Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Danni Zhang

AI-Based Surrogate Modeling Technique In Hydrogen Release Prediction

Master's thesis in Reliability, Availability, Maintainability, and Safety (RAMS)
Supervisor: Shen Yin
June 2023

NTNU
Norwegian University of
Science and Technology

RAMS
Reliability, Availability,
Maintainability, and Safety

Danni Zhang

AI-Based Surrogate Modeling Technique In Hydrogen Release Prediction

Master's thesis in Reliability, Availability, Maintainability, and Safety (RAMS)
Supervisor: Shen Yin
June 2023

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

RAMS
Reliability, Availability,
Maintainability, and Safety

AI-Based Surrogate Modeling Technique In Hydrogen Release Prediction

Danni Zhang

JUNE 2023

MASTER THESIS
Department of Mechanical and Industrial Engineering
Norwegian University of Science and Technology

Supervisor: Shen Yin

Preface

This study uses two machine learning algorithms to predict the hydrogen spill problem. It was carried out during the spring semester of 2023. This master thesis is written for master students with similar interests in AI-based surrogate modeling techniques.

Trondheim, 2023-06-11

Danni Zhang

Acknowledgment

I would like to express my gratitude to my supervisor, Shen Yin, for his patience and assistance throughout this thesis. In addition, I would like to extend my appreciation to Ph.D. student Muhammad Gibran Alfarizi and Federica Ferrari from the RAMS group for their help. Lastly, I would like to thank my classmates, friends, and family for their continuous encouragement and support throughout this journey.

Danni.Zhang

Remark:

Given the opportunity, the RAMS group would like to acknowledge the contributions of Professor Emeritus Marvin Rausand for his efforts in preparing this template. We would also like to express our gratitude to Professor Mary Ann Lundteigen and Professor Jørn Vatn for their valuable suggestions and proposed modifications. Their input has greatly improved the content of this document. Furthermore, we would like to acknowledge Associate Professor Anita Romsdal for providing important material that has been incorporated into this revised version. Her contributions have significantly enhanced the overall quality of this work.

Abstract

Hydrogen is a clean substitute energy for traditional carbon-fuel energy. Since it is the lightest element, it has low density. Liquid hydrogen becomes a good way of storing and transporting hydrogen. But since its low boiling point (-253°C). When it spills it can cause the surrounding air to liquefy or even solidify. Hydrogen has a wide flammability range(4-75%vol in the air) and low ignition energy (0.019 mJ),so it is very dangerous when it releases into the air.In a previous study, Random Forests perform good metrics, In this paper we are trying to use different machine learning models, for example, XGBoost and deep learning Artificial Neural Networks for predicting the hydrogen release according to the data we received.

Contents

Chapter 1

Introduction

1.1 Background

Non-renewable fossil fuels has led to a serious environmental crisis, causing widespread pollution and consuming natural resources at an alarming rate (Debe, 2012). To mitigate the negative impact of these fuels, there has been an increase in research on identifying renewable sources of energy. Hydrogen has emerged as a prominent contender for the future of the energy industry among the various alternatives being investigated, thanks to its distinctive attributes like an inexhaustible supply and exceptional energy efficiency. (Zheng et al., 2012a).

Hydrogen is a lightweight substance, characterized by a range (4-75%vol in the air) and an low ignition energy (0.019 mJ) (Aaneby and Hafskjold, 2021). With its potential to provide a cleaner and more sustainable source of energy, hydrogen has the capacity to bring about a revolutionary transformation in the energy sector.(Hu et al., 2015). However, the transportation and storage of hydrogen can pose significant challenges due to its physical properties. This is where liquid hydrogen (LH2) comes in, as it transports and store hydrogen (Mazzotti and Bidini, 2020).

Despite the benefits of LH2, it is important to know that there are several safety hazards associates with its handling and use. The most significant risk is the extremely low boiling point of LH2, which stands at -253°C (O'Connor, 2019; Huang et al., 2019). At such low temperatures, the condensation or solidification of air (boiling point -192°C, freezing point

-215.2°C) can lead to a potentially dangerous situation. Furthermore, if a flammable mixture of LH2 and solid oxygen ignites, it can result in an explosion Wang and Yuan (2014).

To gain a deeper understanding of the potential risks associated with large-scale LH2 releases, a series of experiments were recently conducted in both an outdoor facility and a closed room (Hong et al., 2021). These tests simulated spills that could occur during bunkering operations. The results of these tests were analyzed using advanced machine learning approaches to gain new insights into the behavior of LH2 under various conditions (Li et al., 2020). The aim of this research was to develop new safety protocols and guidelines for the handling and use of LH2, with the goal of ensuring the safe and sustainable use of hydrogen as a clean energy source (Zheng et al., 2012b).

