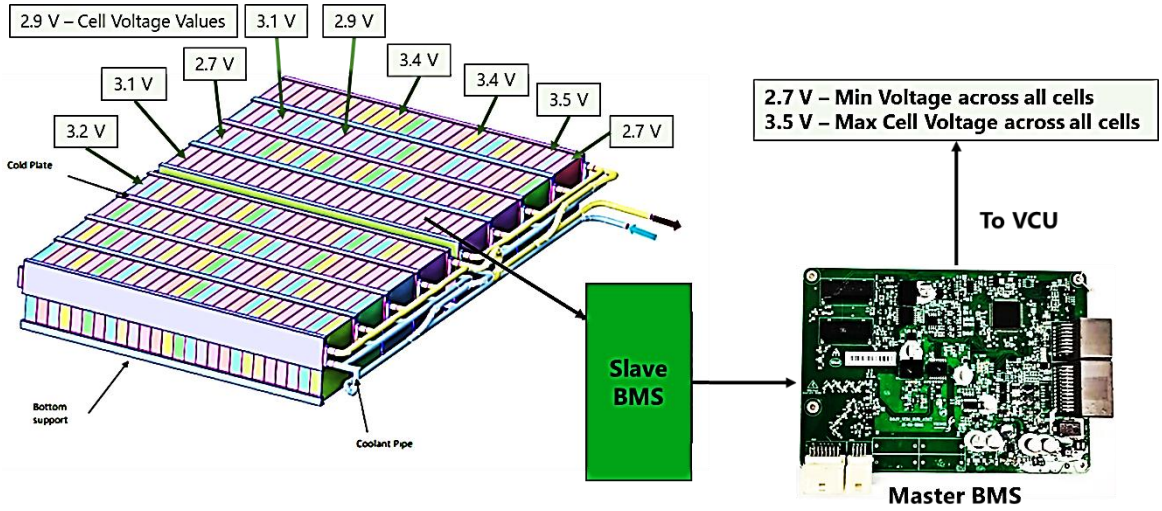



Figure 11 Octillion Battery Pack with Slave and Master BMS

### 3.2.3 Motor Temperature

Electric vehicle traction motors, particularly Permanent Magnet Synchronous Motors (PMSM), are critical components that require careful temperature management to ensure optimal performance and reliability. The temperature of the PMSM motor is influenced by various factors, including motor overload, inconsistent power delivery, frequent start-stop operations, charging current during regenerative braking, and the effectiveness of the thermal management system.

Motor overload, occurring when the motor is subjected to excessive torque demands or prolonged high-speed operation, results in increased heat generation within the motor. This can lead to temperature spikes, causing thermal stress on motor components and potentially leading to overheating-related failures. Inconsistent power delivery, often caused by voltage fluctuations or irregularities in the powertrain system, can also impact motor temperature. Rapid changes in power demand or voltage levels can lead to thermal cycling of the motor, resulting in temperature variations and potential performance degradation.

Frequent start-stop operations, commonly encountered in urban driving conditions, can contribute to motor temperature fluctuations. Each start-stop cycle subjects the motor to thermal stress, as it rapidly transitions between stationary and operational states. Over time, this can lead to cumulative heat buildup and increased risk of overheating-related issues. During regenerative braking, the motor operates in

generator mode, converting kinetic energy into electrical energy to recharge the battery. The charging current flowing back into the battery generates heat within the motor windings, contributing to temperature elevation. Without proper thermal management, excessive heat buildup can occur, leading to thermal runaway and potential motor damage.

Motor temperature varies with the current flowing through the motor windings and the voltage applied across them. Higher currents result in increased Joule heating within the motor windings, raising the temperature. Similarly, higher voltages can lead to increased heat dissipation, particularly during rapid acceleration or high-speed operation.

High motor temperatures can have detrimental effects on performance and reliability. Excessive heat can degrade insulation materials, leading to insulation breakdown and short circuits. It can also cause demagnetization of the permanent magnets, reducing motor efficiency and torque output. Additionally, high temperatures can accelerate bearing wear and lubricant degradation, leading to premature failure and increased maintenance requirements. Conversely, low motor temperatures can also impact performance negatively. Cold temperatures can increase motor winding resistance, reducing efficiency and power output. It can also increase friction and wear on mechanical components, such as bearings and gears, leading to decreased reliability and increased energy consumption.

To manage motor temperatures effectively, EVs employ various thermal management strategies. Active cooling systems, such as liquid cooling or air cooling, are commonly used to dissipate heat generated during motor operation. These systems circulate coolant or air through channels or passages within the motor housing, removing heat and maintaining optimal operating temperatures. The optimum temperature range for PMSM motors typically falls between 40 to 80°C. Operating within this temperature range ensures efficient motor performance, longevity, and reliability. Deviations from this range can result in decreased efficiency, increased wear, and potential motor failures.

Figure 12 shows the possible reasons leading to high motor temperatures such as motor overload, poor power, and thermal management system. A correlation heatmap matrix shows the significance of individual parameters in motor temperature.

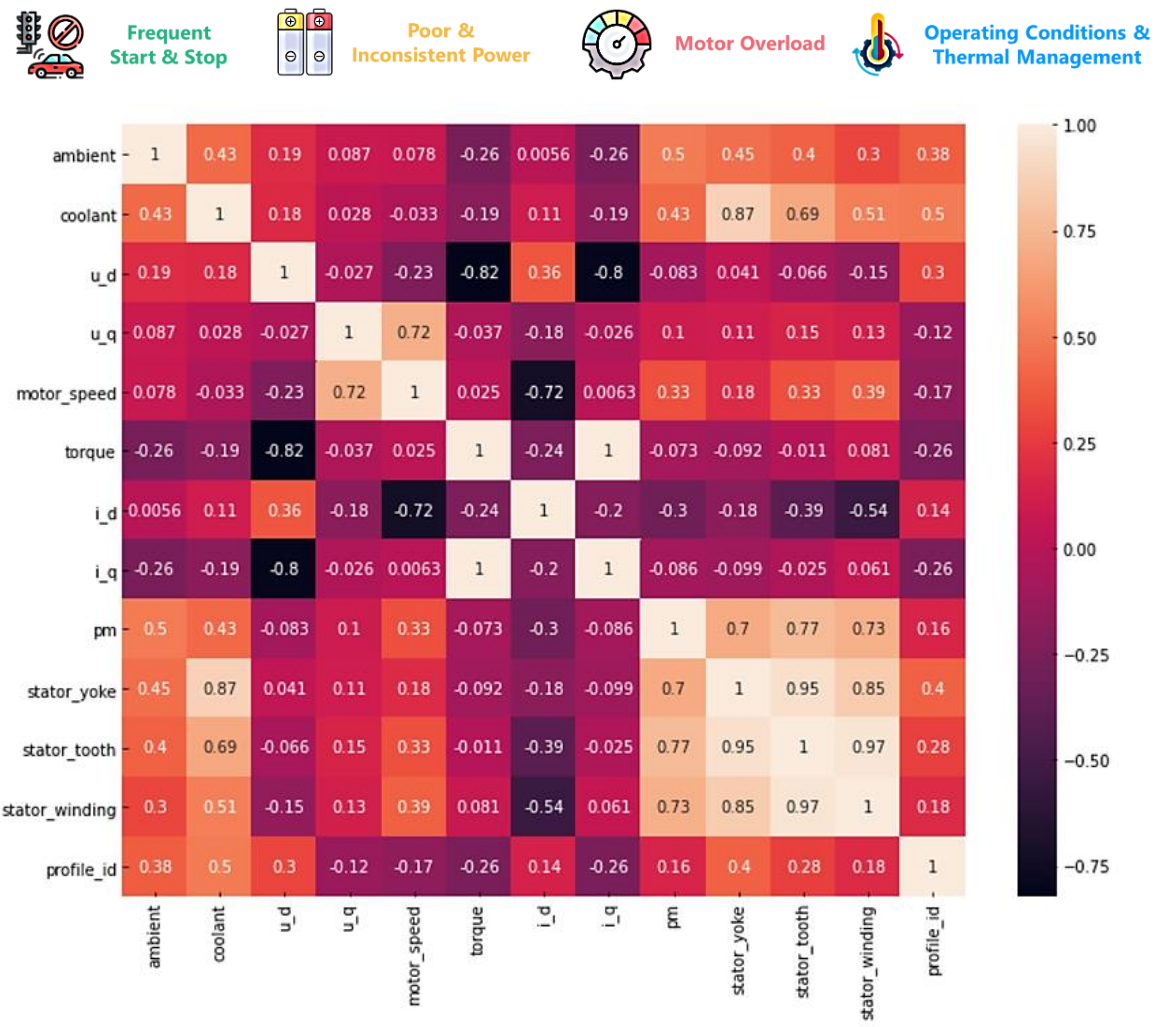



Figure 12 EV Motor Temperature causes and Correlation Heat Map Matrix

### 3.2.4 Engine Oil Pressure

Low engine oil pressure is a critical issue that can lead to severe damage and malfunctions within an internal combustion engine. Several influential parameters contribute to low engine oil pressure, including engine RPM, engine oil viscosity, a plugged or clogged oil filter, low oil level, apparent low pressure caused by worn bearings or oil passages, and pump wear.

Engine RPM plays a significant role in determining oil pressure, as higher RPMs result in increased oil demand to lubricate and cool engine components. At low RPMs, oil pressure tends to decrease due to reduced oil flow rate through the engine's oil passages. This can be particularly evident during idling or low-speed operation, where insufficient oil circulation can lead to low-pressure conditions.

Engine oil viscosity, or the thickness of the oil, also influences oil pressure. Higher viscosity oils provide better lubrication at high temperatures and pressures, resulting in higher oil pressure. Conversely, lower-viscosity oils may experience reduced oil pressure, especially in high-temperature operating conditions, where the oil thins out and flows more easily. A plugged or clogged oil filter can restrict oil flow, causing a drop in oil pressure. Over time, debris, contaminants, and sludge can accumulate in the oil filter, impeding oil circulation and reducing pressure. Regular oil and filter changes are essential to prevent filter blockages and maintain optimal oil flow and pressure.

Low oil level is another common cause of low engine oil pressure. Insufficient oil volume in the crankcase reduces the amount of oil available for lubrication, leading to decreased oil pressure. Monitoring oil levels regularly and topping up as needed is crucial for preventing low oil pressure conditions. Apparent low pressure, often caused by worn engine bearings, oil pump wear, or blocked oil passages, can give the impression of low oil pressure even when oil levels and viscosity are adequate. Worn bearings create larger gaps between moving parts, allowing oil to flow more freely and reducing pressure. Similarly, pump wear can result in a decreased oil flow rate, leading to lower pressure readings.

Low engine oil pressure can have detrimental effects on vehicle performance and engine life. Inadequate lubrication can cause increased friction and wear on engine components, leading to premature engine failure and reduced longevity. Severe cases of low oil pressure can result in engine overheating, seizure, or catastrophic failure, necessitating costly repairs or replacements.

To manage low engine oil pressure effectively, regular maintenance and inspections are essential. This includes checking oil levels and quality, replacing oil filters at recommended intervals, and monitoring oil pressure using a gauge or warning light. Addressing any issues promptly, such as leaks, worn bearings, or pump wear, can help prevent low oil pressure conditions and ensure optimal engine performance and longevity.

Figure 13 shows the possible reasons leading to low oil pressure in the engine such as oil viscosity, engine RPM, plugged filter, low oil level, pump wear, and apparent low pressure.

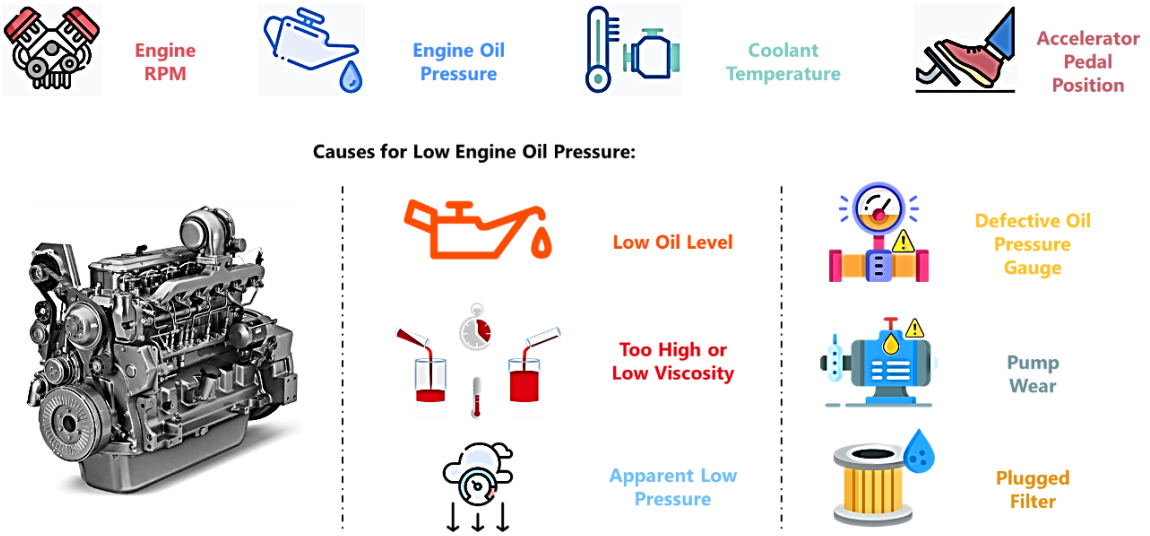



Figure 13 Low Oil Pressure in Engine and its causes

## 3.3 Methodology Flow Chart

### 3.3.1 Preprocessing the Data

Techniques such as interpolation or forward/backward filling are used to remove missing values, ensuring continuity in the time-series dataset. Statistical methods or machine learning algorithms are employed to detect outliers, which may indicate sensor malfunctions or abnormal vehicle behavior. Exploratory Data Analysis (EDA) techniques, including time-series decomposition and trend visualization, are used to understand the data distribution and identify patterns. Relevant features such as rolling averages, lagged variables, and seasonality indicators are engineered to capture important information for predictive modeling. Data transformation techniques such as logarithmic scaling or differencing are applied to stabilize variance and make the data more amenable to modeling.

### 3.3.2 Splitting the Data

Data is traditionally divided into three categories for training, validation, and testing using a predetermined ratio (e.g., 70% training, 15% validation, 15% test) to ensure model generalization and prevent overfitting. Cross-validation is performed using K-fold to predict the performance of the model across various subsets of the data, providing insights into its robustness and stability.

### 3.3.3 Data Analysis

Exploratory data analysis techniques such as correlation analysis and feature importance ranking are used to identify relevant variables and assess their impact on model performance. Statistical models like ARIMA and VAR are used for time-series forecasting, while ML models like XGBoost and LSTM are employed for complex pattern recognition and prediction tasks.

### 3.3.4 Machine Learning Model

Various statistical models and machine learning algorithms, including VAR, ARIMA, XGBoost, LGBM, and LSTM neural networks, are explored and implemented to address the specific use cases. A model will be selected based on its suitability for the specific use case and dataset characteristics. Model hyperparameters are tuned using grid search or random search to optimize performance.

### 3.3.5 Evaluation

The metrics Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R squared ( $R^2$ ) are used to assess model accuracy and predictive performance. Performance analysis based on these metrics helps identify areas for improvement, such as feature selection, hyperparameter tuning, or data augmentation.

### 3.3.6 Model Deployment

Trained models are deployed in production environments to make real-time predictions on new data, enabling proactive maintenance and decision-making. Inference mechanisms interpret model predictions and provide actionable insights to stakeholders, facilitating informed decision-making and operational optimization.

Figure 14 shows the methodology adopted and the subsystems involved. This comprehensive methodology outlines the entire process of leveraging predictive analytics for electric and IC engine vehicles. Each step is tailored to address the unique challenges and requirements, ensuring robust and scalable solutions for vehicle monitoring and maintenance.

