# Integration of Machine Learning and Explainable AI (XAI) approaches for Optimizing Battery Health on Electric Vehicles

*Project report submitted to the SASTRA Deemed to be University in partial fulfilment of the requirements for the award of the degree of*

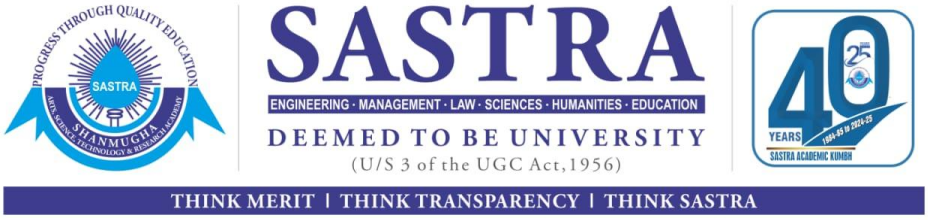
## MASTER OF SCIENCE IN COMPUTER SCIENCE

*Submitted by*

**PAKTHAPRIYAN S**
**225058031**

Under the supervision of

**Dr. R. BALA KRISHNAN**
Assistant Professor, Department of CSE




**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**Srinivasan Ramanujam Centre**

**Kumbakonam - 612001**

**Tamil Nadu, India**

**May 2025**

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act,1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

## Srinivasa Ramanujan Centre

**Kumbakonam —612001**

**Tamil Nadu, India**

## BONAFIDE CERTIFICATE

This is to verify that the project report titled "**Integration of Machine Learning and Explainable AI (XAI) approaches for Optimizing Battery Health on Electric Vehicles**" submitted to the Srinivasa Ramanujan Centre, SASTRA Deemed to be University, Kumbakonam by **PAKTHAPRIYAN S (225058031)** in partial fulfilment of the requirements for the award of the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE** work carried out under the supervision of **Dr. R. BALA KRISHNAN**, Assistant Professor, Department of CSE during the period January 2025 to May 2025.

Signature of Project Supervisor :

Name with Affiliation : **Dr. R. Bala Krishnan**,
Assistant Professor,
Department of CSE.

Date :

Project Viva Voce held on :

Examiner I Examiner II

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**Srinivasa Ramanujan Centre**

**Kumbakonam - 612001**

**Tamil Nadu, India**

## DECLARATION

I submit this project work entitled "**Integration of Machine Learning and Explainable AI (XAI) approaches for Optimizing Battery Health on Electric Vehicles**" to Srinivasa Ramanujan Centre, SASTRA Deemed to be University, Kumbakonam –612 001, in partial fulfilment of the requirements for the award of the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE**. I declare that this is my original work carried out under the guidance of **Dr. R. Bala Krishnan,** Assistant Professor, Department of Computer Science and Engineering, Srinivasa Ramanujan Centre.

Place : Kumbakonam

Date :

Name : **Pakthapriyan S**

Reg No : **225058031**

Signature :

# ACKNOWLEDGEMENT

I pay my sincere pranam to **God ALMIGHTY** for his grace and infinite mercy and for showing me his choicest blessings.

I would like to express our sincere gratitude to our Honorable Chancellor, **Prof. R. Sethuraman**, for providing the opportunity and the necessary infrastructure to carry out this project as part of our curriculum.

I am deeply thankful to our Honorable Vice-Chancellor, **Dr. S. Vaidhyasubramaniam**, and **Dr. S. Swaminathan**, Dean, Planning & Development, for their constant encouragement and strategic support throughout our college journey. We also extend our sincere thanks to **Dr**. **R. Chandramouli**, Registrar, SASTRA Deemed to be University, for providing us with the opportunity to pursue this project.

My heartfelt thanks go to **Dr. V. Ramaswamy**, Dean, and **Dr. A. Alli Rani**, Associate Dean, Srinivasa Ramanujan Centre, SASTRA Deemed to be University. We are also pleased to express our gratitude to **Dr. V. Kalaichelvi**, Associate Professor, Department of Computer Science and Engineering, Srinivasa Ramanujan Centre, for her encouragement and support during our project work

I express our sincere gratitude to my guide, **Dr. R. Bala Krishnan**, Assistant Professor - II, Department of Computer Science and Engineering, Srinivasa Ramanujan Centre, whose unwavering support, deep insights and invaluable suggestions were instrumental in driving this project forward from its inception to successful completion.

I also thank our project coordinators, **Dr. N. Saravanan**, Assistant Professor - III and **Dr. P. Venkateswari,** Assistant Professor-II, Department of Computer Science and Engineering, Srinivasa Ramanujan Centre, for their valuable feedback and insights that significantly enhanced the quality of our work. I owe my sincere thanks to all faculty members in the department who have directly or indirectly helped me in completing this project.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ABBREVIATIONS

EV - Electric Vehicle(s)

FN - False Negative(s)

FNR - False Negative Rate

FP - False Positive(s)

FPR - False Positive Rate

ML - Machine Learning

NPV - Negative Predictive Value

PPV - Positive Predictive Value

RUL - Remaining Useful Life

SHAP - SHapley Additive exPlanations

SOH - State of Health

TN - True Negative(s)

TNR - True Negative Rate

TP - True Positive(s)

TPR - True Positive Rate

XAI - Explainable Artificial Intelligence

# ABSTRACT

# Integration of Machine Learning and Explainable AI (XAI) approaches for Optimizing Battery Health on Electric Vehicles


## Abstract

Machine Learning (ML) empowered the EV to reduce energy consumption intelligently and make driving more comfortable; hence, enabling maintenance forecast capabilities and explaining in detail an otherwise opaque model in machine learning increases transparency in order to encourage trust in their outputs. However, XAI was invented to counter concerns of this transparency with an understanding into its decision-making. Integration of XAI may help to develop more user-friendly and trustworthy EV systems that improve not only performance but also user acceptance and understanding of advanced technologies. This study conducts an investigation into the synergistic integration of Machine Learning— Explainable Artificial Intelligence (XAI) techniques within an electric vehicle (EV) context. Integrating XAI (SHAP) with ML based ensemble classifiers are utilized for optimizing EV systems components, which would contribute to the development of more user-friendly and trustworthy systems. XAI will help us identify relevant features that drive predictions that will also help us better understand the battery degradation phenomenon. The Kaggle repository's EV dataset is used to verify the efficiency of the proposed system with metrics associated with confusion matrix. The proposed scheme would be expected for an efficient outcome with an optimistic accuracy with minimum features for the battery health evaluation process.



**Keywords:** EV Battery Health, Battery Health Classification, Machine Learning, Explainable AI (XAI), Ensemble Classifiers

# CHAPTER 1

# INTRODUCTION

# 1. INTRODUCTION

## 1.1 Background:

The global shift towards sustainable transport has put Electric Vehicles (EVs) at the forefront of automotive technology. EVs have many advantages over traditional internal combustion engine cars, including reduced emissions, reduced running costs, and often enhanced performance characteristics. At the core of all EVs is the battery system, a highly sophisticated and important system whose health immediately impacts the range, efficiency, safety, and overall vehicle longevity. Optimum battery health is therefore paramount to maximize EV performance and long-term user satisfaction and confidence.

Effective battery management relies on advanced systems monitoring current status and predicting future performance. Machine Learning (ML) provides valuable data-driven techniques for estimating State of Charge (SOC), State of Health (SOH), and forecasting Remaining Useful Life (RUL). By analyzing extensive operational and charging data, ML models can identify subtle degradation patterns, enabling proactive maintenance and optimized charging strategies.