In summary, hydrogen presents itself as a promising contender for the future of the energy industry, providing a cleaner and more sustainable alternative when compared to non-renewable fossil fuels. However, it is important to recognize the potential risks associated with its transportation and use, especially with regarding to LH2. By conducting rigorous testing and analysis, we can ensure the safe and sustainable use of hydrogen as a viable source of energy for generations to come.

1.2 Objectives

The objectives are

1. Investigate different algorithms of machine learning. The primary goals involve conducting a comprehensive exploration of various machine learning algorithms, encompassing supervised, unsupervised, and reinforcement learning.

2. Deep understanding of the use case. Moreover, the project strives to enhance both the metrics and performance of the system. This involves identifying the existing metrics and analyzing their limitations. The project will then propose new metrics that are more suitable for the use case and implement them using Python. The new metrics will be evaluated and compared against the existing ones to demonstrate their effectiveness.

3. Use Python to build surrogate models. To achieve these objectives, the project will follow a structured approach that involves data collection and preprocessing, model development, and evaluation. The data collection process will involve acquiring relevant data from various sources and preparing it for analysis. The model development stage will involve selecting appropriate algorithms and building models using Python. The evaluation process will involve comparing the performance of different models using the identified metrics.

4. Improve the metrics based on the current study. Overall, The project's objective is to make a valuable contribution to the field of machine learning by developing a versatile strategy applicable to a wide range of use cases. The project will also contribute to the development of more effective metrics for evaluating the performance of machine learning models.

1.3 Approach

The following is a list of the approaches adopted to address the problem and fulfill the objectives and tasks at hand.

1. Literature review about different algorithms of Machine Learning (King, 2010; Pedregosa et al., 2011). As part of the literature review, the project would also have researched different algorithms of machine learning, which are the foundational techniques used to build AI models. This would have included a broad range of techniques, such as decision trees, random forests (Breiman, 2001), and deep neural networks, among others. Then would have evaluated the strengths and weaknesses of each algorithm and identified the most appropriate techniques to use for the specific use case.

2. Deep understanding of python code. To build effective AI models, it's essential to have a deep understanding of the specific problem that needs to be solved. Additionally, Would have developed a deep understanding of the python code used to build the models, as this would be critical to understanding how the models work and making any necessary modifications or improvements.

3. Deep understanding Extreme Gradient Boosting(XGBoost) (Tian et al., 2016). XGBoost is a powerful algorithm for predictive modeling tasks. Then would have developed a deep understanding of how XGBoost works, including its strengths and weaknesses and how it can be applied to the specific use case. This would have involved exploring the various hyperparameters and tuning them to achieve optimal performance.

4. Deep understanding Neural Networks(Goodfellow et al., 2016; Hinton et al., 2012; LeCun et al., 2015): Artificial neural networks are a class of deep learning algorithms that are particularly effective for tasks. Then, an in-depth comprehension of neural networks would have been cultivated, encompassing their workings, such as different layers and architectures, along with the optimization techniques tailored to the specific use case.

1.4 Outline

Here is an overview of how the report is organized.

- Chapter 1. Introduction: This chapter offers a comprehensive overview of the problem addressed by the report, along with the defined objectives and tasks that it aims to achieve. It also provides some background information on the context of the problem and explains why it is important to solve it. Finally, this chapter outlines the structure of the report and provides a brief overview of what the reader can expect in each chapter.

- Chapter 2. Theoretical Background: This chapter provides a detailed explanation of the theoretical concepts and principles of the project. This encompasses a review of pertinent literature, essential definitions and concepts, and a comprehensive explanation of the methodologies and techniques employed to address the problem.

- Chapter 3. Experiment Design: This chapter provides a detailed explanation of the experiment design, including the research methodology, data sources, and experimental setup. This may include an overview of the data collection and processing methods.

Additionally, this chapter may provide details on the specific machine learning models used, including their parameters and hyperparameters.

- Chapter 4. Results and Discussion: This chapter shows the results of the experiments.This could involve an examination of the precision and efficiency of the machine learning models, along with an investigation into the fundamental elements that influence their accomplishments or shortcomings.
- Chapter 5. Future Work: In this chapter, potential avenues for future research are presented, building upon the conclusions drawn from the current study. These possibilities encompass deliberations on potential enhancements to the experimental design or machine learning models, as well as potential expansions or practical implementations of the research.
- Appendix Acronyms: This appendix offers a comprehensive compilation of acronyms utilized throughout the report, accompanied by their corresponding full definitions.
- Bibliography: This section provides a list of all the references cited in the report, organized according to the citation style specified by the authoring institution or publisher. This may include academic papers, industry reports, and other relevant sources.