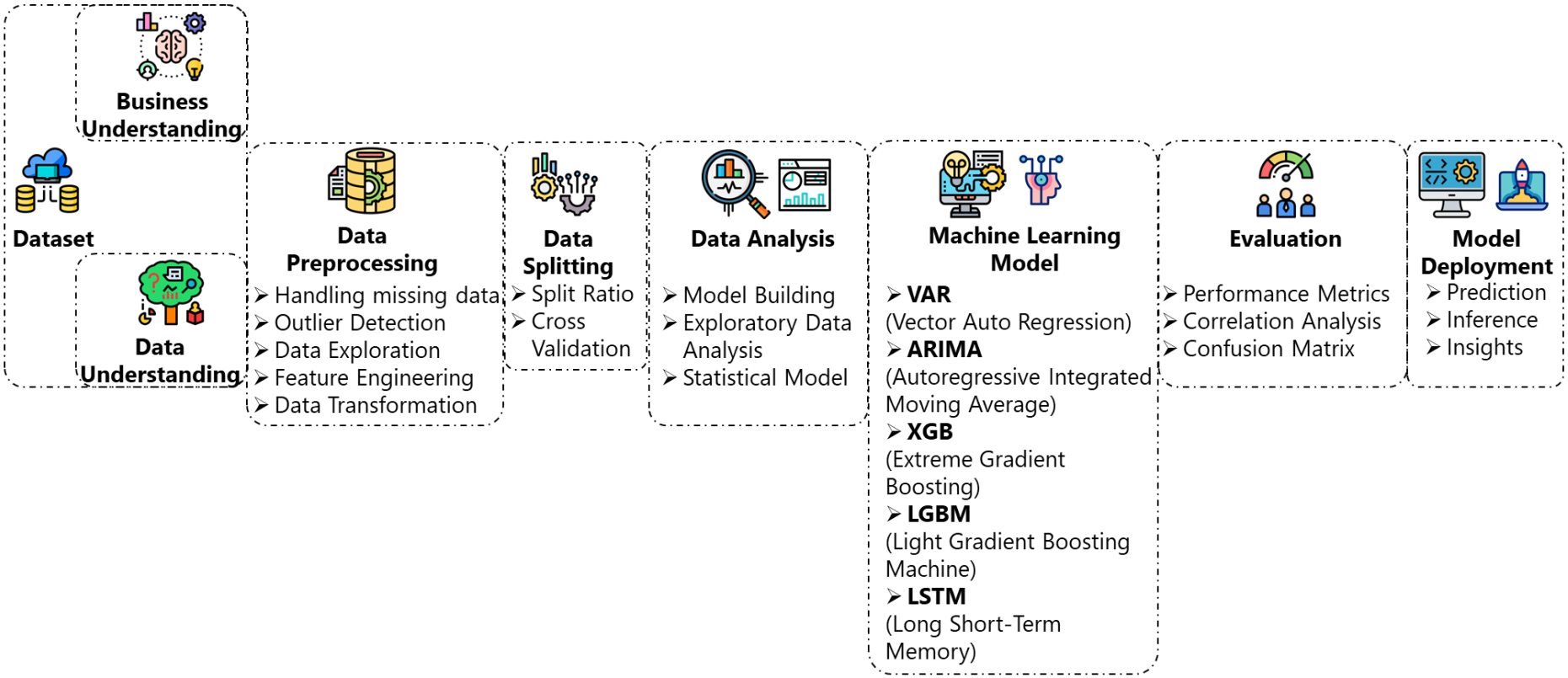



Figure 14 Methodology Flow Chart

# Chapter 4

# Machine Learning Models

Machine learning is a field of artificial intelligence that enables computers to learn from data and make predictions without explicit programming. In automotive predictive diagnostics, ML analyzes data from vehicle sensors to predict faults, detect anomalies, optimize maintenance schedules, and enhance diagnostic accuracy. By identifying patterns in metrics, ML models can forecast issues before they occur, allowing for timely maintenance and reducing breakdowns. This approach improves vehicle reliability, extends lifespan, and provides personalized insights based on driving habits, ultimately optimizing vehicle performance and maintenance efficiency. ML models experimented in this study are explained in detail with their underlying principles, suitability for time-series data, potential advantages, and limitations in this section. The flowchart adapted for it is shown in Figure 15. The dataset is preprocessed, features are selected appropriately based on the use case, correlation analysis is done to verify the significance of the features, and then the dataset is normalized. The normalized data is divided into three subsets for training, validation, and testing. The trained model is saved, and its performance is evaluated based on evaluation metrics.

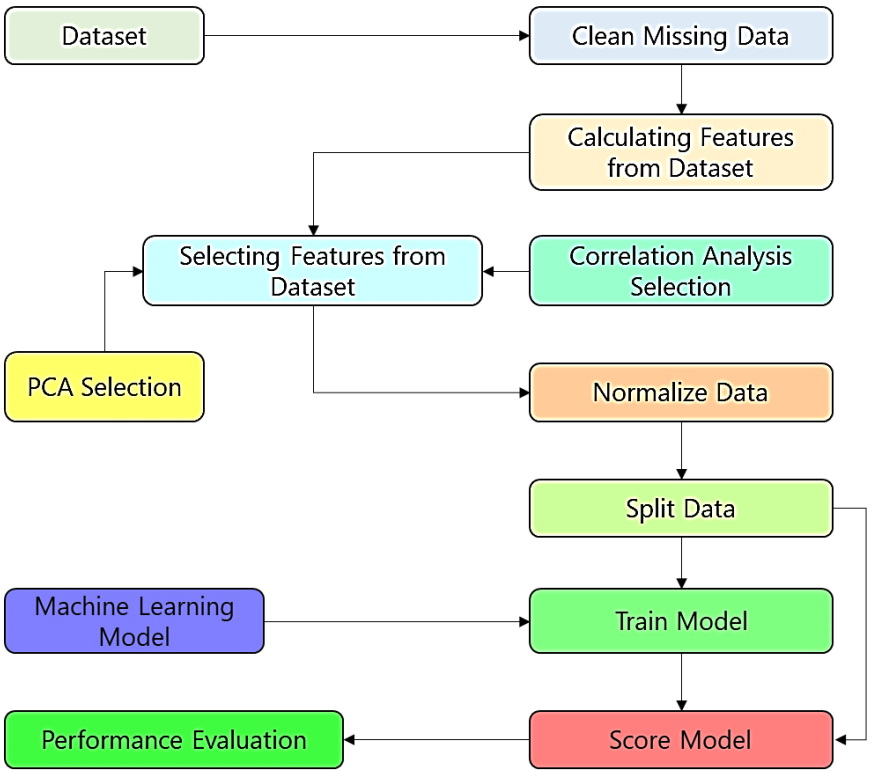



Figure 15 Machine Learning Model Flow Chart

## 4.1 Vector Auto Regression (VAR)

VAR is a multivariate time-series forecasting model that extends autoregression (AR) to multiple variables. It captures linear dependencies between multiple variables by modeling individual variable as a function of its previous values and the previous values of other variables in the system. VAR is suited for capturing dynamic relationships and feedback mechanisms present in multivariate time-series data. It can handle interdependencies between variables and capture their joint evolution over time.

Strengths of VAR is its capability to record dynamic relationships and feedback mechanisms within the data. Examining the effect of changes in one variable with the behavior of another variable over time furnishes useful insights into the underlying dynamics of the system. This makes it an influential tool for predicting multiple variables simultaneously, as it can report the interactions between them and their joint evolution over time. However, despite its versatility, VAR has certain limitations that must be considered when adapting it to real-world datasets. VAR assumes that the relationships between variables are linear. While this simplifying assumption allows for straightforward interpretation and estimation, it may not accurately capture nonlinear dependencies or complex patterns present in the data. In cases where the relationships between variables are highly nonlinear, VAR may produce suboptimal forecasts.

VAR performs best when applied to a static time-series dataset with a constant statistical property. In real-world datasets, achieving stationarity can be challenging, particularly when dealing with multivariate, multi-output data with class imbalance. Variations in data quality, missing values, or active trends can impact performance and lead to inaccurate forecasts. As the number of variables in the system increases, the model may become computationally intensive and prone to overfitting. In such scenarios, careful feature selection and regularization techniques are necessary to mitigate the curse of dimensionality and prevent model overfitting.

In datasets with class imbalance, where certain classes or categories are underrepresented, VAR may struggle to accurately capture the dynamics of minority classes. This results in biased forecasts and poor forecasting accuracy, particularly if the imbalance is severe. Specialized techniques, such as resampling methods or class-weighted loss functions, may be required to address class imbalance and improve the

robustness of the model. While VAR provides valuable insights into the relationships between variables, understanding the results of a multivariate analysis can be demanding, specifically in complex systems with numerous interdependencies.

## 4.2 Autoregressive Integrated Moving Average (ARIMA)

ARIMA is a widely used time-series forecasting model that blends autoregression (AR) and moving average (MA) components with changes to handle static data. ARIMA models can record both the linear dependencies within a time-series dataset and its elemental trend and seasonality patterns present in the dataset.

ARIMA models are characterized by three main components: Autoregression (AR), Integrated (I), and Moving Average (MA). AR component models the relationship between an observation and its former values. It captures the linear dependence between the current value of the time series data and its previous values. The integrated component involves differentiating the time series data to achieve stationarity. By taking differences between consecutive observations, ARIMA removes trends and other non-stationary patterns, making the time series static. The MA component models the relationship between a recorded value and the error residue term derived from a MA model applied to previous observations. It captures the linear dependence between the current value and past forecast errors.

The ARIMA model is inherently a univariate model and does not directly handle multiple input variables or outputs. While extensions like vector ARIMA (VARIMA) exist for multivariate time series, they may not effectively capture the complex interactions and dependencies present in multi-output data with class imbalance. It assumes linear relationships between variables and may struggle to model nonlinear dependencies or complex patterns in the data.

This model performs best when applied to stationary time series data. Achieving stationarity can be challenging in multivariate datasets with class imbalance, where variations in data quality, missing values, or non-stationary trends are common. It might give inaccurate predictions if the base data is not stationary. As the number of variables increases, ARIMA models may become computationally intensive and prone to overfitting. Feature selection and regularization techniques are necessary to mitigate overfitting and ensure model robustness. It also struggles to accurately capture the

dynamics of minority classes in datasets with class imbalance. This can lead to biased forecasts and poor predictive performance, particularly if the imbalance is severe.

## 4.3 Extreme Gradient Boosting (XGBoost):

XGBoost is a powerful ensemble learning algorithm with effectiveness in supervised learning tasks. It develops a series of decision trees sequentially, with each tree learning from the errors of its predecessors, thereby improving the long-term predictive performance of the model. It can be applied by framing the problem as a supervised learning task, where historical observations are used to predict future values. It excels at capturing complex nonlinear relationships and patterns in the data, making it suitable for modeling the dynamic behavior of time series data.

High performance, scalability, and efficiency are advantages of this model. It can handle large datasets with high dimensionality, making it suitable for tasks involving many features and observations. Additionally, it is less prone to overfitting compared to traditional decision trees, thanks to its regularization techniques and ensemble approach.

XGBoost requires careful tuning of hyperparameters to optimize its performance. Choosing the correct mix of hyperparameters, such as learning rate, tree depth, and regularization parameters, can be challenging and may require extensive experimentation. In multivariate, multi-output time series datasets, the number of hyperparameters to tune increases, making the tuning process more complex. While it provides high predictive accuracy, its models can be less understandable compared to simpler models like ARIMA or VAR. The ensemble nature of XGBoost and the complex interactions between decision trees make it demanding to interpret the underlying relationships between input variables and output targets.

This model may struggle to predict minority classes in datasets with class imbalance accurately. The algorithm tends to focus more on optimizing overall accuracy, which can lead to biased predictions for minority classes. Techniques such as class weighting or resampling may be required to address class imbalance and improve model performance. It may still face scalability issues with extremely large datasets or complex models.

## 4.4 Light Gradient Boosting Machine (LGBM)

LGBM is a powerful ML algorithm used for supervised learning, particularly in the field of gradient boosting. It operates similarly to XGBoost but is optimized for efficiency and speed, suited for large-scale datasets and real-time applications. It constructs decision trees in a leaf-wise manner rather than level-wise, optimizing for the maximum reduction in loss at each step. This approach allows it to build trees more efficiently and achieve better performance with fewer splits, resulting in faster training times and lower memory usage.

One of the key advantages of LGBM is its scalability and efficiency. It can handle large datasets with millions of observations and thousands of features, making it suitable for tasks requiring high-dimensional data. Additionally, it is highly parallelizable and can take advantage of multi-core processors to accelerate training. It is also capable of handling categorical features and missing values effectively, eliminating the need for extensive data preprocessing. It automatically handles categorical features by encoding them into numerical values during training, and it can handle missing values by partitioning data based on missingness and treating them as separate categories.

When applied to time-series forecasting tasks, it offers similar advantages to XGBoost. It can record complex patterns and nonlinear relationships present in the dataset, is apt for modeling the dynamic behavior of time series data. However, it has limitations while dealing with multivariate, multi-output time series data with class imbalance. Like other ensemble methods, this model can be less interpretable compared to simpler models such as linear regression. The complex interactions between decision trees and the ensemble nature of the model make it challenging to interpret the underlying relationships between input variables and output targets.

LGBM requires careful tuning of hyperparameters to achieve optimal performance. Selecting the correct set of hyperparameters can be demanding and may need extensive experimentation. It may struggle to predict minority classes in datasets with class imbalance accurately. The algorithm tends to focus more on optimizing overall accuracy, which can lead to biased predictions for minority classes. Specialized techniques such as class weighting or resampling may be required to address class imbalance and improve model performance.

## 4.5 Long Short-Term Memory (LSTM) Neural Networks

LSTM Neural Network is based on recurrent neural network (RNN) type architecture explicitly designed to address the difficulties of securing long-term dependencies in sequential data. Dissimilar to traditional RNNs, which struggle to preserve data over long sequences due to the fading gradient problem, LSTM networks incorporate gating mechanisms and memory cells to carefully preserve and update the data over time. In time-series forecasting tasks, LSTM networks offer several advantages, specifically when dealing with multivariate, multi-output time series class imbalanced datasets.

LSTM networks are suited for modeling time-series dataset with long-term dependencies. They can effectively capture patterns and relationships that span across multiple time steps, allowing them to record both short term fluctuations and long-term trends in the dataset. This makes it effective for predicting tasks where understanding the historical context is crucial for making accurate predictions. This network can also easily accommodate multivariate time-series data, where multiple variables are observed simultaneously over time. By processing multiple input features concurrently, LSTM networks can record complicated dependencies and interactions between different variables with more accurate forecasts.

This model is capable of processing sequences of variable lengths, making them versatile for handling time-series data with irregular sampling intervals or missing observations. This flexibility allows it to adapt to the temporal dynamics of the data and effectively model sequences of different lengths. It also excels at capturing sequential patterns and temporal dynamics in the data. By learning from past observations and updating their internal state over time, it can capture subtle changes and nonlinear relationships in the dataset, enabling them to make skillful forecasts even in the presence of complex temporal patterns.

In multivariate, multi-output time series datasets with class imbalance, LSTM networks can adapt their learning process to focus more on minority classes by adjusting the loss function or incorporating class weighting techniques. This allows it to effectively handle imbalanced datasets and make accurate predictions for all classes. Despite these advantages, LSTM networks also have limitations, such as their susceptibility to

overfitting and the need for larger amounts of training data and longer training times compared to traditional statistical models. However, with careful regularization techniques, hyperparameter tuning, and appropriate preprocessing of the data, LSTM networks can overcome these limitations and deliver accurate forecasts in multivariate, multi-output time series class imbalanced datasets.

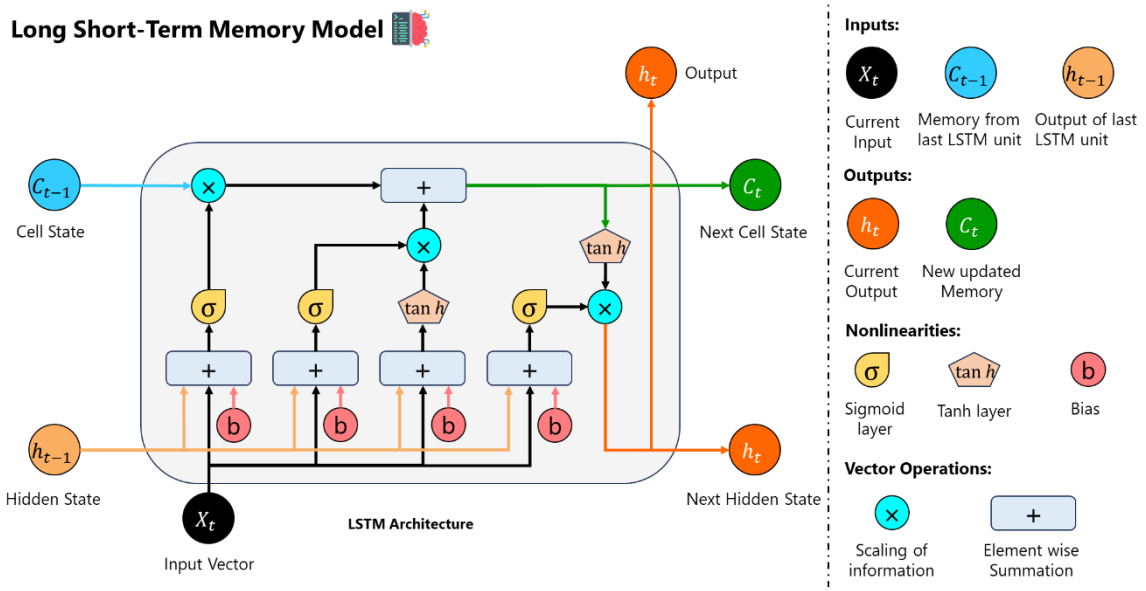



Figure 16 Long Short-Term Memory (LSTM) model architecture and its system

Figure 16 shows the architecture of an LSTM model and its nonlinearities, types of vector operations involved, inputs considered, and the outputs generated. The model updates every node in each layer by following this architecture to predict the final parameters.