## 1.2 Problem Statement:

Sophisticated ML models, especially complex ensembles, have achieved high accuracy in predicting or classifying EV battery health. their very nature tends to render them "black boxes." The models may make very accurate predictions, but they do not usually provide transparent, human-understandable explanations of why a particular prediction was made. This lack of transparency creates problems of considerable magnitude, especially for high-risk applications like battery health management where reliability and safety are paramount. Stakeholders like engineers, maintenance personnel, and even car owners may struggle to trust or act on black-box model predictions. Furthermore, the black-box nature prevents the possibility of obtaining greater scientific insight into the complex underlying processes that govern battery degradation. There clearly is a need to enhance the interpretability of such powerful ML models to build trust, facilitate

better debugging, and allow better understanding of the factors affecting battery health, Where reliability and safety are the topmost concerns. Individuals like engineers, maintenance personnel, or even car owners are confronted with problems in trusting or acting on black-box model responses.

## 1.3 Objectives and Scope:

This project aims to address the problem of explainability of ML models for the application of EV battery health estimation by symbiotically integrating Machine Learning and Explainable AI (XAI) techniques. This work aims to:

1. To Develop and validate ensemble machine learning classifiers for accurate EV battery health status classification.
2. To Employ Shapley Additive explanations (SHAP) to interpret the ML models' predictions.
3. To apply SHAP explanations to discover and examine the most important features that greatly impact battery health predictions.
4. To illustrate how the XAI explanations can be used to enhance the understanding of the underlying battery degradation process.
5. To compare the performance of the ML models and the learned knowledge from XAI on an actual EV dataset from a Kaggle library.

The scope of the project is battery health classification and SHAP interpretation of the classifications. While the information gained can be used to suggest more general battery management processes or optimization, the direct outcome is an interpretable classification model trained for battery health evaluation from the provided dataset.

## 1.4 Proposed Approach:

The Proposed approach is to build and compare various ensemble classification models suitable for predicting various battery health classes. Following training and performance evaluation of models in terms of metrics derived from the confusion matrix, SHAP analysis will be conducted. SHAP values will be computed on individual predictions to facilitate local

explanations (why a specific battery instance was predicted as it was predicted) and aggregated to facilitate global explanations (why specific features tend to be of greatest importance to the model's predictions). By doing so, we will not just be able to confirm model performance but also gain actionable insights into the causes of battery health degradation as modeled.

# CHAPTER 2

# OBJECTIVE

## 2. OBJECTIVES

This project aims to improve EV battery health understanding and forecasting by integrating machine learning (ML) and explainable AI (XAI). Specifically, we will develop robust ensemble classification models for individual battery health forecasting and use SHAP to interpret these models' decision-making. Through the implementation of SHAP analysis to the results of predictions carried out on an EV dataset, we aim to identify the key features with high contributions towards the classification of battery health, thus gaining significant knowledge about the complex factors underlying battery degradation and enhancing the trustworthiness of the predictive models deployed in the battery management.

# CHAPTER 3

# SYSTEM REQUIREMENTS

# 3. SYSTEM SPECIFICATIONS

## 3.1 HARDWARE SPECIFICATIONS:

| | |
|---|---|
| System | : AMD Ryzen 7 5800HS with Radeon Graphics 3 .20 GHz |
| Hard Disk | : 500 GB |
| RAM | : 16.0 GB |

## 3.2 SOFTWARE SPECIFICATIONS:

| | |
|---|---|
| OS | : Windows 11 Home Single Language |
| Programming Language | : Python 3.11.9 |
| Platform | : Jupyter Notebook 7.3.2 |

# CHAPTER 4
# LITERATURE SURVEY

# 4. LITERATURE SURVEY

**HyPELS: Enhancing Li-Ion Battery Remaining Useful Life Prediction with Hybrid Perspective Ensemble Learning Strategy** by Xuewei Han, Huimei Yuan, and Lifeng Wu, published in *Engineering Research Express* in November 2024, introduces a novel approach to predicting the remaining useful life (RUL) of lithium-ion batteries. The proposed Hybrid Perspective Ensemble Learning Strategy (HyPELS) integrates two distinct perspectives: battery health indicators (HIs) and capacity degradation data. Experimental results demonstrate that HyPELS outperforms traditional methods across multiple datasets, showcasing enhanced prediction accuracy and robustness. The integration of both perspectives allows for a comprehensive understanding of battery aging, capturing multi-dimensional characteristics effectively.[1]

**Enhancing State of Health Prediction Accuracy in Lithium-Ion Batteries through a Simplified Health Indicator Method** by Dongxu Han, Nan Zhou, and Zeyu Chen, published in *Batteries* (2024), introduces a novel approach to predicting the state of health (SOH) of lithium-ion batteries. The authors propose a simplified health indicator method that utilizes the area under the incremental capacity (IC) curve, derived from charge and discharge data, to construct a weighted health indicator sequence. The study also explores the feasibility of incorporating impedance as an additional input, despite measurement challenges. Validation with different datasets demonstrates that the proposed method achieves an average relative error and root mean square error within 5%, outperforming other methods in terms of minimizing error and ensuring stability.[2]

**Stage-Based Remaining Useful Life Prediction for Bearings Using GNN and Correlation-Driven Feature Extraction** by Guangzhong Huang, Wenping Lei, Xinmin Dong, Dongliang Zou, Shijin Chen, and Xing Dong, published in *Machines* (2025), presents an advanced methodology for predicting the remaining useful life (RUL) of bearings. This approach integrates correlation-driven feature extraction with Graph Neural Networks (GNNs) to address the complex degradation stages of bearings Experimental validation demonstrates that the proposed model outperforms traditional methods in accurately predicting bearing RUL, effectively capturing the

multi-stage degradation process inherent in bearing performance.[3]

**Advanced Data-Driven Techniques in AI for Predicting Lithium-Ion Battery Remaining Useful Life: A Comprehensive Review** by Sijing Wang, Ruoyu Zhou, Yijia Ren, Meiyuan Jiao, Honglai Liu, and Cheng Lian, published in *Green Chemical Engineering* (2024), provides an extensive overview of artificial intelligence (AI) methodologies applied to predict the remaining useful life (RUL) of lithium-ion batteries.[4]

**A Scientific Machine Learning Approach for Predicting and Forecasting Battery Degradation in Electric Vehicles** by Murgai et al. (2024) introduces a hybrid approach combining domain knowledge with machine learning to predict lithium-ion battery degradation in electric vehicles (EVs). The authors propose using Neural Ordinary Differential Equations (NeuralODEs) for improved interpretability and accuracy. The model outperforms traditional approaches, achieving a low Mean Squared Error (MSE) of 9.90 with the Uncertainty-aware Degradation Estimator (UDE). This approach enhances battery lifespan predictions, contributing to sustainable energy solutions by reducing waste and improving EV performance, aligning with environmental goals.[5]

**State of Health Estimation for Li-Ion Battery Using Machine Learning** by Sharma et al. (2023) explores the use of machine learning techniques to estimate the state of health (SOH) of Li-Ion batteries.The authors present various machine learning models, focusing on their effectiveness in predicting battery degradation and performance. By leveraging data-driven approaches, the study aims to enhance battery monitoring systems and improve the reliability of energy storage technologies in applications like electric vehicles and portable electronics.[6]