Chapter 2

Theoretical Background

We present the theoretical framework for hydrogen production methods, as well as introduce two machine learning techniques: XGBoost and Artificial Neural Networks (ANNs).

2.1 Methods of hydrogen production

Hydrogen gas can be produced through various methods, including:Steam Methane Reforming (SMR): In this process, natural gas (primarily methane) reacts with high-temperature steam in the presence of a catalyst, typically nickel. The reaction produces hydrogen gas and carbon dioxide as byproducts. Electrolysis: Electrolysis involves passing an electric current through water (H2O) to split it into hydrogen gas (H2) and oxygen gas (O2). This process requires a source of electricity and can be achieved using renewable sources like solar or wind power. Partial Oxidation: Partial oxidation involves reacting a hydrocarbon fuel, such as methane or propane, with a limited supply of oxygen. This reaction produces hydrogen gas, carbon monoxide, and some carbon dioxide. The resulting gas mixture can be further processed to purify the hydrogen. Biomass Gasification: Biomass, such as agricultural waste or wood chips, can be converted into hydrogen gas through gasification. The biomass is heated in a low-oxygen environment, producing a mixture of hydrogen, carbon monoxide, methane, and other gases. The gas can be further processed to extract and purify the hydrogen. Photoelectrochemical (PEC) Water Splitting: PEC utilizes a semiconductor material that absorbs sunlight to facilitate the water splitting process. The absorbed photons generate electron-hole pairs, which drive the electrochemical reactions to produce hydrogen gas and oxygen gas. These are just a few examples of methods used to produce hydrogen gas. The choice of method depends on factors such as availability of feedstock, energy source, cost, and environmental considerations.

2.2 General Machine Learning Algorithms

Machine learning algorithms are to use statistical techniques to learn from data and improve performance on a specific task.

2.2.1 Definition of Machine Learning Algorithms

Machine learning (Pedregosa et al., 2011; Tian et al., 2016; Bishop, 2006) has garnered substantial interest and recognition in recent times. Its appeal stems from its capacity to learn from data and generate predictions or make decisions without the need for explicit programming. Now AI has evolved into a complex field involving many different categories such as Machine Learning, Neural Networks, Deep learning, and so on see figure2.1. The term "machine learning" was first coined in 1959 by Arthur Samuel, an IBM scientist who developed a checkers-playing program that improved its performance through self-play and learning from its mistakes. Since then, machine learning has evolved into a complex field that includes a range of techniques and approaches for solving a variety of problems in many domains.

Machine learning and AI have made a profound impact across various sectors of society, encompassing healthcare, transportation, education, and more. Notably, personalized medicine has advanced through machine learning, leading to improved patient outcomes. Additionally, machine learning aids in the detection of fraud and prevention of financial crime. It also optimizes transportation systems, mitigating traffic congestion. Furthermore, machine learning contributes to personalized education, enhancing learning outcomes for individuals. However, there are also concerns about the ethical and societal implications of machine learning, such as bias in algorithms, job displacement, and privacy concerns.

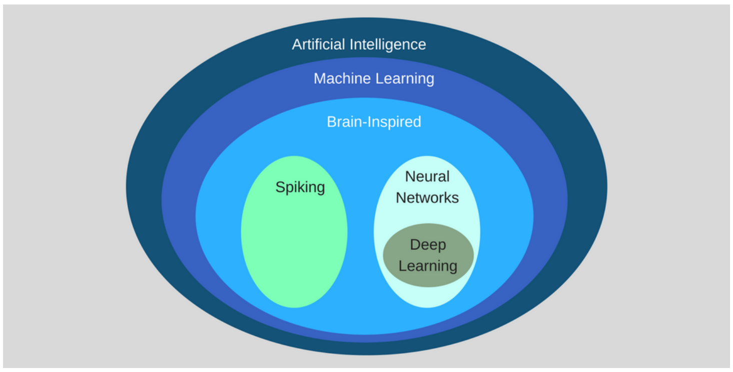


Figure 2.1: Relation between AI ML DL

2.2.2 General Types of Machine Learning Algorithms

There are 3 main categories of machine learning:

- **Supervised Learning.** Supervised learning is a type of machine learning in which a model is trained on labeled data, known as the training set (Hastie et al., 2009; Bishop, 2006; Ng, 2018). Supervised learning is often used for two main types of problems: classification and regression. Classification involves predicting which category or class a new piece of data belongs to, while regression involves predicting a continuous numerical value. For example, a supervised learning model might be used to predict whether an email is spam or not (classification), or to predict the price of a house based on its features (regression) (Mitchell, 1997; Bengio et al., 2016; Hastie et al., 2015). Supervised learning algorithms include linear regression, logistic regression, decision trees, random forests, and support vector machines, among others.