## 4.6 Correlation Analysis

Correlation analysis determines the strength and direction of the linear relationship between variables in the dataset. It helps to analyze potential dependencies and patterns in the dataset. A positive correlation signifies a direct relationship between variables, a negative correlation signifies an inverse relationship, and a zero correlation indicates no linear relationship.

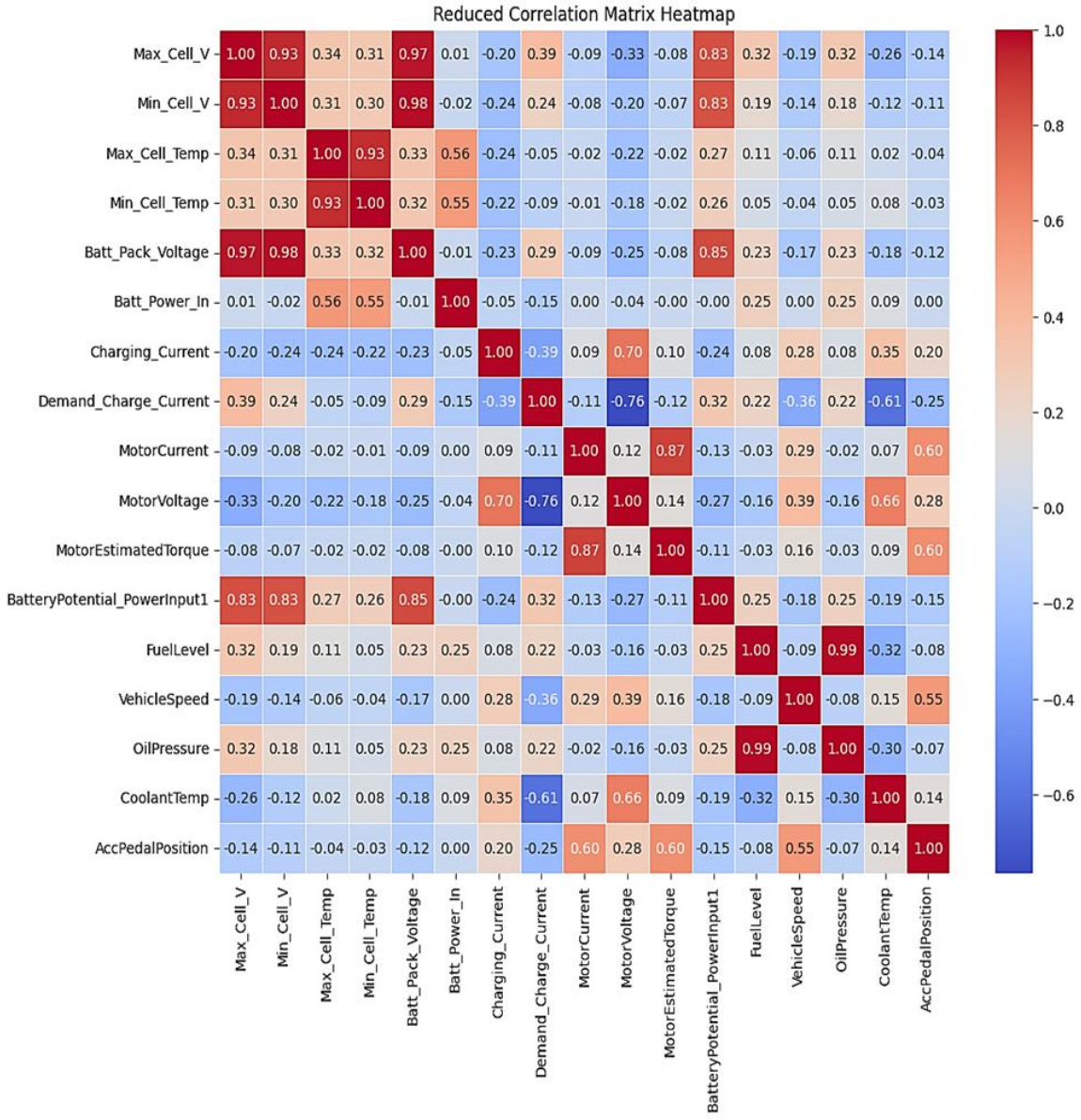



Figure 17 Correlation Analysis Matrix Heatmap

Figure 17 shows the correlation analysis of all parameters significant for three electric vehicle use cases in a heatmap. A positive correlation is observed between cell voltages and battery potential power input, cell temperatures and battery power input, accelerator pedal position and motor current, coolant temperature and motor voltage. This implies the significance of these parameters in individual use cases. Similarly, a negative correlation is observed between motor voltage and demand charge current, coolant temperature and demand charge current which indicates a negative effect on parameters.

The individual heatmap correlation helps to focus only on certain influential parameters and their effect on the use case. This activity helps to reduce the computation time while the machine learning model is training. The negative effect of leaving out other parameters is assumed to be insignificant.

An individual use-case-based correlation matrix was prepared. The heatmap of cell imbalance parameters is shown in Figure 18 which indicates that the individual cell voltages, cell temperatures, and fuel level (State of Charge – SoC) are the significant parameters. Cell temperature and battery potential has higher influence on the cell voltages as per the values in the heat map matrix.

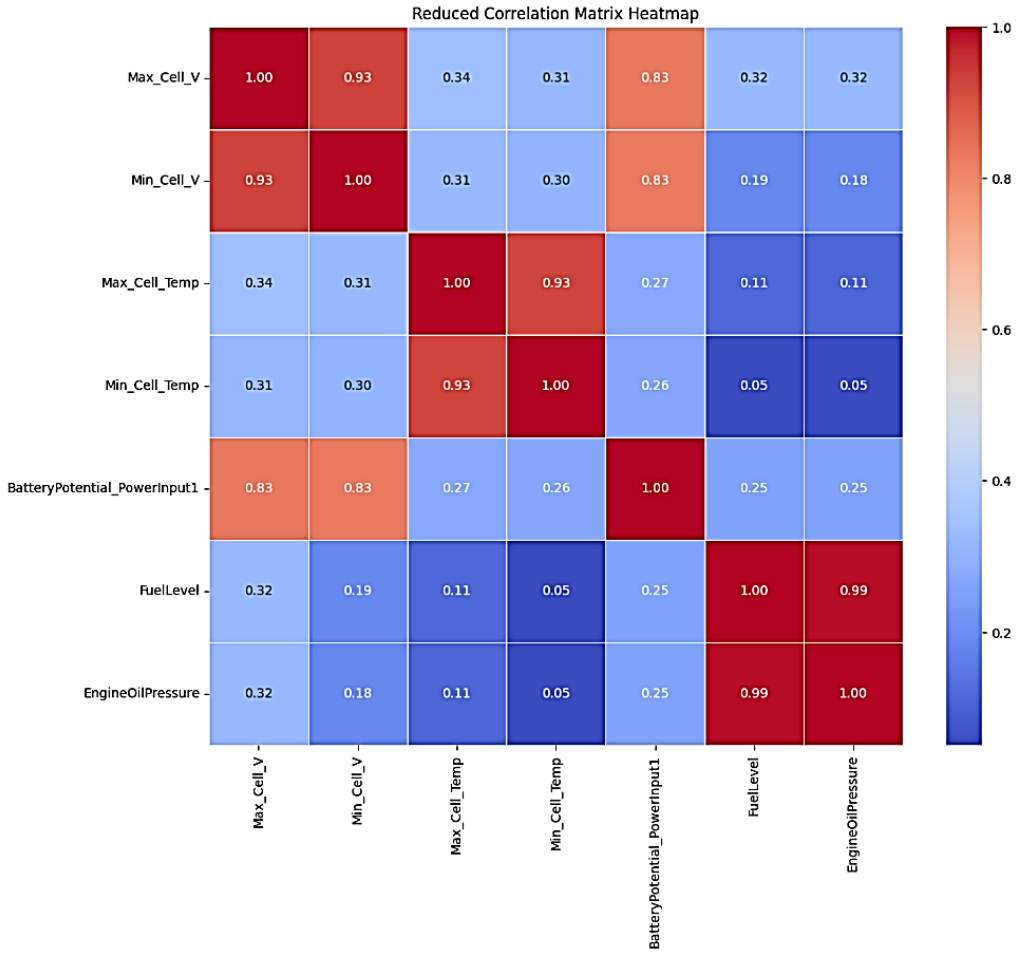



Figure 18 Correlation Analysis Matrix Heatmap for Cell Imbalance

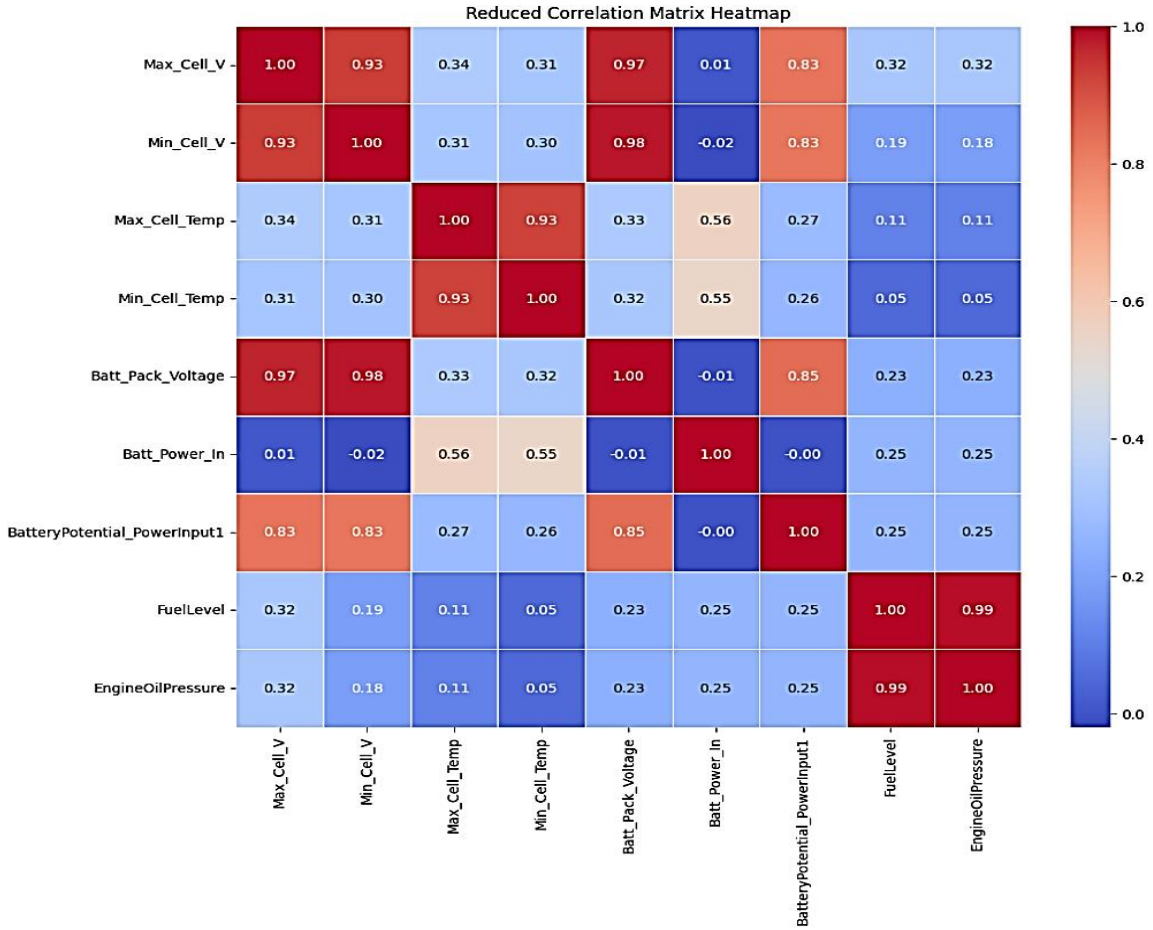



Figure 19 Correlation Analysis Matrix Heatmap for Cell Temperature

Significant parameters for the cell temperature use case are battery power, cell temperatures, and cell voltages as shown in Figure 19. Battery input power and voltage have higher positive correlation values whereas SOC and coolant pressure have lower values. This indicates that SOC and coolant pressure has very minimal effects on cell temperature

Influential parameters for the PMSM Traction Motor temperature use case is charging current, motor voltage, vehicle speed, coolant temperature as shown in Figure 20. Motor voltage and charging current are having higher influence on the motor temperature as per the positive correlation values in the heat map matrix whereas vehicle speed has the least influence.

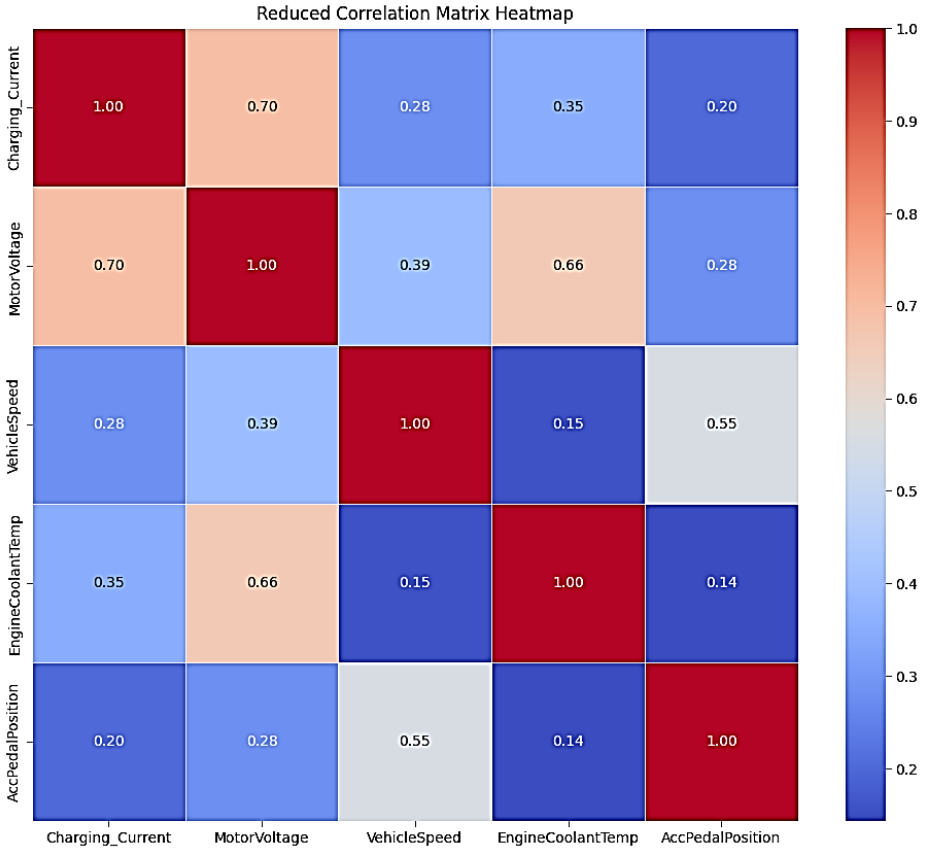



Figure 20 Correlation Analysis Matrix Heatmap for Motor Temperature

## 4.7 Performance Metrics

### 4.7.1 Mean Absolute Error (MAE)

MAE quantifies the mean absolute deviation between the real-time values and predicted values. It presents a forthright evaluation of prediction precision, with lower MAE values indicating higher accuracy in forecasting. It is calculated for each data point and is appropriate for calculating the overall accuracy of model prediction, providing insights into the magnitude of errors. MAE helps assess the capability of models to make predictions more accurate across the entire dataset, regardless of the direction of errors.

### 4.7.2 Root Mean Squared Error (RMSE)

RMSE quantifies the root of mean squared difference between forecasted and real-time values. It castigates larger errors more than MAE; this makes it responsive to deviations. RMSE is suitable for evaluating the magnitude of errors and assessing the overall performance of predictive models. RMSE gives perception into the variability of errors and supports identifying the presence of outliers or extreme predictions.