**Machine Learning-Based Approach for Predicting Battery Remaining Useful Life (RUL): A Data-Driven Methodology** by Devi and V. Suresh Kumar (2023) presents a data-driven methodology utilizing machine learning techniques to predict the remaining useful life (RUL) of batteries. Accurate RUL prediction is crucial for effective battery management in various applications, including electric vehicles and renewable energy systems. The authors propose a machine learning-based approach that leverages historical battery data to model and predict the

RUL, aiming to enhance the reliability and efficiency of battery-operated systems.[7]

**Battery Health Prediction Using Deep Hybrid Learning** by Nair, Suresh, and Rajeswari (2023) introduces a novel deep hybrid learning model to predict the health status of lithium-ion batteries. The proposed approach integrates convolutional neural networks (CNNs) with long short-term memory (LSTM) networks, leveraging CNNs for feature extraction from battery data and LSTMs for temporal sequence modeling. This hybrid architecture aims to enhance prediction accuracy by capturing both spatial and temporal dependencies in battery performance data. The study demonstrates the model's effectiveness in forecasting battery degradation, contributing to improved battery management and lifespan prediction.[8]

**Deep Learning-Based Approach for State-of-Health Estimation of Lithium-Ion Battery in the Electric Vehicles** by Aagya Niraula and Jai Govind Singh (2023) presents a deep learning methodology to assess the state-of-health (SoH) of lithium-ion batteries in electric vehicles (EVs). Given the critical role of battery health in EV performance and safety, the authors propose a model that leverages deep learning techniques to predict battery degradation and estimate its remaining useful life. This approach aims to enhance battery management systems, ensuring optimal performance and longevity of EV batteries.[9]

**Battery Health Prediction Using Deep Hybrid Learning** by Rahul Anil Nair, Dev Karan Suresh, and Dr. Rajeswari D. (2023) presents a novel approach to predicting electric vehicle (EV) battery health by integrating deep learning techniques with satellite imagery. The study utilizes pre-trained models—VGG16, ResNet50, and VGG19—combined with boosting algorithms like XGBoost and AdaBoost to classify EV routes based on battery health indicators. The dataset comprises vehicle telemetry data such as latitude, longitude, battery state of charge, air conditioning power, heater power, battery voltage, current, and vehicle speed, which were used to generate satellite images of the vehicle paths via Google Earth. These images were then categorized into 'best' and 'worst' routes based on the number of brakes applied during the journey. The deep hybrid learning models achieved classification accuracies ranging from 79.25% to 87.63%, demonstrating the potential of this approach in enhancing EV battery management systems.[10]

# CHAPTER 5

# METHODOLOGY

## 5. METHODOLOGY

This chapter outlines the methodology steps adopted to complete the project objectives, focusing on data collection and preprocessing, followed by model building, evaluation, and deployment of Explainable AI (XAI) methods. The methodology combines selected high-performing Machine Learning classification models with the explanatory strength of the SHapley Additive exPlanations (SHAP) method.

### 5.1 Data Collection and Importation:

The initial step of the project was to acquire the dataset needed for model training and testing. Information pertaining to the properties of Electric Vehicle (EV) battery and its operating parameters, and a target variable for battery health classification (Battery_Class), was acquired from a publicly available dataset in a Kaggle repository. The dataset, with approximately 100,000 records, was used as the empirical basis for the development and testing of the predictive models. The data was structured into a format suitable for further processing and analysis using standard data manipulation libraries.

### 5.2 Data Preparation:

Data preprocessing is a critical phase of ensuring both the dataset and data relevance for Machine Learning model training. The process involves a set of critical steps:

#### 5.2.1 Handling Incomplete Data:

A check of the dataset revealed that there were missing values in certain features. To address this, an imputation step was performed. A SimpleImputer with a 'mean' strategy was used to replace missing numeric values with the mean of the respective feature column. This is useful in preserving the dataset size and making the most of the available information without exacerbating the incompleteness of the data issue.

### 5.2.2 Data Splitting:

After missing value handling, the data was divided into training and test sets. Independent variables (X) were separated from the dependent variable (y, named as Battery_Class). This was followed by dividing the data into training and test sets using train_test_split function, keeping 80% of the dataset to train models and keeping the remaining 20% to test the performance of the model on data not seen by the model previously. The same random_state was utilized to make the split reproducible.

## 5.3 Model Training and Selection:

Based on their established performance properties and applicability to the dataset, a targeted selection of three high-performance classifiers was performed: XGBoost, CatBoost, and ExtraTreesClassifier. These tree ensemble and boosting-based algorithms were selected for their stability and performance in detecting intricate patterns, and for their capacity to accommodate tree-specific explanation techniques such as SHAP. The models were trained on the preprocessed training data using their respective implementations in standard ML libraries.

## 5.4 Model Evaluation:

After the training process, the performance of each selected model—XGBoost, CatBoost, and ExtraTreesClassifier—was extensively evaluated utilizing the given test set. Predictions were generated for the test samples, and these predictions were compared with the true target values. For each model, the common classification metrics utilizing the confusion matrix were computed, including accuracy, precision, recall, and F1-score. Furthermore, a classification report was generated to provide detailed performance metrics for various battery health classes for all three models. 5.5 Explainable AI (XAI) Application: A key component of this project is the use of Explainable Artificial Intelligence (XAI) to interpret the predictions of the trained models. Specifically, the SHapley Additive exPlanations (SHAP) framework was used. Given that XGBoost, CatBoost, and ExtraTreesClassifier are tree-based models, the effective shap.TreeExplainer was used. The SHAP values were approximated for the test dataset for all three trained models, thus approximating the contribution of each input feature to the prediction for each individual instance. This process supplied the necessary data for the following analysis of feature

importance and effect. 5.6 Analysis and Interpretation of Results: The second phase involved the analysis of the evaluation metrics in conjunction with the interpretation of SHAP results obtained for XGBoost, CatBoost, and ExtraTreesClassifier. Model performance metrics comparison across the three classifiers was conducted to determine their respective strengths. The SHAP values obtained from them were then depicted and analyzed through different approaches, including summary plots, dependence plots, and force plots. The analytical method facilitated the determination of the most contributing global features to each model's prediction, a glimpse into the interaction between the values of specific features and the expected battery health outcome, and understandable explanations at the instance level. The results obtained through this rigorous analysis proved to be key in advancing the understanding of empirical drivers of battery health classification and provided insight into the phenomenon of battery degradation evidenced in the dataset.