- **Unsupervised Learning.** Unsupervised learning is another type of machine learning in which the data in the training set is unlabeled. This means that the model is not given any specific target or output to predict, but rather it must learn the underlying structure or patterns in the data itself. This makes unsupervised learning more challenging than supervised learning, as there is no "correct" output to compare the model's predictions against. To extract useful information from unlabeled data, unsupervised learning algorithms use techniques such as clustering, dimensionality reduction, and association rule learning. Clustering involves grouping similar data points together into clusters, while dimensionality reduction involves finding a lowerdimensional representation of the data that preserves its important features. Association rule learning involves discovering relationships or patterns between different variables in the data. Unsupervised learning algorithms use mathematical reasoning processes or similarity measures to identify patterns or structures in the data, without the need for labeled examples. The applications of unsupervised learning are numerous, including anomaly detection, recommendation systems, and natural language processing. Unsupervised learning can also be used as a preprocessing step for supervised learning, by identifying important features or reducing the dimensionality of the data.

- **Reinforcement Learning.** Reinforcement learning (Sutton and Barto, 2018; Goodfellow et al., 2014; Karypis, 2004) is a type of machine learning that is different from supervised and unsupervised learning.The goal of the machine is to learn a policy or strategy that will maximize the total reward it receives over time. A classic example of reinforcement learning is playing a game, where the machine learns to make the best moves based on the rewards it receives. For example, in a game of chess, the machine may receive a positive reward for capturing an opponent's piece, but a negative reward for losing its own piece. Over time, the machine learns which moves lead to the highest total reward, and develops a strategy for winning the game. Reinforcement learning algorithms use a technique called "trial and error" to learn the optimal policy. The machine explores different actions in the environment and observes the resulting rewards, and then updates its policy based on the rewards it received. Over time, the machine learns to associate certain actions with positive rewards and others with negative rewards, and adjusts its behavior accordingly. Reinforcement learning has applications in a variety of fields, such as robotics, game playing, and autonomous systems. It has been used to teach robots to perform complex tasks such as assembling objects, and to develop algorithms for self-driving cars. Reinforcement learning is also being used to develop intelligent agents for video games, which can learn to play and win against human players.

2.3 Two algorithms of machine learning

We elaborate on the following two models, primarily utilized for experimentation purposes:

2.3.1 Extreme Gradient Boosting-XGBOOST

XGBoost, short for Extreme Gradient Boosting, is a powerful gradient boosting framework that leverages decision trees as its foundational components. It is renowned for its exceptional performance in various machine learning tasks. This powerful tool is widely acclaimed for its exceptional speed, accuracy, and exceptional handling of large and complex datasets. One of the key features of XGBoost is the use of a gradient descent optimization algorithm that repeatedly improves the model's predictive ability by minimizing a loss function.

Due to its unmatched accuracy and remarkable speed, XGBoost has emerged as one of the most sought-after and extensively employed machine learning algorithms in the industry. Its popularity stems from its ability to deliver impressive results across a wide range of applications. This framework can effectively handle a wide range of applications, including regression, classification, and ranking problems.

Furthermore, XGBoost incorporates several advanced techniques that make it stand out from other machine learning algorithms. For instance, it includes a regularization parameter that helps prevent overfitting and ensures better generalization of the model. Additionally, XGBoost supports parallel computing, which significantly reduces the training time and improves performance.

Overall, XGBoost is an incredibly powerful and versatile tool that has revolutionized the field of machine learning. Its ability to handle large datasets, speed, and accuracy make it an ideal choice for solving real-world problems in a wide range of industries.

2.3.2 Artificial Neural Network-ANN

Artificial neural networks (ANNs) are a powerful class of machine learning algorithms that have transformed the field of artificial intelligence. They are loosely based on the structure and function of the human brain.

Artificial Neural Networks (ANNs) are composed of interconnected layers of nodes or neurons that handle and transmit information. Each neuron receives inputs from other neurons in the preceding layer and generates an output, which is then passed on to the subsequent layer. This layered structure makes ANNs particularly well-suited for processing intricate data inputs, such as images, sounds, or text. ANNs have demonstrated their efficacy in various tasks, including pattern recognition, classification, and regression.

One of the key strengths of ANNs is their ability to learn and adapt to new data. They can be trained using large datasets.

In recent years, there have been significant advances in the design and training of ANNs, with the development of new techniques. These advanced architectures have significantly improved the performance of ANNs on a wide range of tasks.