### 4.7.3 R-squared ($R^2$)

$R^2$ quantifies the proportion of variance in the dependent target (variables) that is analyzed by the independent features (variables). It illustrates the integrity of the fit in the model with data. $R^2$ is used to evaluate the predictive power of the model and assess its capability to apprehend the variability in the target variable. It ranges from 0 to 1, with value close to one citing a better fit. It helps quantify the proportion of variability in the target variable that can be attributed to the model's predictors.

### 4.7.4 Confusion Matrix

A confusion matrix is a table that compiles the performance of a classifier model by correlating predicted and actual values. It is beneficial for evaluating models with discrete outcomes. True positive (TP), true negative (TN), false positive (FP), and false negative (FN) are the components of the confusion matrix. Precision, accuracy, recall (sensitivity), and F1 score are the metrics of the model. The formula to calculate these metrics are shown in equations (i), (ii), (iii) and (iv) respectively. The confusion matrix gives insights into the model's capability to correctly distinguish instances into different classes and identify errors (false positives and false negatives).

Table 1 Confusion Matrix

| | Predicted Positive | Predicted Negative |
|---|---|---|
| Actual Positive | **True Positive (TP)** | **False Negative (FN)** |
| Actual Negative | **False Positive (FP)** | **True Negative (TN)** |

$$Precision = \frac{TP}{TP+FP} \quad \text{(i)}$$

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad \text{(ii)}$$

$$Recall = \frac{TP}{TP+FN} \quad \text{(iii)}$$

$$F1 - Score = \frac{2 \times Precision \times Recall}{Precision+Recall} \quad \text{(iv)}$$

By employing these evaluation techniques, the performance and reliability of the predictive analytics models can be assessed. These metrics and analyses give critical understanding of the strengths and limitations of the models, guiding further refinement and optimization to enhance predictive accuracy and effectiveness.

# Chapter 5

# Logic Based Models

Logic-based models, also known as rule-based systems or expert systems, operate on a set of predefined rules or logical statements to make decisions or predictions. Rules, conditions, actions, and inference engines are the components of a logic model. Unlike machine learning models that rely on data-driven approaches, logic-based models utilize predefined rules and logical reasoning to make predictions and draw conclusions from the data. These models are often used in scenarios where the underlying mechanisms are well understood or where interpretability and explainability are crucial.

The model evaluates input data against a set of rules or conditions and generates output based on the logical implications of those rules. Logic-based models are suitable for scenarios where the decision-making process can be formalized into explicit rules or where domain knowledge and expertise play a significant role in decision-making.

## 5.1 Types of Logic-Based Models

### 5.1.1 Decision Trees

Decision trees are hierarchical structures composed of nodes that represent decision points and branches that represent possible outcomes based on different conditions. Each node corresponds to a feature or attribute, and each branch represents a decision or rule based on the value of that feature. Decision trees are easy to interpret and visualize, making them valuable for understanding the decision-making process.

### 5.1.2 Expert Systems

Expert systems integrate knowledge from human experts into a computerized system to make decisions or provide advice in a specific domain. They mimic the problem-solving behavior of human experts by encoding their knowledge into a set of rules or logical statements. Expert systems are valuable for tasks that require expertise or domain-specific knowledge, such as fault diagnosis and troubleshooting.

### 5.1.3 Rule-Based Systems

Rule-based systems consist of a set of rules that encode domain-specific knowledge or expertise. Each rule comprises conditions and corresponding actions, specifying the actions to be taken when certain conditions are met. Rule-based systems can be transparent and interpretable, allowing users to understand the reasoning behind the model's decisions.

Figure 21 shows a flow chart of the process followed for logic-based model. The recorded parameters from the vehicles shared to the cloud from the TCU are available in the Volvo Eicher Uptime Centre's server. The parameters required for analyzing the different use cases of electric vehicles over a period are selected and downloaded. Eicher's electric buses typically run between six hours to ten hours a day. The data recorded will be for every minute once during its operation. If the bus operates for three hours, then 180 entries of values of all parameters will be recorded.

The recorded data is shared as a Comma-Separated Values (CSV) file and it consists of many parameters including vehicle ID details, location, minimum & maximum cell voltage, minimum & maximum cell temperature, battery pack voltage, battery power input, charging current, demand charge current, charging time, motor current, motor voltage, motor estimated torque, charging status, battery potential power input, regeneration power, auxiliary power consumption, and date &time details.

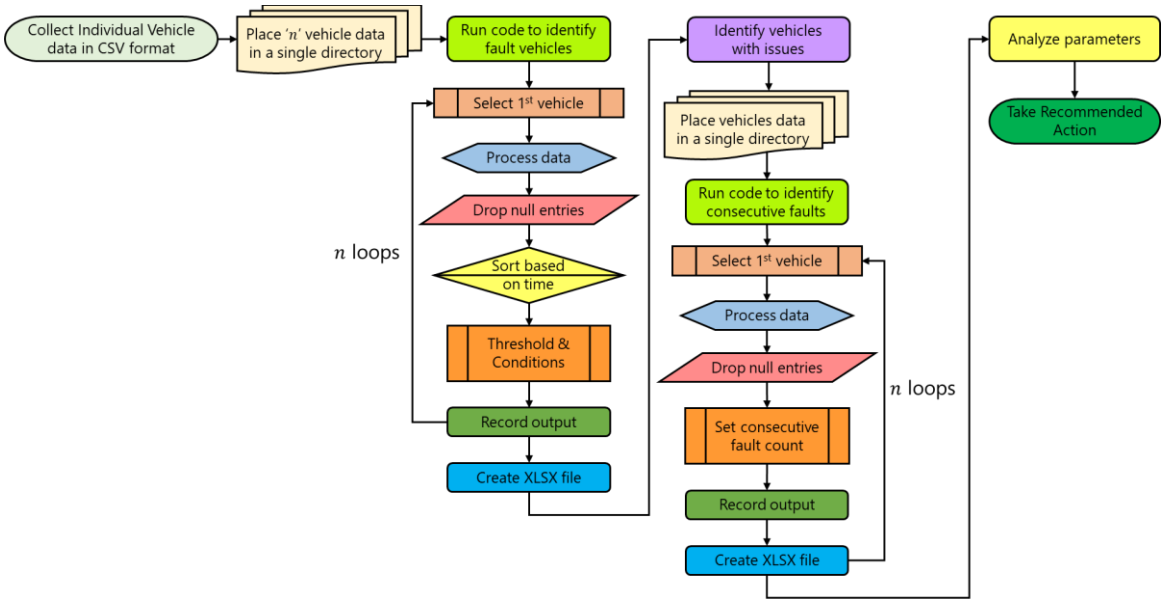



Figure 21 Flow chart for Logic Based Model

Based on the number of electric buses that needs to be analyzed, all CSV files will be placed in a common directory. First python code will run to preprocess the data, drop any null entries, sort the data sequentially, apply the use case thresholds and conditions, record the output, and create an excel (XLSX) file to store the results. This excel file distinguishes the electric buses with and without issues.

The electric bus datasets with issues will be placed in another directory for analyzing the consecutive faults if any present in the vehicle. Second python selects the datasets one by one, preprocess the data, drop any null entries, set consecutive fault count limit, record the output, and create an excel (XLSX) file to store the results. The results are analyzed and recommended actions are to be considered.

The flow of fifty-three octillion battery-operated electric bus with two months of data analyzed using two python codes for filtering out the faulty vehicles and finding the consecutive faults present in it are shown in Figure 22.

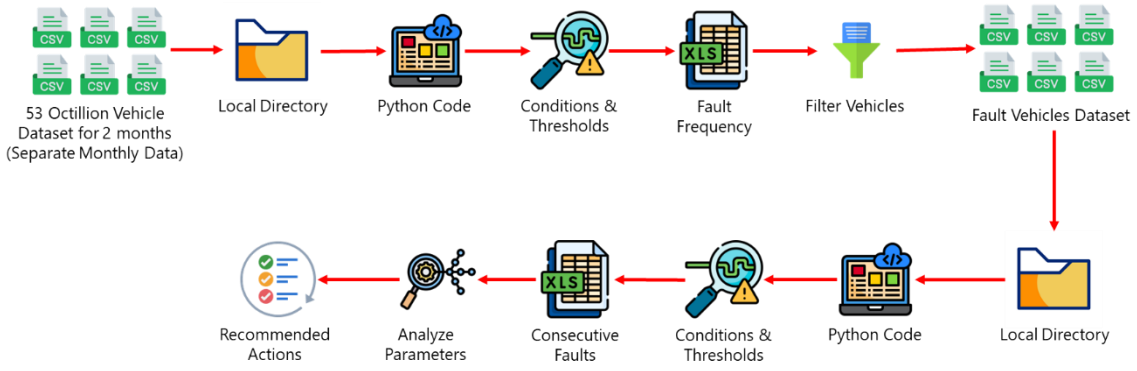



Figure 22 Pictorial representation of Logic Based Model analysis methodology

## 5.3 Application of Logic-Based Models

Logic-based models can be used for diagnosing faults and identifying potential issues in electric and IC engine vehicles based on predefined rules and diagnostic criteria. These models can also help to identify the underlying causes of problems or failures by analyzing the relationships between different variables and components in the vehicle system. These models can provide recommendations for maintenance and repair actions based on diagnostic results and historical data, helping optimize maintenance schedules and resource allocation.

## 5.4 Advantages and Limitations

### Advantages

- ✓ Interpretable and transparent decision-making process.
- ✓ Utilizes domain knowledge and expertise effectively.
- ✓ Can handle complex decision logic and uncertainty.

### Limitations

- ✓ Reliance on explicit rules may lead to oversimplification or omission of important factors.
- ✓ Difficulty in capturing implicit knowledge that is not explicitly represented in rules.
- ✓ Limited ability to adapt to changing conditions or unforeseen scenarios without manual intervention.

By exploring logic-based models in the context of predictive analytics, its strengths in interpretability and domain knowledge integration to complement the predictive capabilities of machine learning models can be leveraged. These models offer valuable insights and decision support for various tasks ranging from fault diagnosis to maintenance planning in automotive systems.

# Chapter 6

# Results and Discussion

The machine learning model experiments conducted for this research project focused on battery cell imbalance use case and the final acceptable results are extended to the remaining use cases as the type of dataset under study is similar. The logic-based models are developed for all the use cases and the results are compared with ML models. This chapter will discuss the results of two machine learning models LGBM and LSTM for cell imbalance use case, the comparison between the performances of machine learning model and logic-based model.

## 6.1 Machine Learning Model Results

### 6.1.1 Light Gradient Boosting Machine (LGBM) Model

LGBM model is developed in python to make predictions of test dataset based on the learnings from trained dataset. The code uses pandas and lightbgm libraries along with sci-kit learn libraries for metrics and model selection. A vehicle with 6 months of data in form of CSV file is considered with the help of pandas library, sorted by the IST time of data recording, preprocessing the dataset by dropping unnecessary columns and handling missing values.

The model splits the data into features and targets with target being cell imbalance column while remaining influential columns being features. This cell imbalance column is populated based on the conditions and threshold defined by the battery supplier. Cell imbalance value is one (1) if any of the following conditions hold true else it is zero (0):

- ✓ Maximum cell voltage crossed 3.65V
- ✓ Minimum cell voltage went below 2.9V
- ✓ The voltage difference between maximum and minimum cell is more than 0.5V.

The model then sets the LGBM parameters with binary objective and binary log loss metric. The code ensures that column wise parameters are evaluated while training the model. The maximum number of boosting rounds for the training process is set to hundred (100) and the maximum number of rounds to wait for if there's no improvement in the evaluation metric on the validation set before stopping the training is set to ten (10).

The code iterates over a hundred training sessions *lgb.train()* function is called to train the model for one boosting round and results from the previous model *(bst)* from the last boosting round is used for incremental learning.

After each boosting round, the code checks if the current round minus the best iteration so far is greater than or equal to the stopping criteria and if it is, it breaks out of the loop, stopping the training early. After training, the trained model *(bst)* is used to make predictions on the test dataset and *bst.predict()* function returns the predicted probabilities for each sample in the test dataset. Finally, it converts the predicted probabilities into binary predictions by thresholding at 0.5. Values greater than 0.5 are considered as Class 1, and values less than or equal to 0.5 are considered as Class 0.

The model was run across different input datasets to visualize the variation in performance. The trials conducted yielded different sets of results as shown in Figure 23. The accuracy of the model is high, but the precision, recall, and F1-Score results dropped when the input dataset was changed. This variation observed is due to the very severe class imbalance of the data in the ratio of 1:400, whereas the nominal ratio is of 1:10.

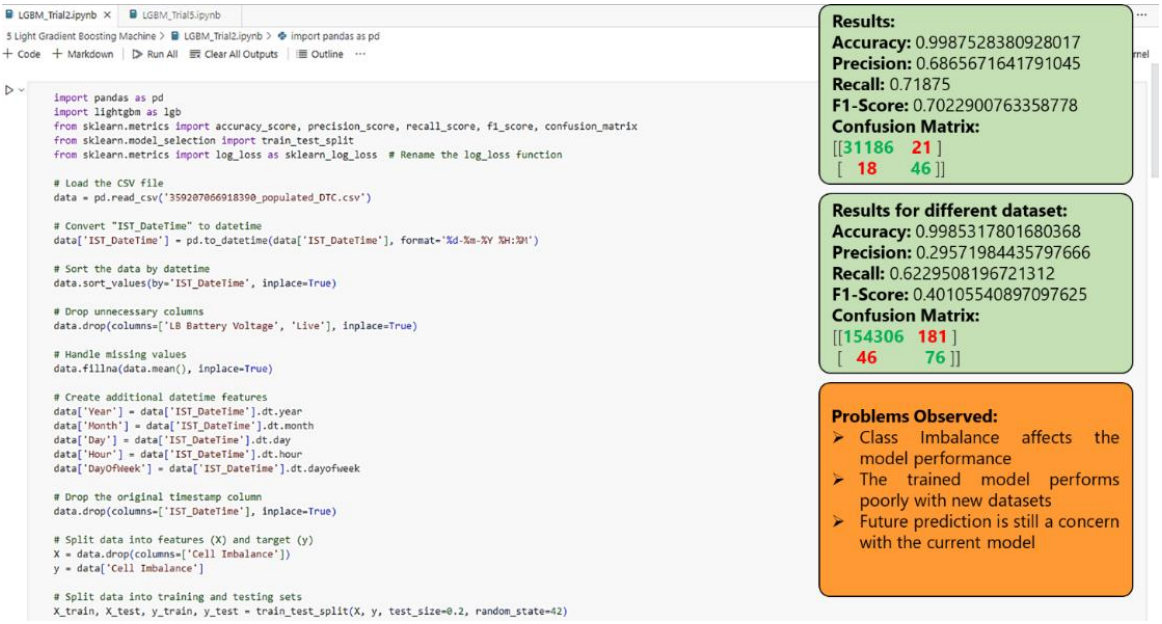



Figure 23 LGBM model and its results across datasets

### 6.1.2 Long-Short Term Memory (LSTM) Model

An LSTM model is developed in Python to make predictions on a test dataset based on learnings from a trained dataset. The code utilizes Pandas and Keras libraries along with scikit-learn libraries for metrics and model selection. A vehicle with 6 months of data in the form of a CSV file is considered with the help of the pandas library, sorted by the IST time of data recording, preprocessing the dataset by dropping unnecessary columns and handling missing values.

Like LGBM model, LSTM also splits the data into features and targets in the same manner. Cell imbalance column is populated like LGBM model. The data is split into training and testing sets, with 80% used for training and 20% for testing. The *MinMaxScaler* is initialized for normalization, and the features are normalized. The data is then converted into sliding windows of a specified size for input into the LSTM model.

The model calculates class weights to account for data imbalance and builds and compiles the LSTM model with binary cross-entropy loss and accuracy metrics. The model trains with the class weights for ten (10) epochs and a batch size of thirty-two (32). After training, the trained model is used to make predictions on the test dataset. The *model.predict()* function returns the predicted probabilities for each sample in the test dataset. These probabilities are converted into binary predictions using a threshold of 0.7 to classify the classes as zero (0) and one (1).

The model's performance is evaluated using accuracy, precision, recall, F1-score, and a confusion matrix. The accuracy of the model is high, but the precision, recall, and F1-score results vary depending on the input dataset as shown in Figure 24. This variation is observed due to the severe class imbalance, and the DTC errors occur as an anomaly leading to poor prediction.


Figure 24 LSTM model and its results

### 6.1.3 Multi Variate Multi Step Ahead LSTM Model

An LSTM model is developed in Python to make multi-step ahead predictions on a test dataset based on learnings from a trained dataset. The code utilizes pandas and TensorFlow libraries along with scikit-learn libraries for metrics and model selection.