# CHAPTER 6

# IMPLEMENTATION

# 6. IMPLEMENTATION

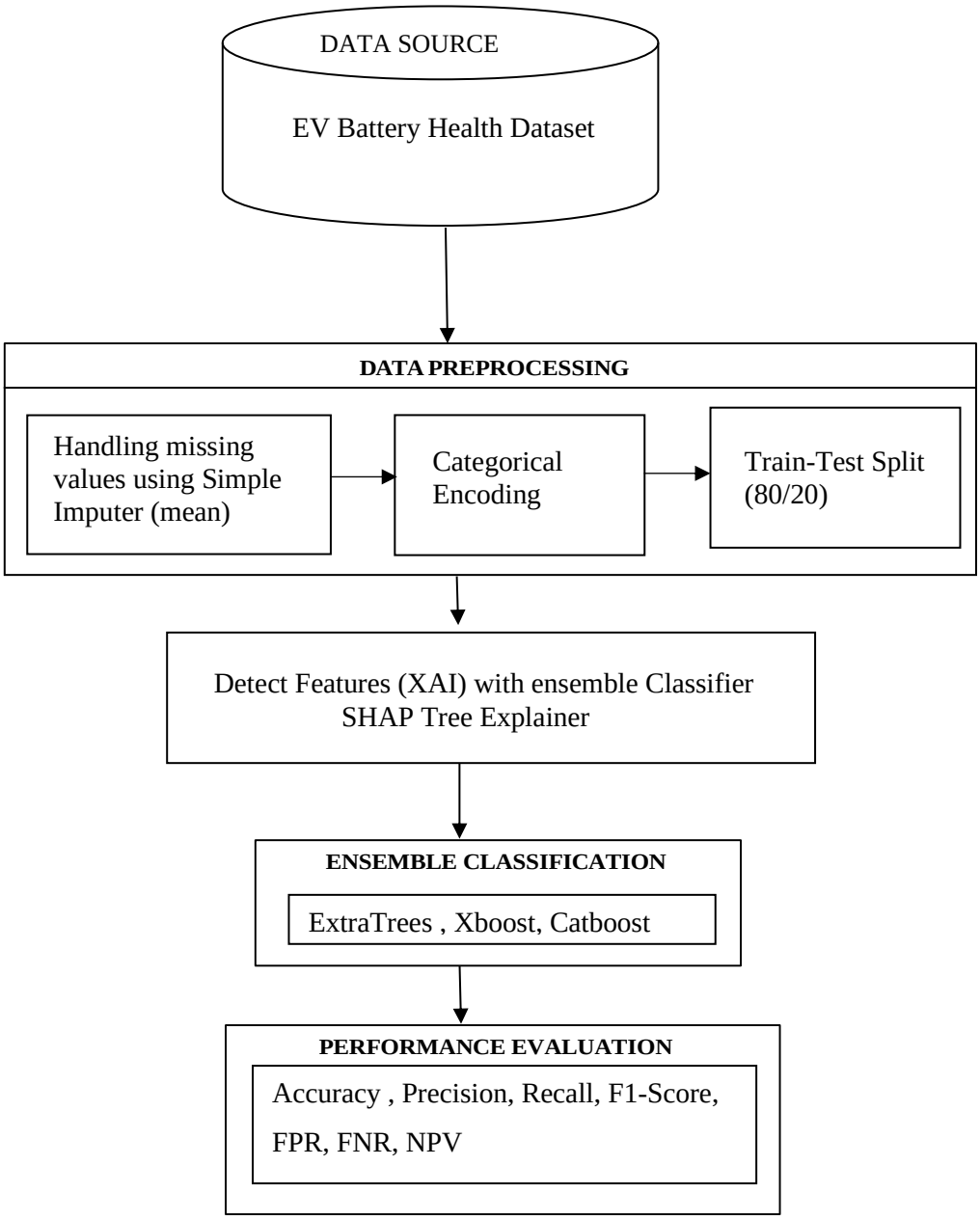



Figure 6.1: Flow Diagram

The given methodology involved converting the phases of data handling, model training, evaluation, and explanation to code that was executable. In this chapter, the tools, libraries, and procedure used in executing the EV battery health classification and XAI analysis are presented.

The project implementation as a whole was done primarily with the Python programming language. Python was used since it has an incredibly large set of libraries appropriate for data science, machine learning, and the explainability side. Key libraries utilized were: Pandas and NumPy for data loading, manipulation, and preprocessing, such as missing value handling with SimpleImputer. Scikit-learn to split the data (train_test_split) and to offer the implementation of the ExtraTreesClassifier and the core evaluation metrics such as accuracy_score and classification_report. XGBoost for the implementation of the XGBoost classification model. CatBoost for the implementation of the CatBoost classification model. Shap (SHapley Additive exPlanations) for applying the XAI techniques for explaining the trained models. Matplotlib and potentially Seaborn for plotting data features, model performance, and most critically, the SHAP results (summary plots, dependence plots, force plots).

The process began by loading the dataset. This initial process was preceded by the preprocessing processes described in Chapter 5, such as imputation of missing values and splitting the data into a training and test subsets. Subsequent to this, the selected models—XGBoost, CatBoost, and ExtraTreesClassifier—were created and separately trained on the training data. Once each model finished training, predictions were made on the unseen test dataset to make comparisons of their performance. Finally, the SHAP library was utilized to compute SHAP values corresponding to each model's prediction on the test data, thereby laying the foundation for the subsequent interpretation phase.

# CHAPTER 7

# PERFORMANCE ANALYSIS

# 7 PERFORMANCE ANALYSIS

Evaluating the performance of the trained classification models is essential to determine their effectiveness in accurately predicting EV battery health status. This chapter details the metrics used to assess the models—XGBoost, CatBoost, and ExtraTreesClassifier—and presents their performance results analyzed using the test dataset. Unlike regression tasks which might use metrics like Mean Squared Error (MSE), the discrete nature of battery health classes in this project necessitates the use of classification-specific evaluation criteria, providing a nuanced view beyond overall correctness.

## 7.1 Evaluation Metrics Used:

To thoroughly assess model performance in classifying EV battery health, a range of standard classification metrics were utilized. These include:

**Accuracy:** The proportion of correctly predicted instances (both positive and negative) out of the total number of instances.

Metrics derived from the Confusion Matrix: A table summarizing correct and incorrect predictions for each class. From the Confusion Matrix, we obtain more granular metrics:

**True Positive Rate (TPR) / Recall (Sensitivity) (Class 1):** The proportion of actual positive instances (Class 1) that were correctly identified.

**True Negative Rate (TNR) / Recall (Specificity) (Class 0):** The proportion of actual negative instances (Class 0) that were correctly identified.

**False Positive Rate (FPR):** The proportion of actual negative instances (Class 0) that were incorrectly predicted as positive (Class 1). (Calculated as 1 - TNR).

**False Negative Rate (FNR):** The proportion of actual positive instances (Class 1) that were incorrectly predicted as negative (Class 0). (Calculated as 1 - TPR).

**Precision (Class 0):** Of all instances predicted as Class 0,proportion that were actually Class 0.

**Negative Predictive Value (NPV):** Of all instances predicted as Class 0, the proportion that were actually Class 0.

**F1-score:** The harmonic mean of Precision and Recall, providing a balanced measure (not explicitly shown per class in the provided table, but standard).

**Classification Report:** A summary table presenting Precision, Recall, and F1-score for each class, along with support (number of instances per class) and overall averages (macro and weighted). These metrics collectively provide a comprehensive understanding of how well each model performs, highlighting not just overall accuracy but also performance characteristics for correctly identifying both positive (Class 1) and negative (Class 0) cases, and the types of errors made.