Artificial Neural Networks (ANNs) are indeed a powerful and versatile class of machine learning algorithms that are revolutionizing problem-solving across diverse fields. Their impact spans domains like computer vision, natural language processing, robotics, and more. As research in this field progresses, ANNs are poised to assume an even more significant role in the advancement of intelligent systems and applications. Their ability to learn from data, adapt to new information, and make complex decisions has the potential to drive transformative innovations in the future.

Chapter 3

Experiment Design

3.1 Two situations of Liquid hydrogen release

3.1.1 Outdoor Leakage

The collection of atmospheric data, such as atmospheric pressure, humidity, wind speed, and direction, is crucial in understanding the behavior of a cryogenic fluid spill. These factors can affect the formation and propagation of a gas cloud and the safety measures needed during refueling operations (Gavelli et al., 2016; Jia et al., 2019). The use of thermocouples to measure the temperature is essential in evaluating the safety of cryogenic fluid spills. The thermocouples were placed on the test pad and in the field to measure the temperature of the liquid hydrogen and the surrounding area.This information is important in understanding the heat transfer during the spill and the potential for ignition. Thermocouples were used to measure the temperature. Oxygen sensors were used to measure the amount of hydrogen in the air, and the hydrogen concentrations were calculated based on the amount of oxygen in the air (Zhang et al., 2019; Chi et al., 2018). The measurement of gas concentration using oxygen sensors is also essential in evaluating the safety of cryogenic fluid spills. This information is crucial in establishing safety distances for personnel and equipment during refueling operations involving cryogenic fluids (Hasanzadeh and Abbassi, 2017; Zeng et al., 2018). The wind's direction and speed were monitored using sensors inserted in a pole near the pad. This information is important in understanding the behavior of the gas cloud and the potential for ignition during refueling operations. The wind's direction and speed can affect the gas cloud's propagation and the safety measures needed to protect personnel and equipment(Ferrari, 2022; Zhu et al., 2017).

The results of the outdoor leakage testing of liquid hydrogen showed that the liquid pool's development on the ground was dependent on the orientation of the release, whether it was pointed vertically down or horizontally. The liquid pool only extended 0.5 meters from the release point, which is an important safety factor to consider during refueling operations involving cryogenic fluids. Due to the extremely low boiling point of hydrogen (20 K), the release of a cryogenic fluid may cause condensation and freezing of air components on the ground. This phenomenon can increase the likelihood that the flammable mixture will explode in the event of an ignition. Therefore, safety measures need to be taken into account during refueling operations to prevent such occurrences. This information is important for establishing safety distances for personnel and equipment during refueling operations involving cryogenic fluids. No spontaneous ignition was seen in any of the tests. However, the information gathered during the testing can help improve safety measures and prevent accidents in the future. The use of multiple sensors and thermocouples provided a comprehensive analysis of the experimental results, allowing for a more precise evaluation of the safety measures needed during refueling operations. Overall, the outdoor leakage testing of liquid hydrogen was an important step in evaluating the safety of cryogenic fluid refueling operations. The results provided valuable insights into the behavior of liquid hydrogen spills and gas cloud formation, which can help improve safety measures and prevent accidents in the future.

3.1.2 Closed room and Ventilation mast studies.

The emission of liquid hydrogen from the bottom has been studied extensively in laboratory simulations of cryogenic fluid leaks. Confined rooms and ventilation derricks have been used to investigate the behavior of liquid hydrogen in different environments. In these studies, the storage tank is connected to the TCS (tank connection space), as shown in Figure 3.1. The environmental conditions during the tests are recorded, including measurements of the ambient pressure, humidity, wind speed, and direction. The wind speed and direction measurements are particularly important in understanding the behavior of the liquid hydrogen in different wind conditions. Near the launch pad, sensors are positioned on a mast to measure the temperature and gas concentration during the tests. In the closed-room research environment, 10 thermocouples and 15 thermocouples were installed on the TCS floor to measure the temperature during the liquid hydrogen emission. Additionally, 3 thermocouples were installed within the ventilation mast, and a thermocouple was positioned in the TCS, as shown in Figure 3.2, to provide more detailed temperature measurements. The use of multiple thermocouples and sensors in laboratory simulations of cryogenic fluid leaks provides a more comprehensive understanding of the behavior of liquid hydrogen in different environments. The data gathered from these tests can be used to develop more accurate models and simulations of liquid hydrogen emissions, improving safety measures in the handling and transport of cryogenic fluids.