Initially, the input data is strategically reduced to address class imbalance. This is achieved by selecting key indices where "Cell Imbalance" is present and ensuring a balanced representation of the data (1:25). Specifically, the reduction process involves identifying instances where the "Cell Imbalance" column equals one (1) and then selecting surrounding data points to provide context. This method ensures that both balanced and imbalanced data points are adequately represented, improving the model's performance.

The model splits the data into features and targets in the same manner. Cell imbalance column is populated like LGBM model. The input data is normalized using the *MinMaxScaler*, and the features are transformed into sequences of data for input into the LSTM model. The model defines the number of time steps and features to create sequences of data for training. The data is split into training and testing sets, with 80% used for training and 20% for testing.

The model builds and compiles the LSTM network with an Adagrad optimizer and Mean Squared Error (MSE) loss function. The LSTM model consists of 50 units, followed by a dropout layer to prevent overfitting, and a dense layer for output. The model trains for one hundred (100) epochs with a batch size of ninety-six (96), and the training process includes validation to monitor performance on the test set. After training, the trained model is used to make predictions on the test dataset. The *model.predict()* function returns the predicted values for each sample in the test dataset.

The model's performance is evaluated using RMSE and MAE for each time step from 1 to 10. The evaluation results demonstrate model's capability to predict multi-step ahead values effectively. RMSE and MAE scores for the tenth time step indicate that the model can predict the maximum cell voltage with a high degree of accuracy. The training and validation loss over epochs are plotted to visualize the model's learning process.

Overall, the LSTM model's ability to predict values better is attributed to its multivariate approach, considering multiple influential features, and its multi-step ahead forecasting capability. The reduction in data imbalance before training further enhances the model's performance, ensuring a more balanced and representative dataset for training.

The results in Figure 25 show that the model has a stable training process with minimal overfitting, as indicated by the close alignment of training and validation loss curves. The model was run across different input datasets to visualize the variation in performance. The trials conducted yielded consistent results, demonstrating the model's robustness in handling multivariate time series data. The high accuracy of the model, coupled with low RMSE and MAE scores, highlights its effectiveness in making reliable multi-step predictions.

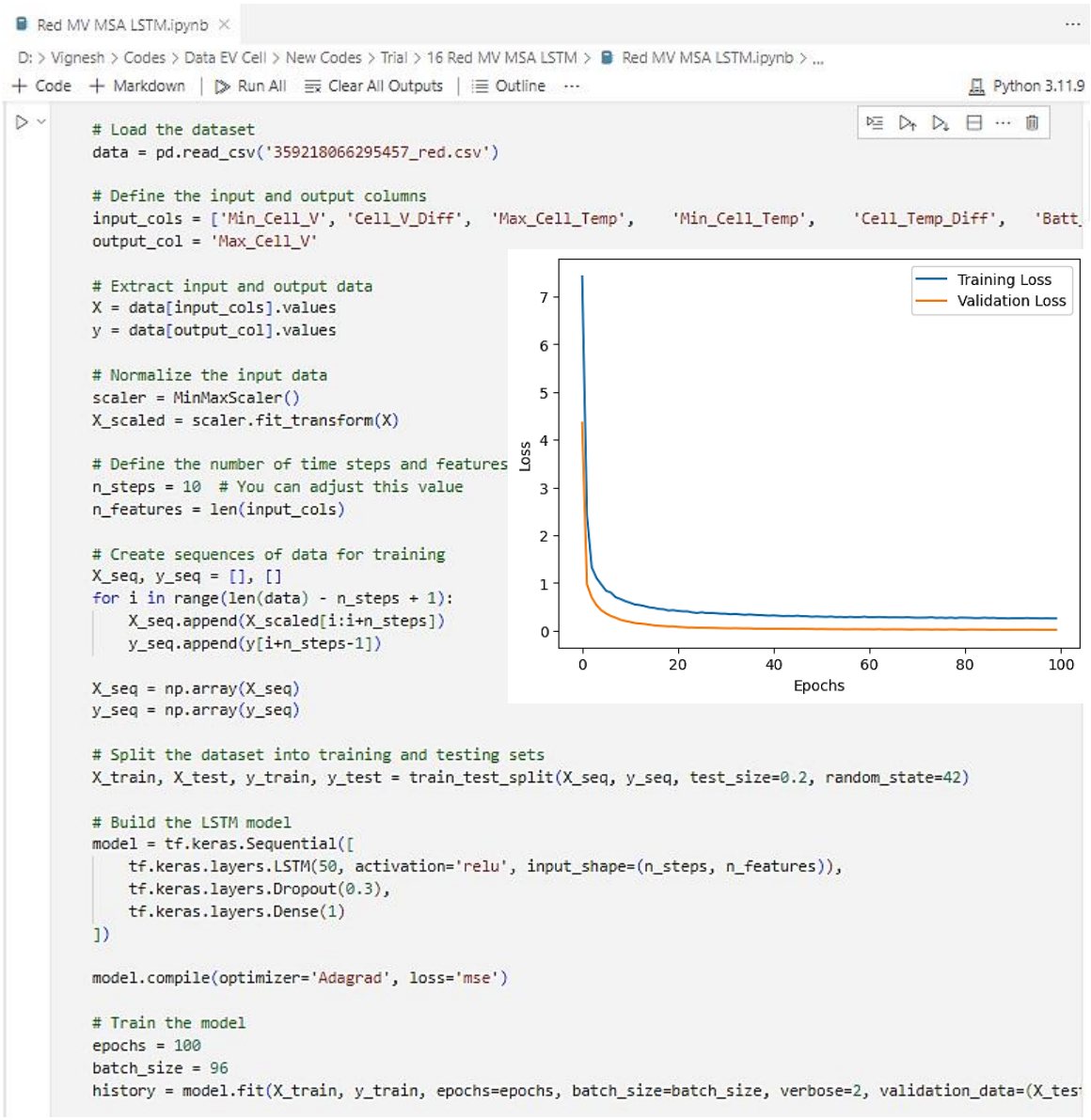



Figure 25 Training and Validation loss of multivariate LSTM model across epochs

To improve the model performance, Design of Experiments (DoE) was conducted on window size, LSTM units, and batch size for the input data. Window size varied between 40, and 50; LSTM units between 50, 100, and 150; batch size between 32, 64, and 128 in the experiments. The results of the DoE show cases that errors, losses are low and validation accuracy is high for combinations of higher unit size with higher batch size. Table 2 shows the set of results and its corresponding input parameters along with individual computation time.

Table 2 Hyper Parameter Tuning through Design of Experiments

| Experiment | Window size | LSTM Units | Epochs | Batch Size | RMSE | MAE | Training Loss | Validation Loss | Validation Accuracy | Run Time (mins) | Comments |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 50 | 100 | 200 | 64 | 0.056 | 0.042 | 0.106 | 0.003 | 0.944 | 12 | Baseline |
| 2 | 50 | 100 | 200 | 32 | 3.442 | 3.441 | 11.856 | 11.848 | -2.442 | 17 | Smaller Batch |
| 3 | 50 | 100 | 200 | 128 | 0.085 | 0.062 | 0.126 | 0.007 | 0.915 | 12 | Bigger Batch |
| 4 | 50 | 150 | 200 | 64 | 0.090 | 0.067 | 0.098 | 0.008 | 0.910 | 18 | Increased Layers |
| 5 | 50 | 50 | 200 | 64 | 136.827 | 114.851 | 23336.156 | 18721.645 | -135.827 | 9 | Reduced Layers |
| 6 | 40 | 100 | 200 | 64 | 0.062 | 0.046 | 0.082 | 0.004 | 0.938 | 11 | Reduced Window Size |

Based on the hyper parameter tuning, the multi variate multi step ahead model is run again to predict the values and it showcases minimal deviation from the actual value. The trend of the cell voltage over time and sample index is shown in Figure 26 and minor variations are predicted with a close range whereas abnormalities are forecasted with slightly higher deviations. These show the improvement in the model performance after the hyper parameter tuning.

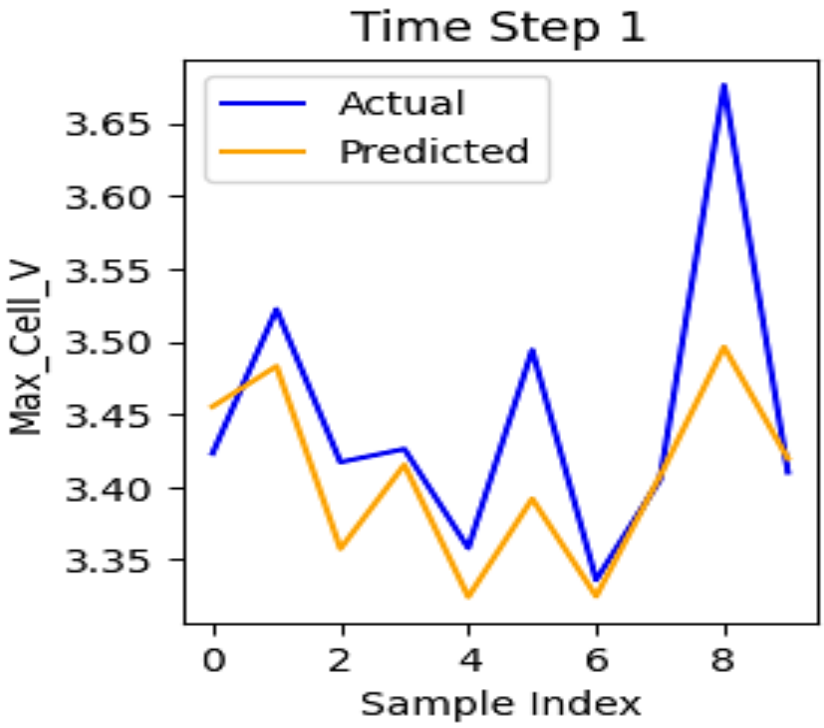



Figure 26 Actual and predicted values of the cell voltage based on hyper tuned model

## 6.2 Logic Based Model Results

### 6.2.1 Battery Cell Imbalance Use Case

A logic-based model is developed in Python to analyze a dataset and calculate various metrics based on predefined thresholds. The code utilizes pandas for data manipulation and os for file operations. This model processes multiple CSV files to extract and compute metrics related to cell voltage in a vehicle's battery pack. Initially, the model loads each CSV file from the current directory, filtering the data to remove entries where both maximum and minimum cell voltages are zero. It then sorts the data by time to ensure chronological order.

The model defines thresholds for maximum and minimum cell voltages and computes additional metrics such as the difference between the maximum and minimum cell voltages *(Cell_V_Diff)*. If this difference exceeds a specified threshold, it counts the occurrence. The metrics calculated include Total number of entries, Maximum cell voltage across all entries, Minimum cell voltage across all entries, Frequency of entries where *Cell_V_Diff* exceeds the threshold.

For each CSV file, the model calculates the following metrics and stores them in a summary Data Frame. Those metrics are Frequency of *'Max_Cell_V'* crossing the threshold (3.65V), Frequency of *'Min_Cell_V'* crossing the threshold (2.9V), Frequency of *'Cell_V_Diff'* crossing the threshold (0.5V), Total Entries in the Vehicle, *Max_Cell_V* out of All Entries, and *Min_Cell_V* out of All Entries.

After processing all files, the model consolidates the results into a single Excel file with a summary sheet. Each column in the summary sheet is auto-sized based on the content for better readability. The logic-based approach ensures that all significant metrics related to battery cell voltage are computed efficiently, providing insights into the frequency and extent of voltage variations. By aggregating the data across multiple files, the model offers a comprehensive view of the vehicle's battery performance over time.

Overall, this model is effective in identifying critical events related to cell voltage thresholds and providing a detailed summary of battery health indicators across multiple datasets. The strategic calculation of additional metrics enhances the understanding of voltage behavior, contributing to better battery management and maintenance.

The model's output demonstrates the frequency and extent of cell voltage variations, highlighting key metrics such as the frequency of threshold crossings and overall voltage extremes. This information is crucial for monitoring battery performance and identifying potential issues related to cell imbalance and voltage fluctuations.

### 6.2.2 Battery Cell Temperature Use Case

A model similar to cell imbalance use case is developed for cell temperature, the change being the conditions and threshold. This model defines thresholds for maximum and minimum cell temperatures and calculates additional metrics. These include Total number of entries, Maximum cell temperature across all entries, Minimum cell temperature across all entries.

For each CSV file, the model calculates the following metrics and stores them in a summary Data Frame. Those metrics are Frequency of *'Max_Cell_Temp'* crossing the threshold (55°C), Frequency of *'Min_Cell_Temp'* crossing the threshold (15°C), Total Entries in the Vehicle, *Max_Cell_Temp* out of All Entries, *Min_Cell_Temp* out of All Entries. The model calculates these metrics using a helper function that processes the filtered data. For each file, it creates a result dictionary that is then converted to a Data Frame. All individual results are concatenated into a single Data Frame that aggregates the results across all files. Finally, the combined results are written to an Excel file with a summary sheet. Each column in the summary sheet is auto-sized based on the content for better readability.

This logic-based approach ensures comprehensive analysis of cell temperature metrics, providing valuable insights into temperature behavior within the battery pack. By processing multiple files, the model offers a detailed overview of the temperature variations, which is crucial for monitoring battery health and performance. Overall, this model effectively identifies critical temperature events, enhancing the understanding of cell temperature dynamics. The consistent calculation of additional metrics across multiple datasets provides a robust framework for temperature monitoring and analysis.

The model's output demonstrates the frequency and extent of temperature variations, highlighting key metrics such as the frequency of threshold crossings and overall temperature extremes. This information is vital for assessing battery performance and identifying potential issues related to cell temperature fluctuations.

### 6.2.3 Motor Temperature Use Case

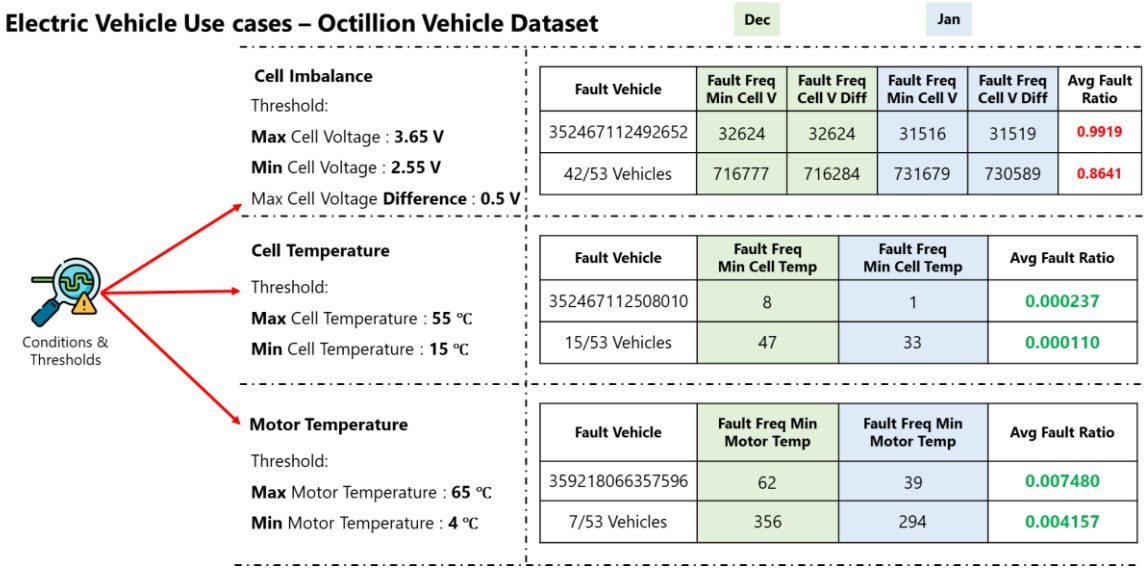
A model similar to cell temperature use case is developed for motor temperature, the change being the conditions and threshold. This model defines thresholds for maximum and minimum motor temperatures and calculates additional metrics. These include Total number of entries, Maximum motor temperature across all entries, Minimum motor temperature across all entries.

For each CSV file, the model calculates the following metrics and stores them in a summary Data Frame. Those metrics are Frequency of *'Max_Motor_Temp'* crossing the threshold (65°C), Frequency of *'Min_Motor_Temp'* crossing the threshold (4°C), Total Entries in the Vehicle, *Max_Motor_Temp* out of All Entries, *Min_Motor_Temp* out of All Entries. The model calculates these metrics using a helper function that processes the filtered data. For each file, it creates a result dictionary that is then converted to a Data Frame. All individual results are concatenated into a single Data Frame that aggregates the results across all files. Finally, the combined results are written to an Excel file with a summary sheet. Each column in the summary sheet is auto-sized based on the content for better readability.