## 7.2 Results

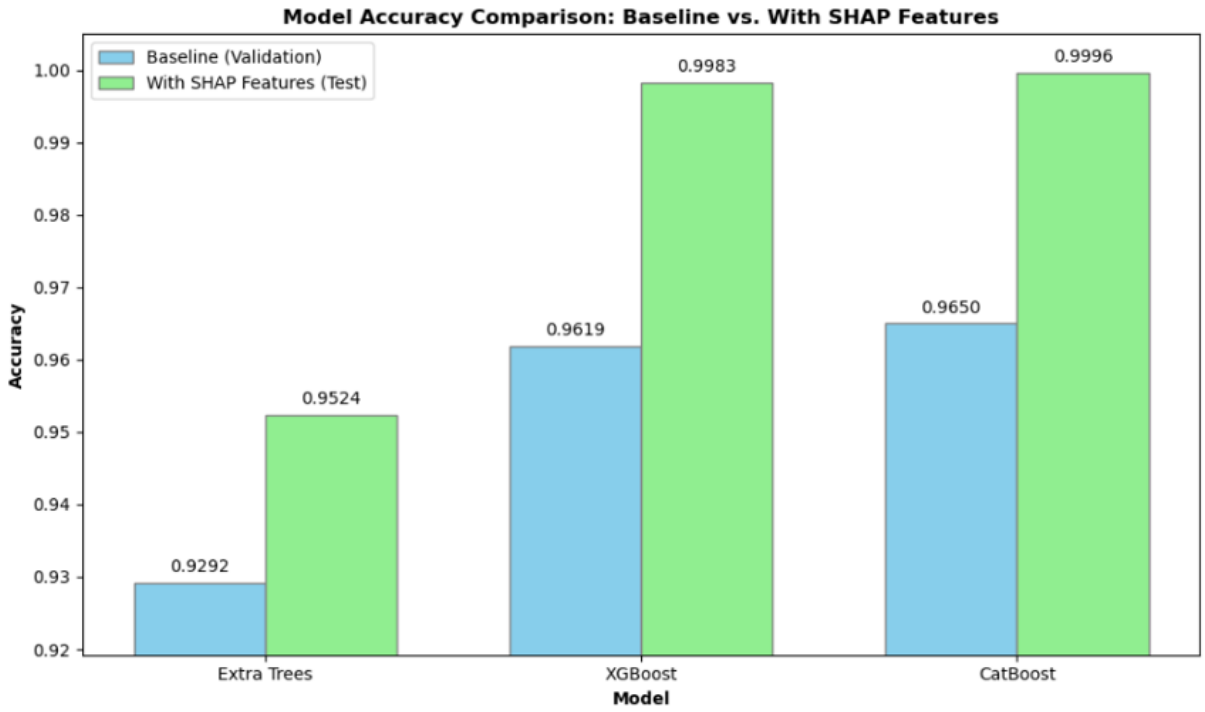



**Figure 7.2.1: Model Accuracy Comparison: Baseline (Validation) vs. With SHAP Features**

**Table 7.2.1: Performance Metrics for Baseline Models (Validation Set)**

| Model (Baseline Models) | Accuracy | Precision | Recall | FPR | FNR | NPV |
|---|---|---|---|---|---|---|
| CatBoost | 0.965 | 1 | 0.9208 | 0 | 0.0792 | 0.941 |
| XGBoost | 0.9619 | 1 | 0.9138 | 0 | 0.0862 | 0.936 |
| Extra Trees | 0.9292 | 0.9967 | 0.8426 | 0.0022 | 0.1574 | 0.8889 |

**Table 7.2.2: Performance Metrics for Models with XAI selected features on the Test Set**

| Model (with selected features) | Accuracy | Precision | Recall | FPR | FNR | NPV |
|---|---|---|---|---|---|---|
| XGBoost Model | 0.9983 | 0.9983 | 0.9979 | 0.0014 | 0.0021 | 0.9983 |
| Extra Trees Model | 0.9524 | 0.9999 | 0.8935 | 0.0001 | 0.1065 | 0.9208 |
| CatBoost Model | 0.9996 | 0.999 | 1 | 0.0008 | 0 | 1 |

# CHAPTER 8

# CONCLUSION

# 9.CONCLUSION

Our project set out to tackle a critical challenge in Electric Vehicle (EV) technology: reliably classifying battery health while also understanding why the predictions are made. This involved integrating Machine Learning (ML) with Explainable AI (XAI). For the core predictive task, we focused on three high-performing classification models known for their effectiveness: XGBoost, CatBoost, and ExtraTreesClassifier.

Assessing how well these models performed was a key step, detailed in Chapter 8. We relied on standard classification metrics drawn from the confusion matrix—measures like Accuracy, Precision, Recall, and the F1-score. Evaluating them against the unseen test dataset revealed truly exceptional results across the board. CatBoost led the way with an accuracy of 99.96%, very closely matched by XGBoost at 99.83%. The Extra Trees Classifier also performed strongly, achieving 95.24% accuracy. These figures clearly underscore the robust predictive power inherent in our chosen models for classifying battery health states.

However, achieving high accuracy was only half the battle. A crucial part of this project involved integrating the SHapley Additive exPlanations (SHAP) framework. Applying SHAP to our trained models gave us a powerful lens into their decision-making. We could quantify precisely how each feature contributed to an individual prediction and identify which factors were globally most influential. This interpretability provided invaluable insights, helping us understand the empirical drivers behind the battery health classifications and shedding light on the complex process of battery degradation itself, as reflected in our data.

In conclusion, this project effectively demonstrates the significant benefits of pairing powerful, accurate Machine Learning classifiers with XAI techniques such as SHAP. Our work shows that building truly interpretable models isn't just possible; it's vital for cultivating trust and extracting deeper, actionable insights, particularly in critical domains like EV battery health management. The outcome is a step towards more reliable and user-friendly AI systems for electric vehicles.

# CHAPTER 9

# FUTURE ENHANCEMENT

## 10. FUTURE ENHANCEMENT

Based on the findings of this project, the following critical areas have been established for additional development and study with the intent of further refining the system for electric vehicle battery health analysis: Future research opportunity includes the integration of real-time operational data streams from electric vehicles to facilitate dynamic health estimates and early anomaly detection. Experimental verification and extension to other battery chemistries and a broader range of operating conditions is needed to enhance generalizability and real-world applicability. Other or more advanced machine learning and deep learning architectures suited to battery data features could be investigated through research. Investigation of other XAI methods other than SHAP and comparative studies could provide additional insights into model interpretations. Development of interactive visualization tools or dashboards for presenting predictions and explanations would improve usability by engineers and maintenance staff. Lastly, integration of the predictive and explanatory capability directly into Battery Management Systems (BMS) provides an opportunity for advanced predictive maintenance and potential prescriptive advice.

# REFERENCES

# REFERENCES

[1] Han, Xuewei, Huimei Yuan, and Lifeng Wu. "HyPELS: enhancing li-ion battery remaining useful life prediction with hybrid perspective ensemble learning strategy." *Engineering Research Express* 6, no. 4 (2024): 045218.

[2] Han, Dongxu, Nan Zhou, and Zeyu Chen. "Enhancing State of Health Prediction Accuracy in Lithium-Ion Batteries through a Simplified Health Indicator Method." *Batteries* 10, no. 10 (2024): 342.

[3] Huang, Guangzhong, Wenping Lei, Xinmin Dong, Dongliang Zou, Shijin Chen, and Xing Dong. "Stage-Based Remaining Useful Life Prediction for Bearings Using GNN and Correlation-Driven Feature Extraction." *Machines* 13, no. 1 (2025): 43.

[4] Wang, Sijing, Ruoyu Zhou, Yijia Ren, Meiyuan Jiao, Honglai Liu, and Cheng Lian. "Advanced data-driven techniques in AI for predicting lithium-ion battery remaining useful life: a comprehensive review." *Green Chemical Engineering* (2024).

[5] Murgai, S., Bhagwat, H., Dandekar, R. A., Dandekar, R., & Panat, S. (2024). A Scientific Machine Learning Approach for Predicting and Forecasting Battery Degradation in Electric Vehicles. *arXiv preprint arXiv:2410.14347*.