In total, 8 tests were carried out to study the behavior of liquid hydrogen in different environments. One of the tests involved decontaminating a sealed area first to avoid the production of nitrogen-containing combustible mixtures that could explode upon ignition. During the test, it was observed that the hydrogen concentration in the room reached 100% in less than 30 seconds after the start of the release. However, there was no pressure buildup in the room due to its not being completely airtight. The release of liquid hydrogen led to its evaporation, and accumulation was seen in the room. Interestingly, the ventilation mast did not become clogged due to condensation or solidification of air components inside of it. The test also provided evidence that airborne particles either condensed or froze in the presence of liquid hydrogen. It was observed that the liquid hydrogen settled on the ground and evaporated quickly, but the floor surface remained wet for a considerable amount of time. These findings are important for understanding the behavior of liquid hydrogen in enclosed spaces and developing safety protocols for handling and transporting cryogenic fluids. The data obtained from these tests can be used to refine existing models and simulations of liquid hydrogen emissions in enclosed spaces, ultimately leading to improved safety measures for the handling and transport of cryogenic fluids.

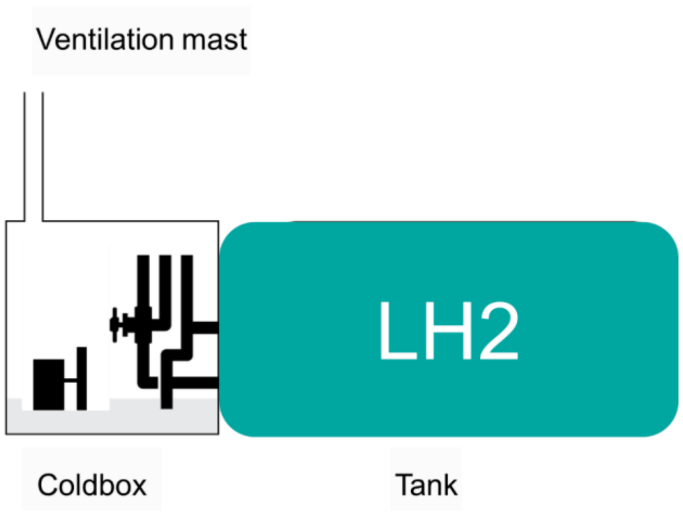


Figure 3.1: Illustration of a ColdBox, which is connected to a ventilation mast(Aaneby et al., 2021).

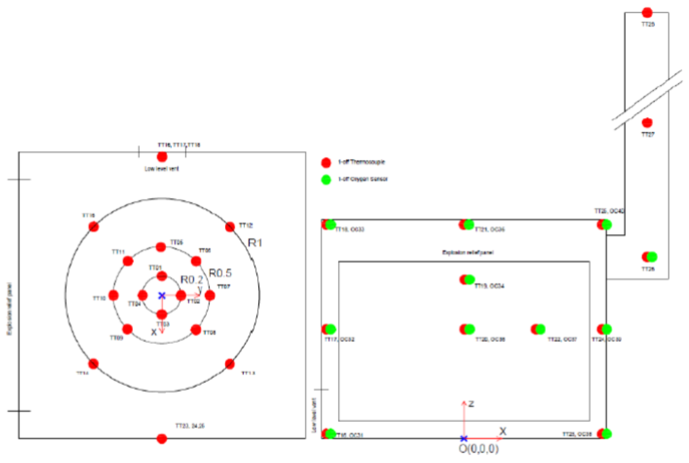


Figure 3.2: Ventilation mast studies conducted by (Aaneby et al., 2021) involved the placement of thermocouples and oxygen sensors at specific locations. In the top view of the setup, with the floor of the Thermodynamic Control System (TCS) to the left, the following placements were made.

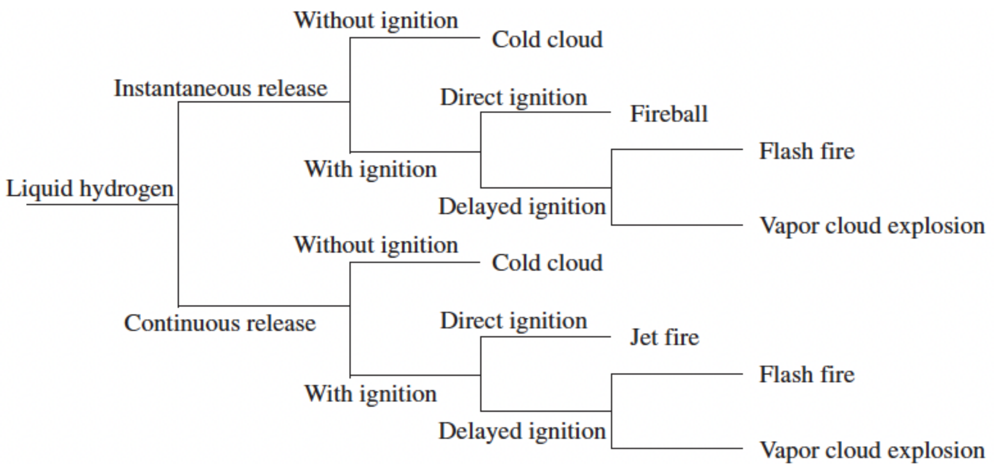


Figure 3.3: Event tree of releases from liquid hydrogen tank.(Li et al., 2012).