This logic-based approach ensures comprehensive analysis of engine coolant temperature metrics, providing valuable insights into temperature behavior within the vehicle's system. By processing multiple files, the model offers a detailed overview of the temperature variations, which is crucial for monitoring engine performance and identifying potential issues.

Overall, this model effectively identifies critical temperature events, enhancing the understanding of engine coolant temperature dynamics. The consistent calculation of additional metrics across multiple datasets provides a robust framework for temperature monitoring and analysis. The model's output demonstrates the frequency and extent of temperature variations, highlighting key metrics such as the frequency of threshold crossings and overall temperature extremes. This information is vital for assessing engine performance and identifying potential issues related to engine coolant temperature fluctuations.

Figure 27 shows the consolidated selective results obtained from the vehicles across three different use cases of electric vehicles. The results show the issues pertaining to battery cell imbalance are very high compared to remaining use cases. This is due to the possibility of high impedance during battery cell assembly that results in increased resistance on the cell which leads to imbalance between cells.




| Fault Vehicle | Fault Freq Min Cell V | Fault Freq Cell V Diff | Fault Freq Min Cell V | Fault Freq Cell V Diff | Avg Fault Ratio |
|---|---|---|---|---|---|
| 352467112492652 | 32624 | 32624 | 31516 | 31519 | 0.9919 |
| 42/53 Vehicles | 716777 | 716284 | 731679 | 730589 | 0.8641 |

| Fault Vehicle | Fault Freq Min Cell Temp | Fault Freq Min Cell Temp | Avg Fault Ratio |
|---|---|---|---|
| 352467112508010 | 8 | 1 | 0.000237 |
| 15/53 Vehicles | 47 | 33 | 0.000110 |

| Fault Vehicle | Fault Freq Min Motor Temp | Fault Freq Min Motor Temp | Avg Fault Ratio |
|---|---|---|---|
| 359218066357596 | 62 | 39 | 0.007480 |
| 7/53 Vehicles | 356 | 294 | 0.004157 |

Figure 27 Consolidated results of most faulty vehicles across use cases for logic model

The passive balancing present in the BMS is not able to fix the imbalance signifies that there are issues with charging that individual faulty cell. This issue was highlighted to the battery supplier through the manufacturer, the root cause analysis also suggests that the issue is with an individual cell present in a module with high impedance. To rectify the issue impedance matching needs to be performed or the faulty cell must be replaced.

Remaining use cases did not pose severe threats like cell imbalance use case, as only very minor issues are reported occasionally. The model is also capable of triggering in case of major continuous faults.

### 6.2.4 Engine Oil Pressure Use Case

A logic-based model is developed in Python to analyze a dataset for low oil pressure conditions based on predefined thresholds. The model comprises two main scripts: one for splitting the dataset into smaller, more manageable files and another for analyzing these files to identify critical low oil pressure events.

This model leverages pandas for data manipulation, os for file operations, and Json for configuration management. The first script is responsible for splitting a large CSV file into multiple smaller files based on the 'Chassis' column. The script begins by loading configuration settings from a JSON file, which specify the input directory, output directory, and the name of the input CSV file. It checks if the output directory exists and creates it if necessary, ensuring there is a dedicated space for the separated files.

The script reads the input CSV file into a pandas Data Frame, identifies unique chassis values, and filters the Data Frame to include only rows associated with each chassis. For each unique chassis value, the script creates a new CSV file containing data specific to that chassis and saves it in the output directory. This organized separation of data facilitates targeted analysis and improves the manageability of large datasets.

The second script analyzes the separated CSV files to identify instances where the engine oil pressure falls below specified thresholds under certain conditions. The script loads analysis parameters from a JSON file, which include the required columns, engine speed intervals, oil pressure thresholds, and output settings. Similar to the splitting script, this script ensures the existence of an output directory for storing the analysis results.

The script iterates through each CSV file generated by the splitting process. It reads each file, filters out rows where critical columns have zero values, and processes the data based on the specified conditions. For each engine speed interval defined in the configuration, the script identifies rows where the engine oil pressure is below the threshold, the vehicle speed is above a certain value, and the accelerator pedal position exceeds a defined threshold. These rows are compiled into a Data Frame. The engine speed intervals and corresponding low oil pressure thresholds are as follows:

Table 3 Engine RPM range and its corresponding oil pressure threshold

| Engine RPM Range | Oil Pressure Threshold (kPa) |
|---|---|
| 800 – 1250 | 125 |
| 1250 – 1500 | 150 |
| 1500 – 2000 | 200 |
| 2000 – 2500 | 225 |
| 2500 – 4000 | 250 |

The filtered rows meeting the conditions are compiled into a Data Frame, sorted by their row entry index, and saved to an Excel file. This file contains a sheet with all entries that meet the criteria. The script identifies sequences of consecutive rows where the low oil pressure condition persists. If the number of consecutive rows exceeds a specified threshold, these are compiled into a separate sheet within the same Excel file.

The model provides comprehensive insights into engine performance by highlighting key metrics, such as the frequency of low oil pressure occurrences and consecutive low-pressure events. This information is crucial for monitoring engine health, identifying potential issues, and ensuring timely maintenance. By structuring the analysis results in a well-organized Excel format, the model facilitates easy review and further analysis, contributing to better decision-making and engine management. Overall, this model offers a robust approach to understanding and mitigating risks associated with low oil pressure in vehicle engines, thereby enhancing reliability and performance.

Figure 28 shows the python codes for data splitting along with the JSON files. Python file uses JSON file to set the output directory and file name. Figure 29 shows the python codes for data analysis and the thresholds set in the JSON files; the code provides warning based on the consecutive count of set thresholds.

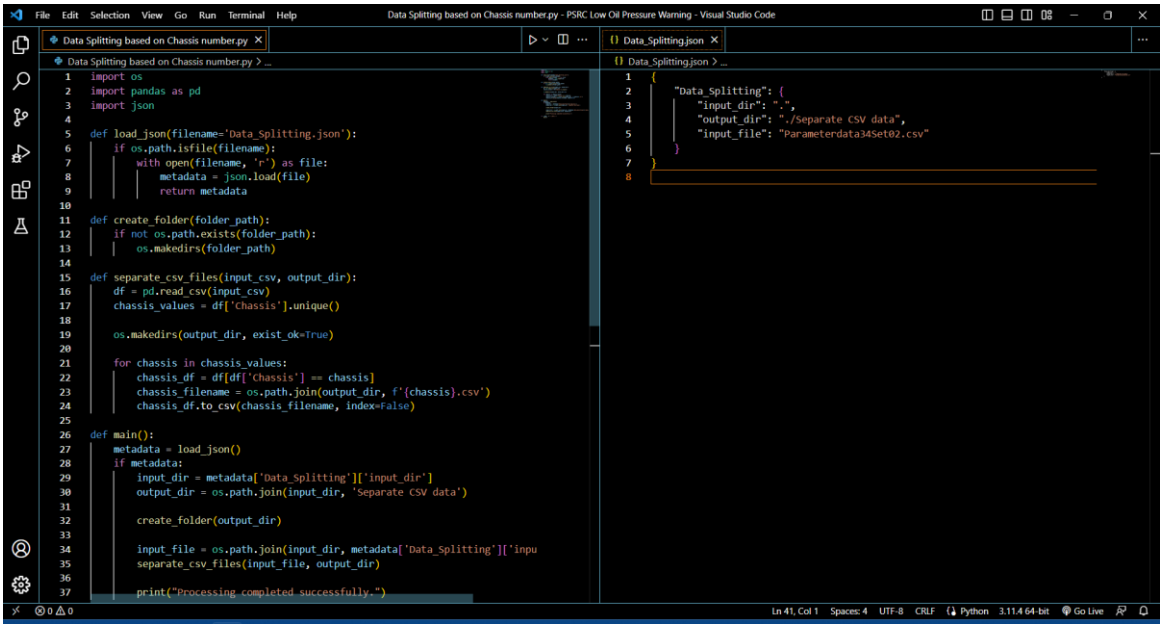



Figure 28 Production server codes for data splitting for oil pressure use case

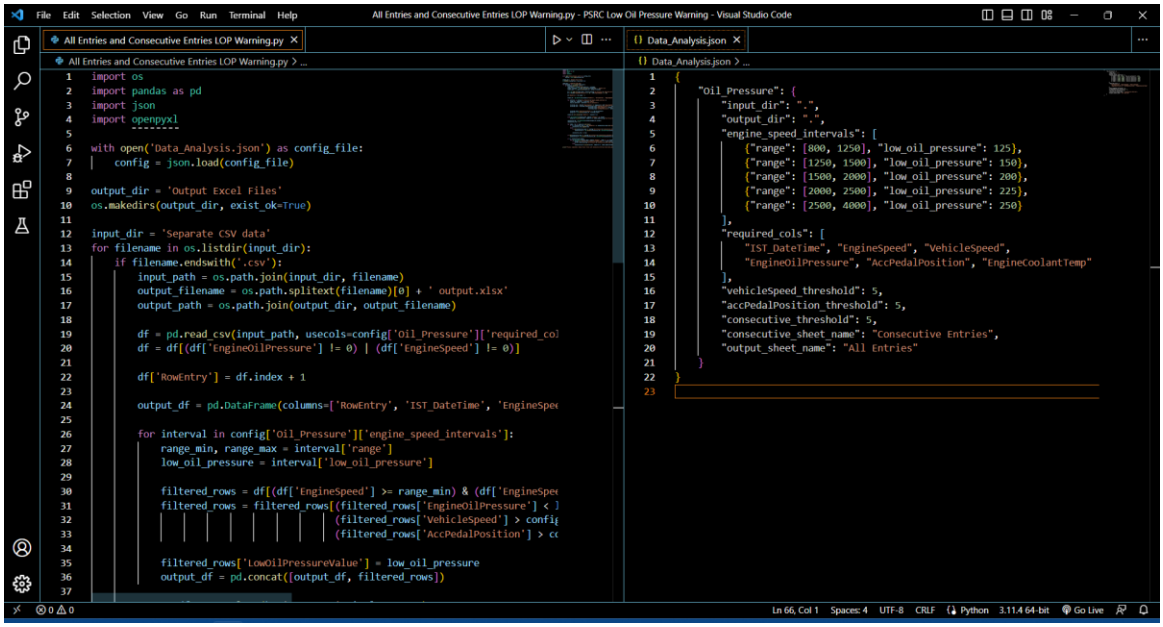



Figure 29 Production server codes for data analysis for oil pressure use case

## 6.3 Comparison of Machine Learning and Logic-Based Models

The comparison between performances and applicability of machine learning and logic-based models under various aspects are shown in Table 4. This comparison helps to analyze the advantages and limitations of both ML and Logic model-based approaches chosen for the use cases.

Table 4 Comparison between Machine learning models and Logic-based models

| Aspect | Machine Learning Model | Logic-Based Model |
|---|---|---|
| **Approach** | Uses algorithms to learn from data and make predictions. | Uses predefined rules and thresholds to analyze data. |
| **Flexibility** | Highly flexible; can adapt to new data patterns without programming. | Less flexible; requires manual updates to rules and thresholds. |
| **Complexity** | More complex; requires understanding of algorithms. | Simpler; based on straightforward conditional logic. |
| **Data Requirement** | Requires a large amount of historical data for training. | Can operate with relatively smaller datasets. |
| **Accuracy** | Higher accuracy with sufficient and high-quality training data. | Accuracy depends on appropriateness of predefined rules. |
| **Scalability** | Scalable to large datasets with advanced algorithms and hardware. | May struggle with scalability for very large datasets. |

| | | |
|---|---|---|
| **Maintenance** | Requires regular retraining with new data to maintain accuracy. | Requires manual updates to logic as new rules are identified. |
| **Transparency** | Can be a "black box"; difficult to interpret how decisions are made. | Transparent; decision-making process is clear and understandable. |
| **Deployment** | More challenging; needs specialized skills for deployment. | Easier to deploy; straightforward implementation. |
| **Use Case** | Suitable for complex, dynamic environments | Suitable for well-defined, stable environments |
| **Computational Resources** | Requires significant computational power, especially for training. | Generally, requires fewer computational resources. |
| **Initial Setup** | Time-consuming; involves data preprocessing, model selection, and training. | Quick; primarily involves defining rules and thresholds. |
| **Handling New Scenarios** | Can generalize and adapt to new, unseen scenarios. | Limited to predefined scenarios. |

### 6.3.1 Strengths and Weaknesses

ML models have the capability to learn from large amounts of dataset and identify intricate patterns that may not be immediately visible through traditional analysis. This capability is particularly advantageous for electric vehicles, where battery health and performance can be influenced by numerous interdependent factors such as temperature, charge cycles, and usage patterns. ML models can analyze these variables collectively, providing more accurate and robust predictions. Additionally, ML models are adaptable and can continuously improve their performance as more data becomes available. By retraining with updated data, ML models can maintain high levels of accuracy and relevance.

However, the complexity of ML models also presents certain weaknesses. These models often operate as "black boxes," meaning their decision-making processes are not easily interpretable. This lack of transparency can be a significant drawback, especially in safety-critical applications like vehicle performance monitoring, where understanding the reasoning behind a prediction is essential. Moreover, ML models require substantial amounts of high-quality data for training, and the process of model selection, training, and validation can be resource intensive and time-consuming. Regular maintenance and retraining are also necessary to ensure the models remain effective over time.

Logic based models are highly reliable for well-defined, stable environments where the relationships between variables are straightforward. The simplicity of logic-based models is a significant strength. They are relatively easy to implement and do not require large datasets or extensive computational resources. Their decisions are easily interpretable, which is crucial for debugging and ensuring compliance with safety standards. This transparency is particularly beneficial in automotive applications, where regulatory requirements often necessitate clear and auditable decision-making processes.

However, logic-based models have their limitations. They are less flexible and adaptable than ML models, as they rely on predefined rules that may not account for all possible scenarios or changes in the operating environment. In complex and dynamic systems like electric vehicles, where performance can be influenced by numerous interacting factors, logic-based models may struggle to capture the full extent of these interactions. They are also less effective at handling nonlinear relationships, which can limit their accuracy and predictive power in certain applications.

### 6.3.2 Cross-Validation and Model Validation

Cross-validation and model validation are critical processes in ensuring the reliability and generalizability of machine learning models. Cross validation divides the dataset into many subsets and trains it on several combinations of these subsets, while validating the remaining data. This technique helps in assessing the model's performance across various segments of the data, providing a more comprehensive evaluation than a single train-test split. Validation of the model further includes comparing the prediction of the model against actual outcomes on a separate validation dataset to ensure it performs well on unseen data. These practices are essential to prevent overfitting, where a model might perform well on training data but poorly on new data. Cross-validation and model validation are foundational to developing robust ML models that generalize well across different scenarios and datasets.

Logic-based models give analytics to real time data and will not forecast the future values, but the trend of the data can be validated based on the vehicle service history. The variations in the results can be addressed by appropriately modifying the conditions and thresholds applied for that particular use case.

### 6.3.3 Scalability and Computational Efficiency

Scalability and computational efficiency are crucial considerations when deploying machine learning models, especially in contexts that involve large volumes of data or require real-time processing, such as in automotive applications. ML models, particularly those based on deep learning, can be computationally intensive, requiring significant resources for training and inference. Ensuring that these models are scalable involves optimizing algorithms and using techniques like distributed computing or cloud-based solutions to handle large datasets effectively.

In contrast, logic-based models, with their predefined rules, are typically less demanding in terms of computational resources. They can be implemented more efficiently, making them suitable for applications where quick, real-time decisions are needed without the overhead of complex computations. Balancing scalability and computational efficiency are key to ensuring that ML models can be effectively integrated into practical, real-world systems.

### 6.3.4 Interpretability and Explainability

Interpretability and explainability are critical aspects of model deployment, particularly in fields where understanding the decision-making process is essential for trust and accountability. Logic-based models inherently offer high levels of interpretability and explainability due to their use of straightforward, human-readable rules and thresholds. These models make it easy to trace how decisions are made, which is crucial for debugging, regulatory compliance, and gaining user trust.

On the other hand, while ML models, especially complex ones like deep neural networks, can achieve high predictive accuracy, they often operate as "black boxes" with decisions that are difficult to interpret. Striking a balance between leveraging the predictive power of ML models and ensuring their decisions are interpretable and explainable is essential for their successful application in critical domains.