[6] Sharma, Riya, et al. "State of Health Estimation for Li-Ion Battery Using Machine Learning." *2023 IEEE International Conference on Metrology for eXtended Reality, Artificial Intelligence and Neural Engineering (MetroXRAINE)*. IEEE, 2023.

[7] Devi, B., and V. Suresh Kumar. "Machine Learning-Based Approach for Predicting Battery Remaining Useful Life (RUL): A Data-Driven Methodology." *2023 International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS)*. IEEE, 2023.

[8] Nair, Rahul Anil, Dev Karan Suresh, and D. Rajeswari. "Battery Health Prediction Using Deep Hybrid Learning." *2023 3rd International Conference on Pervasive Computing and Social Networking (ICPCSN)*. IEEE, 2023.

[9] Niraula, Aagya, and Jai Govind Singh. "Deep Learning-Based Approach for State-of-Health Estimation of Lithium-Ion Battery in the Electric Vehicles." *2023 International Conference on Power, Instrumentation, Energy and Control (PIECON)*. IEEE, 2023.

[10] Nair, Rahul Anil, Dev Karan Suresh, and D. Rajeswari. "Battery Health Prediction Using Deep Hybrid Learning." *2023 3rd International Conference on Pervasive Computing and Social Networking (ICPCSN)*. IEEE, 2023.

# APPENDIX

## Source Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
import xgboost as xgb
from sklearn.metrics import accuracy_score, classification_report
import shap
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
GradientBoostingClassifier, BaggingClassifier, ExtraTreesClassifier, VotingClassifier,
StackingClassifier
from sklearn.linear_model import LogisticRegression
from catboost import CatBoostClassifier
from sklearn.impute import SimpleImputer
import numpy as np  # Import NumPy
import seaborn as sns # For enhanced visualizations

# 1. Load the processed data (1 lac rows)
df= pd.read_csv("C:\\Users\\spakt\\Downloads\\evbatterydataset.csv")

# --- Exploratory Data Analysis (EDA) ---
print("\n--- Exploratory Data Analysis ---")

# Display the first few rows of the dataframe
print("\nFirst 5 rows of the dataframe:")
print(df.head())

# Get a concise summary of the dataframe, including data types and non-null values
print("\nInformation about the dataframe:")
df.info()

# Display descriptive statistics for numerical columns
print("\nDescriptive statistics for numerical columns:")
print(df.describe())

# Check the distribution of the target variable
print("\nDistribution of the target variable ('Battery_Class'):")
print(df['Battery_Class'].value_counts())
sns.countplot(x='Battery_Class', data=df)
plt.title('Distribution of Battery Class')
```

```
plt.show()

# Visualize the distribution of numerical features using histograms
print("\nHistograms of numerical features:")
numerical_cols = df.select_dtypes(include=np.number).columns.tolist()
if 'Battery_Class' in numerical_cols:
    numerical_cols.remove('Battery_Class')
if 'Battery_Health' in numerical_cols:
    numerical_cols.remove('Battery_Health') # Removing Battery_Health as it might have been
intended to be excluded later

for col in numerical_cols:
    plt.figure(figsize=(8, 6))
    sns.histplot(df[col], kde=True)
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()

# Visualize relationships between numerical features and the target variable using box plots
print("\nBox plots of numerical features vs. Battery Class:")
for col in numerical_cols:
    plt.figure(figsize=(8, 6))
    sns.boxplot(x='Battery_Class', y=col, data=df)
    plt.title(f'{col} vs. Battery Class')
    plt.xlabel('Battery Class')
    plt.ylabel(col)
    plt.show()

# Check for missing values
print("\nMissing values in the dataframe:")
print(df.isnull().sum())

# Check for correlations between numerical features using a heatmap
print("\nCorrelation heatmap of numerical features:")
correlation_matrix = df[numerical_cols + ['Battery_Health']].corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Numerical Features (including Battery_Health)')
plt.show()
```

```
# 2. Define the target variable
target_variable = "Battery_Class"

# 3. Separate features (X) and target (y) - KEEPING 'Battery_Health'
if target_variable in df.columns and 'Battery_Health' in df.columns:
    y = df[target_variable]
    X = df.drop(target_variable, axis=1)
elif target_variable in df.columns:
    print("Warning: 'Battery_Health' not found in the dataframe. Please ensure it's present.")
    exit()
else:
    print(f"Error: Target variable '{target_variable}' not found in the CSV file.")
    exit()

# --- Handle Missing Values using SimpleImputer ---
imputer = SimpleImputer(strategy='mean')
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

# --- Check missing values AFTER imputation ---
print("\n--- Missing values AFTER imputation ---")
print(X_imputed.isnull().sum())

# --- Split the data into training and testing sets ---
# Note: This split was missing in the provided code block before model training
# Assuming X_imputed and y are the processed features and target
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y, test_size=0.2, random_state=42)


# --- Existing XGBoost Model ---
print("\n--- XGBoost Model ---")
model_xgb = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss',
random_state=42)
# Ensure X_train and y_train are defined from a split BEFORE fitting
model_xgb.fit(X_train, y_train)
y_pred_xgb = model_xgb.predict(X_test)
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
print(f"Accuracy on Test Set: {accuracy_xgb:.4f}")
print("Classification Report:")
print(classification_report(y_test, y_pred_xgb))
```

```
print("\n--- Extra Trees Model ---")
from sklearn.ensemble import ExtraTreesClassifier # Ensure ExtraTreesClassifier is imported if not already
model_et_tuned = ExtraTreesClassifier(random_state=42) # Using default parameters for simplicity
# Note: X_train_new, y_train_new, and X_val, y_val are used here, but not defined in the previous code snippet.
# Assuming these are intended to be the standard X_train, y_train, X_test, y_test split
# or a further split for validation. Replacing with standard split for now.
# If you need a validation set, you'd split X_train, y_train again.
# model_et_tuned.fit(X_train_new, y_train_new)
# y_pred_et_tuned = model_et_tuned.predict(X_val)
# accuracy_et_tuned = accuracy_score(y_val, y_pred_et_tuned)
# print(f"Validation Accuracy (Tuned Extra Trees): {accuracy_et_tuned:.4f}")
# print("Validation Classification Report (Tuned Extra Trees):")
# print(classification_report(y_val, y_pred_et_tuned))

# Using standard X_train, X_test split for training and testing
model_et = ExtraTreesClassifier(random_state=42)
model_et.fit(X_train, y_train)
y_pred_et = model_et.predict(X_test)
accuracy_et = accuracy_score(y_test, y_pred_et)
print(f"Accuracy on Test Set (Extra Trees): {accuracy_et:.4f}")
print("Classification Report (Extra Trees):")
print(classification_report(y_test, y_pred_et))



print("\n--- CatBoost Model ---")
model_cat_tuned = CatBoostClassifier(
    verbose=0,
    random_state=42,
    l2_leaf_reg=20,      # Further Increased L2 regularization
    depth=3,             # Further Reduced tree depth
    learning_rate=0.005, # Further Reduced learning rate
    iterations=25,       # Significantly Reduced number of boosting iterations
    colsample_bylevel=0.7 # Correct parameter for feature subsampling in CatBoost
)
# Note: X_train_new, y_train_new, and X_val, y_val are used here, but not defined in the
```