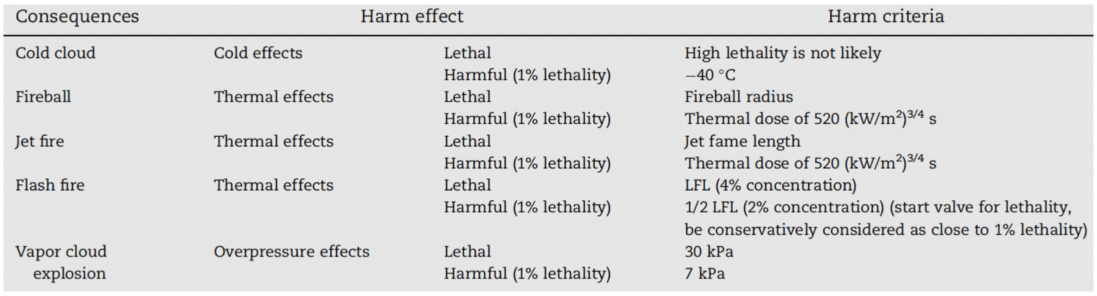
3.2 Liquid Hydrogen Release Consequences

Hydrogen storage and transportation present significant safety challenges. In addition to the potential hazards of high pressure and flammability, cryogenic hydrogen storage and release also introduce cold effects that need to be considered in safety assessments. The low boiling point of hydrogen (-252.87°C) means that it is stored and transported in a cryogenic liquid state at extremely low temperatures, and any leakage could cause rapid cooling of the surrounding area, creating a hazard for personnel.

In summary, the safety issues associated with cryogenic liquid hydrogen storage and release are complex and multifaceted, requiring a comprehensive approach to hazard identification, risk assessment, and safety management. Further research is needed to better understand the safety risks associated with hydrogen storage and transportation, and to develop more effective safety strategies and technologies to reduce these risks.

- Criteria for cold effects: The harm criterion for cold effects, as outlined in IGC Doc (Doc, 2007), is a temperature below -40°C, which corresponds to a 1% probability of fatality to the general population. This criterion is crucial to prevent cryogenic burns and other cold-related injuries in the event of a release from liquid hydrogen tanks. Proper safety measures and precautions must be taken to ensure that individuals are not exposed to such low temperatures, as they can cause severe harm or even death.

- Criteria for overpressure effects: Overpressure effects from hydrogen explosions can



Consequences	Harm effect		Harm criteria
Cold cloud	Cold effects	Lethal	High lethality is not likely
		Harmful (1% lethality)	−40 °C
Fireball	Thermal effects	Lethal	Fireball radius
		Harmful (1% lethality)	Thermal dose of 520 $(kW/m^2)^{3/4}$ s
Jet fire	Thermal effects	Lethal	Jet fame length
		Harmful (1% lethality)	Thermal dose of 520 $(kW/m^2)^{3/4}$ s
Flash fire	Thermal effects	Lethal	LFL (4% concentration)
		Harmful (1% lethality)	1/2 LFL (2% concentration) (start valve for lethality, be conservatively considered as close to 1% lethality)
Vapor cloud explosion	Overpressure effects	Lethal	30 kPa
		Harmful (1% lethality)	7 kPa

Figure 3.4: The harm criteria for the three types of harmful effects resulting from releases from liquid hydrogen tanksLi et al. (2012).

cause severe injuries and fatalities. A high lethality value of 30 kPa is recommended as the criterion for lethal effects, indicating that individuals are at risk of immediate death if exposed to overpressure levels of this magnitude. Proper safety measures and precautions must be taken to prevent or mitigate the risks associated with overpressure effects from hydrogen explosions. As for harmful effects, according to IGC75/07E, 7 kPa is suggested as the harm criterion, corresponding to a 1% probability of fatality. This criterion indicates that individuals are at risk of significant harm or injury if exposed to overpressure levels of this magnitude, but immediate death is not expected. Proper safety measures and precautions must be taken to prevent or mitigate the risks associated with harmful overpressure effects from hydrogen explosions. It is crucial to implement appropriate safety measures to prevent overpressure effects from occurring and to ensure the safety of individuals in the vicinity of liquid hydrogen tanks.