# Chapter 7

# Conclusions and Scope for Future Work

## 7.1 Summary

This study has explored the application of predictive analytics using both machine learning models and logic-based models for electric and internal combustion engine vehicles. By leveraging real-world data from Volvo Eicher Commercial Vehicles Limited, the study investigated four critical use cases: cell imbalance monitoring, temperature monitoring of cell and motor, and engine oil pressure warnings. The findings demonstrate that ML models excel at identifying complex patterns and nonlinear relationships within the data, leading to highly accurate predictions for vehicle metrics such as battery health and engine performance. On the other hand, logic-based models provided clear and interpretable rules for decision-making, which are crucial for understanding the underlying mechanisms of vehicle behavior. The comparison between both approaches has shown that they complement each other, with ML models enhancing predictive accuracy and logic-based models offering transparency and interpretability.

## 7.2 Project Outcome

The project aimed to develop advanced predictive analytics models to address specific use cases in electric and IC engine vehicles, which was successfully achieved. Through a detailed examination, the research provided a comprehensive understanding of various methodologies of ML models like VAR, XGBoost, LGBM, LSTM and logic-based models such as decision trees and expert systems. The project successfully met its objectives by demonstrating how these models can be applied to predict faults, optimize vehicle parameters, and enhance maintenance strategies, thus fulfilling project scope.

## 7.3 Key Contributions

The key contributions in the field of predictive analytics for automotive applications made through this work is listed below:

Enhanced Predictive Models: The study shows the potential of combining ML's predictive power with the interpretability of logic-based models, paving the way for more robust predictive systems.

Practical Applications: By applying these models to real-world data from VECV, the research demonstrates practical applications and benefits, such as improved fault prediction accuracy and reduced false alarms.

Methodological Insights: The project provides insights into cross-validation, model validation techniques, scalability, and computational efficiency, which are crucial for deploying predictive models in real-world automotive systems.

Comparative Analysis: It offers a comparative analysis of ML and logic-based models, highlighting their respective advantages and limitations.

## 7.4 Limitations and Challenges

Despite its successes, the project faced several limitations and challenges. The availability of labeled data, particularly for rare or critical events, was limited, posing a challenge for training ML models. The complexity and computational overhead of deep learning models made them less suitable for real-time applications. Additionally, capturing all relevant domain knowledge in logic-based models was challenging, which sometimes led to oversimplified decision-making rules. Ensuring model interpretability and explainability, especially for complex ML models, also presented a significant challenge.

## 7.5 Future Scope

Logic-based models provide a foundation of domain knowledge that can significantly enhance the optimization of machine learning models. By leveraging the clear and well-defined rules of logic-based models, ML practitioners gain critical insights on underlying patterns and relationships in the data. This knowledge can be used to inform feature selection, engineering, and the construction of more robust and accurate ML models. Predefined thresholds for engine oil pressure or battery temperature derived from logic-based models can be incorporated into ML models as important features or constraints, ensuring that the ML models adhere to critical safety and operational guidelines. Simplicity and transparency of logic-based models make it easier to identify and understand key variables and their interactions, which can then be explored more deeply using the advanced capabilities of ML algorithms. To further enhance the

predictive capabilities and address the identified challenges, future research should explore the following directions:

Hybrid Models: Developing hybrid models that integrate the strengths of ML algorithms with the interpretability of logic-based models, using techniques like rule extraction or model distillation.

Advanced ML Techniques: Investigating advanced ML algorithms such as reinforcement learning and self-supervised learning for more sophisticated predictive analytics, particularly in autonomous vehicles and adaptive control systems.

Data Augmentation: Collecting and annotating larger and more diverse datasets, including rare events, to improve the robustness and generalization of ML models.

Real-Time Scalability: Developing scalable and efficient algorithms for real-time monitoring and optimization of vehicle performance, considering computational constraints and resource limitations in automotive systems.

## 7.6 Conclusion

Machine learning models excel in handling complex patterns and nonlinear relationships within data, making them highly effective for predictive analytics. Logic-based models are built on predefined rules and thresholds, offering clear interpretability and transparency. This thesis has provided significant insights in the use of machine learning and logic-based models for predictive diagnostics in electric and IC engine vehicles. The following results were yielded from the developed ML and Logic models.

- ✓ A unique case based multi-step ahead multivariate LSTM model was developed
- ✓ Hyper parameter tuning of ML model through DOE improved the validation accuracy to **94.4%**
- ✓ Logic based models were developed to identify faulty vehicles and the severity of the issues
- ✓ Results from the logic model indicating issues in the existing vehicle were highlighted and root cause analysis was shared to take corrective action
- ✓ Developed logic models were deployed in VECV's uptime center for real time vehicle monitoring

The study highlights the superior predictive accuracy of ML models and the essential interpretability of logic-based models. By integrating these approaches, future research can further advance automotive diagnostics, leading to more reliable, efficient, and intelligent vehicles. This research lays a strong foundation for ongoing advancements in predictive analytics, ultimately contributing to the development of smarter and more sustainable transportation systems.

.

# APPENDIX-A

**Machine Learning Codes:**

**Light Gradient Boosting Machine Code:**

```
import pandas as pd
import lightgbm as lgb
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import log_loss as sklearn_log_loss  # Rename the
log_loss function

# Load the CSV file
data = pd.read_csv('359207066918390_populated_DTC.csv')

# Convert "IST_DateTime" to datetime
data['IST_DateTime'] = pd.to_datetime(data['IST_DateTime'], format='%d-%m-
%Y %H:%M')

# Sort the data by datetime
data.sort_values(by='IST_DateTime', inplace=True)

# Drop unnecessary columns
data.drop(columns=['LB Battery Voltage', 'Live'], inplace=True)

# Handle missing values
data.fillna(data.mean(), inplace=True)

# Create additional datetime features
data['Year'] = data['IST_DateTime'].dt.year
data['Month'] = data['IST_DateTime'].dt.month
data['Day'] = data['IST_DateTime'].dt.day
data['Hour'] = data['IST_DateTime'].dt.hour
data['DayOfWeek'] = data['IST_DateTime'].dt.dayofweek

# Drop the original timestamp column
data.drop(columns=['IST_DateTime'], inplace=True)

# Split data into features (X) and target (y)
X = data.drop(columns=['Cell Imbalance'])
y = data['Cell Imbalance']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
# Display the shapes of the train and test sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)

# Set the LGBM parameters
params = {
    'objective': 'binary',
    'metric': 'binary_logloss'
}

# Create LGBM datasets with force_col_wise parameter
lgb_train = lgb.Dataset(X_train, y_train, free_raw_data=False,
params={'force_col_wise': True})
lgb_test = lgb.Dataset(X_test, y_test, reference=lgb_train,
free_raw_data=False, params={'force_col_wise': True})

# Train the LGBM model
num_round = 100
early_stopping_rounds = 10  # Define the early stopping rounds
bst = None
for round in range(num_round):
    bst = lgb.train(params, lgb_train, 1, valid_sets=[lgb_test],
init_model=bst)
    if round - bst.best_iteration >= early_stopping_rounds:
        break  # Stop if early stopping conditions are met

# Make predictions
y_pred_prob = bst.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print evaluation metrics and confusion matrix
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", f1)
print("Confusion Matrix:\n", conf_matrix)
```

**Long Short-Term Memory Model Code:**

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.utils.class_weight import compute_class_weight
from keras.models import Sequential
from keras.layers import LSTM, Dense
import numpy as np

# Define the window size (number of time steps before prediction)
window_size = 10

# Split the dataset into features (X) and target (y)
X = vehicle_data.drop("Cell Imbalance", axis=1)
y = vehicle_data["Cell Imbalance"]

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
shuffle=False)

# Initialize MinMaxScaler for normalization
scaler = MinMaxScaler()

# Normalize the features
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Convert the data into sliding windows
def create_sliding_windows(data, window_size):
    windows = []
    for i in range(len(data) - window_size + 1):
        window = data[i : i + window_size]
        windows.append(window)
    return np.array(windows)

X_train_windows = create_sliding_windows(X_train_scaled, window_size)
y_train_windows = y_train[window_size - 1:]

X_test_windows = create_sliding_windows(X_test_scaled, window_size)
y_test_windows = y_test[window_size - 1:]

print("X_train_windows shape:", X_train_windows.shape)
print("y_train_windows shape:", y_train_windows.shape)
print("X_test_windows shape:", X_test_windows.shape)
print("y_test_windows shape:", y_test_windows.shape)

# Calculate class weights to account for imbalance
```

```
class_weights = compute_class_weight("balanced",
classes=np.unique(y_train_windows), y=y_train_windows)
class_weight_dict = {0: class_weights[0], 1: class_weights[1]}

# Build and compile the LSTM model
model = Sequential()
model.add(LSTM(units=50, activation='relu',
input_shape=(X_train_windows.shape[1], X_train_windows.shape[2])))
model.add(Dense(units=1, activation='sigmoid'))  # Sigmoid for binary
classification
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Train the model with class weights
model.fit(X_train_windows, y_train_windows, epochs=10, batch_size=32,
class_weight=class_weight_dict)
```

**Multi Variate Multi Step Ahead Long Short-Term Memory Model Code:**

```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('359218066295457_red.csv')

# Define the input and output columns
input_cols = ['Min_Cell_V',
'Cell_V_Diff',  'Max_Cell_Temp',    'Min_Cell_Temp',    'Cell_Temp_Diff',
 'Batt_Pack_Voltage',    'Batt_Power_In',    'Charging_Current',
'Demand_Charge_Current',    'Charging_Time',    'MotorCurrent',
'MotorVoltage', 'MotorEstimatedTorque', 'BatteryPotential_PowerInput1',
'Regeneration
Power',   'ReserveModSOC',    'HVAuxilaryPowerConsumption',   'LB Battery
Voltage',   'FuelLevel',    'EngineSpeed',  'EngineOperatingHours',
'VehicleSpeed',
'EngineOilPressure',    'EngineCoolantTemp',    'AccPedalPosition']  #
Replace with your input column names
output_col = 'Max_Cell_V'

# Extract input and output data
X = data[input_cols].values
y = data[output_col].values

# Normalize the input data
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Define the number of time steps and features
n_steps = 60  # You can adjust this value
n_features = len(input_cols)

# Create sequences of data for training
X_seq, y_seq = [], []
for i in range(len(data) - n_steps + 1):
    X_seq.append(X_scaled[i:i+n_steps])
    y_seq.append(y[i+n_steps-1])

X_seq = np.array(X_seq)
y_seq = np.array(y_seq)
```

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_seq, y_seq,
test_size=0.2, random_state=42)

# Build the LSTM model
model = tf.keras.Sequential([
    tf.keras.layers.LSTM(200, activation='relu', input_shape=(n_steps,
n_features)),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(1)
])

model.compile(optimizer='Adagrad', loss='mse')

# Train the model
epochs = 500
batch_size = 512
history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size,
verbose=2, validation_data=(X_test, y_test))

# Make predictions for the testing set
y_pred = model.predict(X_test)

# Calculate RMSE and MAE for each time step from 1 to 10
rmse_scores = []
mae_scores = []
for step in range(1, 11):
    y_true_step = y_test
    y_pred_step = y_pred

    rmse = np.sqrt(mean_squared_error(y_true_step, y_pred_step))
    mae = mean_absolute_error(y_true_step, y_pred_step)

    rmse_scores.append(rmse)
    mae_scores.append(mae)

# Print RMSE and MAE scores
for step, rmse, mae in zip(range(1, 11), rmse_scores, mae_scores):
    print(f"Time Step {step}: RMSE = {rmse}, MAE = {mae}")

# Plot training and validation loss over epochs
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
# Plot the difference between predicted and actual Max_Cell_V for the first
5 units
plt.figure(figsize=(12, 6))
for step in range(1, 2):  # Limit to the first 5 time steps
    plt.subplot(2, 5, step)
    plt.plot(y_test[:10], label='Actual', color='blue')
    plt.plot(y_pred[:10], label='Predicted', color='orange')
    plt.title(f"Time Step {step}")
    plt.xlabel('Sample Index')
    plt.ylabel('Max_Cell_V')
    plt.legend()

plt.tight_layout()
plt.show()
```

**Random Forest Classifier with Correlation Analysis Code:**

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.impute import SimpleImputer
import glob

# Step 1: Read CSV files, clean data, and perform correlation analysis
def clean_and_analyze(csv_file):
    # Read CSV
    df = pd.read_csv(csv_file)
    df = df.replace(0, np.nan)  # Replace 0 values with NaN
    df = df.dropna(subset=["Max_Cell_V", "Min_Cell_V", "Max_Cell_Temp",
"Min_Cell_Temp", "EngineCoolantTemp"])
    # Correlation analysis
    max_cell_v_min_cell_v_corr = df[["Max_Cell_V",
"Min_Cell_V"]].corr().iloc[0, 1]
    max_cell_temp_min_cell_temp_corr = df[["Max_Cell_Temp",
"Min_Cell_Temp"]].corr().iloc[0, 1]
    engine_coolant_temp_corr =
df[["EngineCoolantTemp"]].corrwith(df["EngineCoolantTemp"]).values[0]
    return max_cell_v_min_cell_v_corr, max_cell_temp_min_cell_temp_corr,
engine_coolant_temp_corr

# Step 2: Define functions to identify faults and calculate fault ratios
def identify_faults(df):
    df["Cell_Imbalance_Fault"] = (
        (df["Max_Cell_V"] > 3.65) | (df["Min_Cell_V"] < 2.9) |
((df["Max_Cell_V"] - df["Min_Cell_V"]) > 0.5)
    ).astype(int)
    df["Cell_Temperature_Fault"] = (
        (df["Max_Cell_Temp"] > 55) | (df["Min_Cell_Temp"] < 15) |
((df["Max_Cell_Temp"] - df["Min_Cell_Temp"]) > 20)
    ).astype(int)
    df["Motor_Temperature_Fault"] = (
        (df["EngineCoolantTemp"] > 65) | (df["EngineCoolantTemp"] < 5)
    ).astype(int)
    return df
def calculate_fault_ratios(df):
    df["Cell_Imbalance_Fault_Ratio"] = df["Cell_Imbalance_Fault"].cumsum()
/ (df["Cell_Imbalance_Fault"].cumsum() +
df["Cell_Imbalance_Fault"].eq(0).cumsum())
```

```
    df["Cell_Temperature_Fault_Ratio"] =
df["Cell_Temperature_Fault"].cumsum() /
(df["Cell_Temperature_Fault"].cumsum() +
df["Cell_Temperature_Fault"].eq(0).cumsum())
    df["Motor_Temperature_Fault_Ratio"] =
df["Motor_Temperature_Fault"].cumsum() /
(df["Motor_Temperature_Fault"].cumsum() +
df["Motor_Temperature_Fault"].eq(0).cumsum())
    return df

# Step 3: Machine Learning Model
def train_ml_model(df):
    # Select relevant columns for training
    numeric_columns = ["Max_Cell_V", "Min_Cell_V", "Max_Cell_Temp",
"Min_Cell_Temp", "EngineCoolantTemp", "Batt_Pack_Voltage", "Batt_Power_In",
"Charging_Current", "MotorCurrent", "MotorVoltage", "MotorEstimatedTorque",
"BatteryPotential_PowerInput1", "Regeneration Power", "TotalDistance",
"FuelLevel", "EngineSpeed", "VehicleSpeed", "EngineOilPressure",
"AccPedalPosition"]
    features = df[numeric_columns]
    target_cell_imbalance = df["Cell_Imbalance_Fault"]
    target_cell_temperature = df["Cell_Temperature_Fault"]
    target_motor_temperature = df["Motor_Temperature_Fault"]
    # Split data into training and testing sets (use the same split for all
three models)
    features_train, features_test, y_train_cell_imbalance,
y_test_cell_imbalance = train_test_split(features, target_cell_imbalance,
test_size=0.2, random_state=42)
    _, _, y_train_cell_temperature, y_test_cell_temperature =
train_test_split(features, target_cell_temperature, test_size=0.2,
random_state=42)
    _, _, y_train_motor_temperature, y_test_motor_temperature =
train_test_split(features, target_motor_temperature, test_size=0.2,
random_state=42)
    # Build and train models
    model_cell_imbalance = make_pipeline(StandardScaler(),
RandomForestClassifier(n_jobs=-1))
    model_cell_imbalance.fit(features_train, y_train_cell_imbalance)
    model_cell_temperature = make_pipeline(StandardScaler(),
RandomForestClassifier(n_jobs=-1))
    model_cell_temperature.fit(features_train, y_train_cell_temperature)
    model_motor_temperature = make_pipeline(StandardScaler(),
RandomForestClassifier(n_jobs=-1))
    model_motor_temperature.fit(features_train, y_train_motor_temperature)