```
previous code snippet.
# Assuming these are intended to be the standard X_train, y_train, X_test, y_test split
# or a further split for validation. Replacing with standard split for now.
# model_cat_tuned.fit(X_train_new, y_train_new, eval_set=[(X_val, y_val)],
early_stopping_rounds=10)
# y_pred_cat_tuned = model_cat_tuned.predict(X_val)
# accuracy_cat_tuned = accuracy_score(y_val, y_pred_cat_tuned)
# print(f"Validation Accuracy: {accuracy_cat_tuned:.4f}")
# print("Validation Classification Report :")
# print(classification_report(y_val, y_pred_cat_tuned))

# Using standard X_train, X_test split for training and testing
model_cat = CatBoostClassifier(
    verbose=0,
    random_state=42
) # Simplified parameters for example, you can use your tuned ones if X_train/X_test is the final
test set
model_cat.fit(X_train, y_train)
y_pred_cat = model_cat.predict(X_test)
accuracy_cat = accuracy_score(y_test, y_pred_cat)
print(f"Accuracy on Test Set (CatBoost): {accuracy_cat:.4f}")
print("Classification Report (CatBoost):")
print(classification_report(y_test, y_pred_cat))



# --- SHAP Analysis (Visualizing features other than Battery_Health) ---
print("\n--- SHAP Analysis (for XGBoost as an example) ---")
# Initialize the SHAP explainer for the XGBoost model
explainer = shap.TreeExplainer(model_xgb)

# Calculate SHAP values on the test set
shap_values = explainer.shap_values(X_test)

# Check the shape of shap_values
print(f"The shape of shap_values is: {np.array(shap_values).shape}")

if isinstance(shap_values, list):
    print(f"Shape of SHAP values for class 0: {shap_values[0].shape}")
    print(f"Shape of SHAP values for class 1: {shap_values[1].shape}")
```

```
# Get feature names, excluding 'Battery_Health' for visualization
feature_names = [col for col in X_train.columns if col != 'Battery_Health']
feature_names_all = X_train.columns # Keep all feature names for force plot

# Get indices of features excluding 'Battery_Health'
feature_indices_to_visualize = [i for i, col in enumerate(X_train.columns) if col !=
'Battery_Health']

# Visualize SHAP results

# 1. Summary Plot (excluding 'Battery_Health')
plt.figure(figsize=(12, 8))
# Ensure X_test used with SHAP values is the same size and feature order as X_train used for
explainer
shap.summary_plot(shap_values[:, feature_indices_to_visualize], X_test.iloc[:,
feature_indices_to_visualize], feature_names=feature_names)
plt.title("SHAP Summary Plot (excluding Battery_Health)")
plt.tight_layout()
plt.show()


# --- Feature Selection based on SHAP Importance ---
print("\n--- Feature Selection based on SHAP Importance ---")

# Initialize SHAP explainer
explainer = shap.TreeExplainer(model_xgb)
shap_values = explainer.shap_values(X_train) # Use training data to determine feature
importance

# Get feature importances based on mean absolute SHAP values
feature_importances = np.abs(shap_values).mean(axis=0)

# If shap_values is a list (for multi-class), take the mean across all classes
if isinstance(feature_importances, np.ndarray) and feature_importances.ndim == 2:
    feature_importances = feature_importances.mean(axis=0)

feature_names_all = list(X_train.columns)
importance_df = pd.DataFrame({'Feature': feature_names_all, 'Importance':
feature_importances})
importance_df = importance_df.sort_values(by='Importance',
```

```
ascending=False).reset_index(drop=True)

# Select the top N important features
top_n = 20 # You can adjust this number
important_features_to_keep = importance_df['Feature'].head(top_n).tolist()

# Filter the features in your training and testing data
X_train_important = X_train[[col for col in important_features_to_keep if col in
X_train.columns]]
X_test_important = X_test[[col for col in important_features_to_keep if col in X_test.columns]]

print(f"\nNumber of features before selection: {X_train.shape[1]}")
print(f"Number of important features selected: {X_train_important.shape[1]}")
print(f"Selected important features (top {top_n}): {X_train_important.columns.tolist()}")

# Train a new XGBoost model using only the important features
model_xgb_important = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss')
model_xgb_important.fit(X_train_important, y_train)

# Make predictions on the test set
y_pred_important = model_xgb_important.predict(X_test_important)

# Evaluate the model
accuracy_important = accuracy_score(y_test, y_pred_important)
print(f"\nAccuracy after selecting important features: {accuracy_important:.4f}")
print("Classification Report after selecting important features:")
print(classification_report(y_test, y_pred_important))

# --- SHAP Analysis on the Model with Important Features (excluding Battery_Health) ---
print("\n--- SHAP Analysis on the Model with Important Features (excluding Battery_Health) ---
")

explainer_important = shap.TreeExplainer(model_xgb_important)
shap_values_important_all = explainer_important.shap_values(X_test_important)
feature_names_important_all = list(X_test_important.columns)

# Filter out 'Battery_Health' if it's in the selected features
if 'Battery_Health' in feature_names_important_all:
    battery_health_index = feature_names_important_all.index('Battery_Health')
    if isinstance(shap_values_important_all, list):
```

```
        shap_values_important = [vals[:, [i for i in range(vals.shape[1]) if i !=
battery_health_index]] for vals in shap_values_important_all]
    else:
        shap_values_important = shap_values_important_all[:, [i for i in
range(shap_values_important_all.shape[1]) if i != battery_health_index]]
    feature_names_important = [f for f in feature_names_important_all if f != 'Battery_Health']
else:
    shap_values_important = shap_values_important_all
    feature_names_important = feature_names_important_all

plt.figure(figsize=(10, 8))
shap.summary_plot(shap_values_important, pd.DataFrame(X_test_important,
columns=feature_names_important_all)[[col for col in feature_names_important_all if col !=
'Battery_Health']], feature_names=feature_names_important)
plt.title(f"SHAP Summary Plot with Top {top_n} Important Features (excluding
Battery_Health)")
plt.tight_layout()
plt.show()

# --- CatBoost Model (with selected features) ---
print("\n--- CatBoost Model (with selected features) ---")
model_cat_important = CatBoostClassifier(verbose=0, random_state=42) # Using default
parameters
# Ensure X_train_important and y_train are defined from a split BEFORE fitting
model_cat_important.fit(X_train_important, y_train)
y_pred_cat_important = model_cat_important.predict(X_test_important)
accuracy_cat_important = accuracy_score(y_test, y_pred_cat_important)
print(f"Accuracy: {accuracy_cat_important:.4f}")
print(classification_report(y_test, y_pred_cat_important))

print("\n--- Extra Trees Model (with selected features) ---")
model_et_important = ExtraTreesClassifier(random_state=42) # Using default parameters
# Ensure X_train_important and y_train are defined from a split BEFORE fitting
model_et_important.fit(X_train_important, y_train)
y_pred_et_important = model_et_important.predict(X_test_important)
accuracy_et_important = accuracy_score(y_test, y_pred_et_important)
print(f"Accuracy: {accuracy_et_important:.4f}")
print(classification_report(y_test, y_pred_et_important)) import matplotlib.pyplot as plt
import numpy as np
```

```
# Model names
model_names = ['Extra Trees', 'XGBoost', 'CatBoost']

# Accuracies
baseline_accuracies = np.array([0.9292, 0.9619, 0.9650]) # Validation accuracies without SHAP selection
shap_features_accuracies = np.array([0.9524, 0.9983, 0.9996]) # Test accuracies with SHAP features

# Set the width of the bars
bar_width = 0.35

# Set the position of the bars on the x-axis
r1 = np.arange(len(model_names))
r2 = [x + bar_width for x in r1]

# Create the grouped bar chart
plt.figure(figsize=(10, 6))
plt.bar(r1, baseline_accuracies, color='skyblue', width=bar_width, edgecolor='grey', label='Baseline (Validation)')
plt.bar(r2, shap_features_accuracies, color='lightgreen', width=bar_width, edgecolor='grey', label='With SHAP Features (Test)')

# Add labels and title
plt.xlabel('Model', fontweight='bold')
plt.ylabel('Accuracy', fontweight='bold')
plt.title('Model Accuracy Comparison: Baseline vs. With SHAP Features', fontweight='bold')
plt.xticks([r + bar_width/2 for r in range(len(model_names))], model_names)

# Add accuracy values on top of the bars
for i in range(len(model_names)):
    plt.text(r1[i], baseline_accuracies[i] + 0.001, f'{baseline_accuracies[i]:.4f}', ha='center', va='bottom')
    plt.text(r2[i], shap_features_accuracies[i] + 0.001, f'{shap_features_accuracies[i]:.4f}', ha='center', va='bottom')

# Set y-axis limits for better visualization of differences
# Increased the upper limit slightly
plt.ylim(min(baseline_accuracies.min(), shap_features_accuracies.min()) - 0.01, 1.005) # Adjusted upper limit
```

```
# Add a legend
plt.legend()

# Show the plot
plt.tight_layout()
plt.show()
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix, recall_score, precision_score
# Assuming y_pred_cat_tuned, y_pred_xgb_tuned, y_pred_et_tuned, and y_val are available
# from the previous code execution where these models were evaluated on X_val.

print("\n--- Detailed Performance Metrics (Validation Set) ---")

# List of model prediction results and their names
model_predictions = [
    (y_pred_cat_tuned, 'CatBoost'),
    (y_pred_xgb_tuned, 'XGBoost'),
    (y_pred_et_tuned, 'Extra Trees')
]

for y_pred, name in model_predictions:
    print(f"\n--- Metrics for {name} ---")

    # Calculate Accuracy
    accuracy = accuracy_score(y_val, y_pred)
    print(f"Accuracy: {accuracy:.4f}")

    # Calculate Confusion Matrix
    cm = confusion_matrix(y_val, y_pred)
    # Extract components: cm[true_label, predicted_label]
    # Use the .ravel() method which works reliably for 2x2 matrices
    if cm.size == 4:
        tn, fp, fn, tp = cm.ravel()
    else:
        # Handle cases where only one class was predicted (unlikely with high accuracy)
        # But for safety, extract based on shape
        if cm.shape == (2, 2):
            tn, fp, fn, tp = cm[0,0], cm[0,1], cm[1,0], cm[1,1]
```

```
        else:
            print("Warning: Confusion matrix is not 2x2. Cannot calculate all metrics.")
            print(cm)
            continue # Skip to the next model


    # Calculate other metrics from Confusion Matrix components

    # Precision for Class 0 (Negative Predictive Value for Predicted Class 0)
    precision_0 = precision_score(y_val, y_pred, pos_label=0)
    print(f"Precision (Class 0 / Pred Negative PPV): {precision_0:.4f}")

    # Recall for Class 0 (True Negative Rate - TNR)
    recall_0 = recall_score(y_val, y_pred, pos_label=0)
    print(f"Recall (Class 0 / TNR): {recall_0:.4f}")

    # Precision for Class 1 (Positive Predictive Value - PPV)
    precision_1 = precision_score(y_val, y_pred, pos_label=1)
    print(f"Precision (Class 1 / PPV): {precision_1:.4f}")

    # Recall for Class 1 (Sensitivity / True Positive Rate - TPR)
    recall_1 = recall_score(y_val, y_pred, pos_label=1)
    print(f"Recall (Class 1 / Sensitivity / TPR): {recall_1:.4f}")


    # False Positive Rate (FPR) - Rate of predicting positive when actual is negative
    # FPR = FP / (FP + TN)
    fpr = fp / (fp + tn) if (fp + tn) > 0 else 0
    print(f"False Positive Rate (FPR): {fpr:.4f}")

    # False Negative Rate (FNR) - Rate of predicting negative when actual is positive
    # FNR = FN / (FN + TP)
    fnr = fn / (fn + tp) if (fn + tp) > 0 else 0
    print(f"False Negative Rate (FNR): {fnr:.4f}")

    # Negative Predictive Value (NPV) - Proportion of correctly predicted negatives out of all
predicted negatives
    # NPV = TN / (TN + FN)
    npv = tn / (tn + fn) if (tn + fn) > 0 else 0
    print(f"Negative Predictive Value (NPV): {npv:.4f}")
```

```
    print("-" * 30) # Separator for clarity
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# --- Function to print detailed metrics ---
def print_detailed_metrics(y_true, y_pred, model_name):
    print(f"\n--- Metrics for {model_name} ---")
    accuracy = accuracy_score(y_true, y_pred)

    recall_0 = report['0'].get('recall', 0.0) # TNR
    precision_1 = report['1'].get('precision', 0.0) # PPV
    recall_1 = report['1'].get('recall', 0.0) # Sensitivity / TPR

    # Calculate additional metrics
    # FPR = 1 - TNR (Recall for Class 0)
    fpr = 1 - recall_0
    # FNR = 1 - TPR (Recall for Class 1)
    fnr = 1 - recall_1
    # NPV = Precision (Class 0)
    npv = precision_0

    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision (Class 0 / Pred Negative PPV): {precision_0:.4f}")
    print(f"Recall (Class 0 / TNR): {recall_0:.4f}")
    print(f"Precision (Class 1 / PPV): {precision_1:.4f}")
    print(f"Recall (Class 1 / Sensitivity / TPR): {recall_1:.4f}")
    print(f"False Positive Rate (FPR): {fpr:.4f}")
    print(f"False Negative Rate (FNR): {fnr:.4f}")
    print(f"Negative Predictive Value (NPV): {npv:.4f}")

# --- Apply Metrics to Each Trained Model ---

# Note: The model training code from the previous step is required
# to generate the 'y_pred' variables used below.
# Ensure the following models have been trained using X_train_important
# and predictions made on X_test_important:
# model_rf_important, model_ada_important, model_gb_important,
# model_bag_important, model_stacking_important, model_voting_important,
```

```
# model_cat_important, model_logit_important, model_et_important

# Example usage (assuming y_pred variables are available):

print_detailed_metrics(y_test, y_pred_rf_important, "Random Forest Model (with selected features)")
print_detailed_metrics(y_test, y_pred_ada_important, "AdaBoost Model (with selected features)")
print_detailed_metrics(y_test, y_pred_gb_important, "Gradient Boosting Model (with selected features)")
print_detailed_metrics(y_test, y_pred_bag_important, "Bagging Model (with selected features)")
print_detailed_metrics(y_test, y_pred_stacking_important, "Stacking Classifier (with selected features)")
print_detailed_metrics(y_test, y_pred_voting_important, "Voting Classifier (with selected features)")
print_detailed_metrics(y_test, y_pred_cat_important, "CatBoost Model (with selected features)")
print_detailed_metrics(y_test, y_pred_logit_important, "LogitBoost Model (with selected features)")
print_detailed_metrics(y_test, y_pred_et_important, "Extra Trees Model (with selected features)")
print_detailed_metrics(y_test, y_pred_important, "XGBoost Model (with selected features)")
```