    # Make predictions
    predictions_cell_imbalance =
model_cell_imbalance.predict_proba(features_test)[:, 1] if
```

```
model_cell_imbalance.classes_.shape[0] > 1 else
model_cell_imbalance.predict_proba(features_test)
    predictions_cell_temperature =
model_cell_temperature.predict_proba(features_test)[:, 1] if
model_cell_temperature.classes_.shape[0] > 1 else
model_cell_temperature.predict_proba(features_test)
    predictions_motor_temperature =
model_motor_temperature.predict_proba(features_test)[:, 1] if
model_motor_temperature.classes_.shape[0] > 1 else
model_motor_temperature.predict_proba(features_test)
    # Calculate accuracy
    accuracy_cell_imbalance = accuracy_score(y_test_cell_imbalance,
model_cell_imbalance.predict(features_test))
    accuracy_cell_temperature = accuracy_score(y_test_cell_temperature,
model_cell_temperature.predict(features_test))
    accuracy_motor_temperature = accuracy_score(y_test_motor_temperature,
model_motor_temperature.predict(features_test))
    return predictions_cell_imbalance, predictions_cell_temperature,
predictions_motor_temperature, accuracy_cell_imbalance,
accuracy_cell_temperature, accuracy_motor_temperature

# Step 5: Create Excel and CSV files
def create_output_files(csv_file, prob_cell_imbalance,
prob_cell_temperature, prob_motor_temperature):
    try:
        result_df = pd.read_excel("output_results.xlsx")
    except FileNotFoundError:
        result_df = pd.DataFrame(columns=["File Name", "Probability of
Vehicle failure due to Cell Imbalance", "Probability of Vehicle failure due
to Cell Temperature", "Probability of Vehicle failure due to Motor
Temperature"])
    new_row = {
        "File Name": csv_file,
        "Probability of Vehicle failure due to Cell Imbalance":
prob_cell_imbalance,
        "Probability of Vehicle failure due to Cell Temperature":
prob_cell_temperature,
        "Probability of Vehicle failure due to Motor Temperature":
prob_motor_temperature
    }
    result_df = pd.concat([result_df, pd.DataFrame([new_row])],
ignore_index=True)
    result_df.to_excel("output_results.xlsx", index=False)


# Main Loop through CSV files
csv_files = glob.glob("*.csv")
for csv_file in csv_files:
```

```
    # Step 1
    max_cell_v_min_cell_v_corr, max_cell_temp_min_cell_temp_corr,
engine_coolant_temp_corr = clean_and_analyze(csv_file)
    # Step 2
    df = pd.read_csv(csv_file)
    df = identify_faults(df)
    df = calculate_fault_ratios(df)
    # Step 3
    predictions_cell_imbalance, predictions_cell_temperature,
predictions_motor_temperature, \
    accuracy_cell_imbalance, accuracy_cell_temperature,
accuracy_motor_temperature = train_ml_model(df)
    # Step 4
    output_df = df[["Unnamed: 0", "Max_Cell_V", "Min_Cell_V",
Max_Cell_Temp", "Min_Cell_Temp", "Batt_Pack_Voltage", "Batt_Power_In",
"Charging_Current", "MotorCurrent", "MotorVoltage", "MotorEstimatedTorque",
"BatteryPotential_PowerInput1", "Regeneration Power", "IST_DateTime",
"TotalDistance", "FuelLevel", "EngineSpeed", "VehicleSpeed",
"EngineOilPressure", "AccPedalPosition", "Cell_Imbalance_Fault",
"Cell_Temperature_Fault", "Motor_Temperature_Fault",
"Cell_Imbalance_Fault_Ratio", "Cell_Temperature_Fault_Ratio",
"Motor_Temperature_Fault_Ratio"]]
    output_csv_file = csv_file.replace(".csv", "_output.csv")
    output_df.to_csv(output_csv_file, index=False)
    # Step 5
    create_output_files(csv_file, df["Cell_Imbalance_Fault_Ratio"].iloc[-
1], df["Cell_Temperature_Fault_Ratio"].iloc[-1],
df["Motor_Temperature_Fault_Ratio"].iloc[-1])
```

**Logic Model Codes:**

**Cell Imbalance Use Case Code:**

```
import pandas as pd
import os

# Function to calculate additional metrics
def calculate_additional_metrics(df):
    df = df.copy()  # Create a copy to avoid SettingWithCopyWarning
    total_entries = len(df)
    max_cell_v_all_entries = df['Max_Cell_V'].max()
    min_cell_v_all_entries = df['Min_Cell_V'].min()
    # Use loc to avoid SettingWithCopyWarning
    df.loc[:, 'Cell_V_Diff'] = df['Max_Cell_V'] - df['Min_Cell_V']
    cell_v_diff_threshold = 0.5
    cell_v_diff_crosses = len(df[df['Cell_V_Diff'] >
cell_v_diff_threshold])
    return total_entries, max_cell_v_all_entries, min_cell_v_all_entries,
cell_v_diff_crosses
max_cell_v_threshold = 3.65
min_cell_v_threshold = 2.9
csv_files = [f for f in os.listdir() if f.endswith('.csv')]
excel_writer = pd.ExcelWriter('results.xlsx', engine='xlsxwriter')
# Create an empty DataFrame to store results
all_results_df = pd.DataFrame()
for csv_file in csv_files:
    df = pd.read_csv(csv_file)
    df = df.sort_values(by='IST_DateTime')
    # Filter based on conditions
    df_filtered = df[(df['Max_Cell_V'] != 0) | (df['Min_Cell_V'] != 0)]
    # Calculate additional metrics
    total_entries, max_cell_v_all_entries, min_cell_v_all_entries,
cell_v_diff_crosses = calculate_additional_metrics(df_filtered)
    result = {
        "File Name": csv_file[:18],  # Truncate to 18 characters
        "Frequency of 'Max_Cell_V' crossing threshold":
len(df_filtered[df_filtered['Max_Cell_V'] > max_cell_v_threshold]),
        "Frequency of 'Min_Cell_V' crossing threshold":
len(df_filtered[df_filtered['Min_Cell_V'] < min_cell_v_threshold]),
        "Frequency of 'Cell_V_Diff' crossing threshold":
cell_v_diff_crosses,
        "Total Entries in the Vehicle": total_entries,
        "Max_Cell_V out of All Entries": max_cell_v_all_entries,
        "Min_Cell_V out of All Entries": min_cell_v_all_entries
    }
    result_df = pd.DataFrame([result])
```

```
    # Concatenate results to the main DataFrame
    all_results_df = pd.concat([all_results_df, result_df],
ignore_index=True)
# Write the combined results to a single sheet
all_results_df.to_excel(excel_writer, sheet_name='CombinedResults',
index=False)
workbook = excel_writer.book
worksheet = excel_writer.sheets['CombinedResults']
for i, col in enumerate(all_results_df.columns):
    max_len = max(all_results_df[col].astype(str).apply(len).max(),
len(col) + 2)
    worksheet.set_column(i, i, max_len)
excel_writer.close()
```

**Cell Temperature Use Case Code:**

```
import pandas as pd
import os

# Function to calculate additional metrics
def calculate_additional_metrics(df, max_temp_threshold,
min_temp_threshold):
    df = df.copy()  # Create a copy to avoid SettingWithCopyWarning
    total_entries = len(df)
    max_cell_temp_all_entries = df['Max_Cell_Temp'].max()
    min_cell_temp_all_entries = df['Min_Cell_Temp'].min()
    return total_entries, max_cell_temp_all_entries,
min_cell_temp_all_entries
max_temp_threshold = 55
min_temp_threshold = 15
csv_files = [f for f in os.listdir() if f.endswith('.csv')]
excel_writer = pd.ExcelWriter('results.xlsx', engine='xlsxwriter')
# Create an empty DataFrame to store results
all_results_df = pd.DataFrame()
for csv_file in csv_files:
    df = pd.read_csv(csv_file)
    df = df.sort_values(by='IST_DateTime')
    # Filter based on conditions
    df_filtered = df[(df['Max_Cell_Temp'] != 0) | (df['Min_Cell_Temp'] !=
0)]
    # Calculate additional metrics
    total_entries, max_cell_temp_all_entries, min_cell_temp_all_entries =
calculate_additional_metrics(df_filtered, max_temp_threshold,
min_temp_threshold)
    result = {
        "File Name": csv_file[:18],  # Truncate to 18 characters
        "Frequency of 'Max_Cell_Temp' crossing threshold":
len(df_filtered[df_filtered['Max_Cell_Temp'] > max_temp_threshold]),
        "Frequency of 'Min_Cell_Temp' crossing threshold":
len(df_filtered[df_filtered['Min_Cell_Temp'] < min_temp_threshold]),
        "Total Entries in the Vehicle": total_entries,
        "Max_Cell_Temp out of All Entries": max_cell_temp_all_entries,
        "Min_Cell_Temp out of All Entries": min_cell_temp_all_entries
    }
    result_df = pd.DataFrame([result])
    # Concatenate results to the main DataFrame
    all_results_df = pd.concat([all_results_df, result_df],
ignore_index=True)
# Write the combined results to a single sheet
all_results_df.to_excel(excel_writer, sheet_name='CombinedResults',
index=False)
workbook = excel_writer.book
```

```
worksheet = excel_writer.sheets['CombinedResults']
for i, col in enumerate(all_results_df.columns):
    max_len = max(all_results_df[col].astype(str).apply(len).max(),
len(col) + 2)
    worksheet.set_column(i, i, max_len)
excel_writer.close()
```

**Motor Temperature Use Case Code:**

```
import pandas as pd
import os

# Function to calculate additional metrics
def calculate_additional_metrics(df, max_temp_threshold,
min_temp_threshold):
    df = df.copy()  # Create a copy to avoid SettingWithCopyWarning
    total_entries = len(df)
    max_motor_temp_all_entries = df['EngineCoolantTemp'].max()
    min_motor_temp_all_entries = df['EngineCoolantTemp'].min()
    return total_entries, max_motor_temp_all_entries,
min_motor_temp_all_entries
max_temp_threshold = 65
min_temp_threshold = 4
csv_files = [f for f in os.listdir() if f.endswith('.csv')]
excel_writer = pd.ExcelWriter('results.xlsx', engine='xlsxwriter')
# Create an empty DataFrame to store results
all_results_df = pd.DataFrame()
for csv_file in csv_files:
    df = pd.read_csv(csv_file)
    df = df.sort_values(by='IST_DateTime')
    # Filter based on conditions
    df_filtered = df[(df['Max_Cell_Temp'] != 0) | (df['Min_Cell_Temp'] !=
0) | (df['EngineCoolantTemp'] != 0)]
    # Calculate additional metrics
    total_entries, max_motor_temp_all_entries, min_motor_temp_all_entries =
calculate_additional_metrics(df_filtered, max_temp_threshold,
min_temp_threshold)
    result = {
        "File Name": csv_file[:18],  # Truncate to 18 characters
        "Frequency of 'Max_Motor_Temp' crossing threshold":
len(df_filtered[df_filtered['EngineCoolantTemp'] > max_temp_threshold]),
        "Frequency of 'Min_Motor_Temp' crossing threshold":
len(df_filtered[df_filtered['EngineCoolantTemp'] < min_temp_threshold]),
        "Total Entries in the Vehicle": total_entries,
        "Max_Motor_Temp out of All Entries": max_motor_temp_all_entries,
        "Min_Motor_Temp out of All Entries": min_motor_temp_all_entries
    }
    result_df = pd.DataFrame([result])
    # Concatenate results to the main DataFrame
    all_results_df = pd.concat([all_results_df, result_df],
ignore_index=True)
# Write the combined results to a single sheet
all_results_df.to_excel(excel_writer, sheet_name='CombinedResults',
index=False)
workbook = excel_writer.book
```

```
worksheet = excel_writer.sheets['CombinedResults']
for i, col in enumerate(all_results_df.columns):
    max_len = max(all_results_df[col].astype(str).apply(len).max(),
len(col) + 2)
    worksheet.set_column(i, i, max_len)
excel_writer.close()
```

# REFERENCES


1. Chen, J., Zhou, Z., Zhou, Z., Wang, X., & Liaw, B. (2022). Impact of battery cell imbalance on electric vehicle range. Green Energy and Intelligent Transportation, 1(3), 100025. (DOI: 10.1016/j.geits.2022.100025)
2. Du, S., Li, T., Yang, Y., & Horng, S.-J. (2020). Multivariate time series forecasting via the attention-based encoder-decoder framework. Neurocomputing, 388, 269-279. (DOI: 10.1016/j.neucom.2019.12.118)
3. Thomas, J. K., Crasta, H. R., Kausthubha, K., Gowda, C., & Rao, A. (2021). Battery monitoring system using machine learning. Journal of Energy Storage, 40, 102741. (DOI: 10.1016/j.est.2021.102741)
4. He, X., Shi, S., Geng, X., Yu, J., & Xu, L. (2023). Multi-step forecasting of multivariate time series using multi-attention collaborative network. Expert Systems with Applications, 211, 118516. (DOI: 10.1016/j.eswa.2022.118516)
5. De Stefani, J., & Bontempi, G. (2021). Factor-Based Framework for Multivariate and Multi-step-ahead Forecasting of Large-Scale Time Series. Frontiers in Big Data, 4, 690267. (DOI: 10.3389/fdata.2021.690267)
6. Akbar, K., Zou, Y., Awais, Q., Baig, M. J. A., & Jamil, M. (2022). A Machine Learning-Based Robust State of Health (SOH) Prediction Model for Electric Vehicle Batteries. Electronics, 11(8), 1216. (DOI: 10.3390/electronics11081216)
7. Li, K., Zhou, P., Lu, Y., Han, X., & Zheng, Y. (2020). Battery life estimation based on cloud data for electric vehicles. Journal of Power Sources, 468, 228192. (DOI: 10.1016/j.jpowsour.2020.228192)
8. Kulkarni, V., Han, X., & Tjong, J. (2021). Intelligent Detection and Real-time Monitoring of Engine Oil Aeration Using a Machine Learning Model. Applied Artificial Intelligence, 35(15), 1869-1886. (DOI: 10.1080/08839514.2021.1995230)
9. Andrea, D. (2010). Battery Management Systems for Large Lithium-Ion Battery Packs. ARTECH HOUSE. pp. 22-30 (ISBN-13: 978-1-60807-104-3).
10. Pang, K. (2022). Multi-step multivariate time series forecasting using LSTM. Retrieved from https://pangkh98.medium.com/multi-step-multivariate-time-series-forecasting-using-lstm-92c6d22cd9c2
11. ML Review. (n.d.). Understanding LSTM and its diagrams. Retrieved from https://blog.mlreview.com/understanding-lstm-and-its-diagrams-37e2f46f1714

12. Bond, W & Dozier, Haley & Arnold, Thomas & Lam, Michael-Angelo & Dong, Quyen & Shukla, Indu & Hansen, et. al (2020). A Hybrid Learning Approach to Prognostics and Health Management Applied to Military Ground Vehicles Using Time-Series and Maintenance Event Data. Annual Conference of the PHM Society. (DOI: 10.10.36001/phmconf.2020.v12i1.1146)
13. Jingyuan Z, Heping L, Junbin W, Andrew F. B, Yubo L. (2022). Data-driven prediction of battery failure for electric vehicles, iScience, 25(4), 104172. (DOI: 10.1016/j.isci.2022.104172)
14. Malaguti, R., Lourenço, N., & Silva, C. (2022). A supervised machine learning model for determining lubricant oil operating conditions. Expert Systems, 40(5). (DOI: 10.1111/exsy.13116)
15. Jun C, Zhaodong Z, Ziwei Z, Xia W, Boryann L. (2022).Impact of battery cell imbalance on electric vehicle range, Green Energy and Intelligent Transportation, 1(3), 100025. (DOI: 10.1016/j.geits.2022.100025)
16. Gabriele P, Anirudh A, Simona O. (2022). Lithium-ion battery aging dataset based on electric vehicle real-driving profiles, Data in Brief, 41(1), 107995. (DOI: 10.1016/j.dib.2022.107995)
17. S. Vasavi, K. Aswarth, T. Sai Durga Pavan, A. Anu Gokhale. (2021). Predictive analytics as a service for vehicle health monitoring using edge computing and AK-NN algorithm, Materials Today: Proceedings, 46 (17), 8645-8654, (DOI: 10.1016/j.matpr.2021.03.658)
18. Arena F, Collotta M, Luca L, Ruggieri M, Termine FG. (2022) Predictive Maintenance in the Automotive Sector: A Literature Review. Mathematical and Computational Applications. 27(1). (DOI: 10.3390/mca27010002)
19. D. N. Demyanov, L. A. Simonova and A. A. Kapitonov. (2020). The Work Algorithm of the Truck Intelligent Predictive Diagnostics System. International Russian Automation Conference (RusAutoCon), 621-625. (DOI: 10.1109/RusAutoCon49822.2020.9208065)