Figure 3.4 summarizes the harm criteria for the three types of harmful effects resulting from releases from liquid hydrogen tanks. It is important to note that all effects are calculated at a height of one meter, rather than ground level, as this is considered to be a good average height for exposure to both adults and children. By using these criteria and considering the height of exposure, appropriate safety measures can be taken to prevent or mitigate the risks associated with releases from liquid hydrogen tanks.

3.3 Database

3.3.1 Data Used

Three different databases have been developed:

- The data used to develop machine learning algorithms for predicting hydrogen releases and their potential effects were organized into three databases. The first database was focused on predicting the condensation or freezing of air components in outdoor leakage studies. The data for this database was collected from thermocouples located within 30 meters of the release point, and those installed within the test pad were not included due to the lack of air beneath the surface.
- The second database focused on predicting hydrogen concentration within the gas cloud in outdoor leakage studies. The data for this database was collected from all oxygen sensors located at 30, 50, and 100 meters from the release point in the wind direction.
- The third database, also referred to as indoor leakage studies, focused on predicting the condensation or freezing of air components in closed room studies. This database exclusively used data from the 25 thermocouples installed within the test chamber.

To develop machine learning algorithms for predicting hydrogen releases and their potential effects, three databases were created. The first database excluded thermocouples located more than 30 meters from the release point and those installed inside the test pad due to the lack of air beneath the surface. This database was used to forecast the condensation or freezing of air component parts. The second database included all oxygen sensors located at 30, 50, and 100 meters from the release point in the wind direction. This database was used to determine if the concentration of hydrogen in the gas cloud had reached the lower flammability limit or not. The third database focused on examining the production of liquid or solid oxygen inside the TCS and conducting a comparison with the outdoor example. This database was exclusively constructed using the 25 thermocouples installed inside the TCS. All databases were created by taking into account a row for each temperature or concentration value measured by each thermocouple or sensor for each instant of time throughout the entire experimental test. The data gathered from each test was combined to create the databases, and this data can be used to train machine learning algorithms to predict hydrogen releases and their potential effects. In all three databases, a row was created for each temperature or concentration value measured by each thermocouple or sensor for each instant of time throughout the entire experimental test. The data gathered from each test was combined to create the databases, which can be used to train machine learning algorithms to predict the effects of hydrogen releases.

3.3.2 Labels

In the first and third databases, the focus is on predicting whether or not there will be the deposition of liquid or solid oxygen for each point on the pad. For this purpose, two labels have been investigated. If the thermocouple's measurement after 200 seconds is less than either the melting point or the boiling point of pure oxygen, a label of "1" is assigned, indicating that air oxygen will condense or solidify. If not, the label assigned is 0. It should be noted that this method is not applicable in one case of the indoor studies where the enclosed chamber had already been purged with nitrogen, thus there was no oxygen present when the release began. Therefore, each thermocouple has a label of 0 for this particular case. In the second database, the focus is on predicting whether or not there will be hydrogen concentrations in the air that are higher than the lower flammability limit (LFL) after 200 seconds, depending on the location on the pad. A single label has been taken into account for this purpose. If the hydrogen concentration is higher than the LFL, a label of 1 is assigned, otherwise, the label assigned is 0. At atmospheric pressure, hydrogen's LFL is 4% vol. The chosen 200-second time window for assigning the labels is based on the response time of water systems following a hydrogen release, as hydrogen sensor-activated systems have been studied in this thesis as mitigation strategies for the effects of liquid hydrogen leakage.

Based on the information provided, it appears that the study investigated the melting and boiling points of oxygen at an average pressure of 0.96 bar. The boiling point (Tb) was found to be -183.5 °C, while the melting point (Tm) was found to be -218.79 °C. Additionally, the study examined the response time necessary to forecast the condensation, freezing, or development of an atmosphere that is combustible. The chosen response time was 200 seconds, and the models were able to forecast the label that would correlate to a specific thermocouple after this time period. The reason for selecting the 200-second interval was based on the response time of water systems following a hydrogen release. The study also investigated the use of hydrogen sensors as mitigation strategies for the effects of liquid hydrogen leakage.

The study involves three databases. In the first and third databases, two tags are allocated. If oxygen condenses, the label assigned is 1; otherwise, the allocation label is 0. The boiling temperature of oxygen is -183.5 °C, and its melting point is -218.79 °C at a pressure of 0.96 bar. In the second database, two labels are assigned as well. After 200 seconds, if the concentration of hydrogen is zero, the label will be higher than the LFL; otherwise, the allocation label will be 0. The LFL for hydrogen is 4% vol at atmospheric pressure. The 200-second time window was selected based on the response time of water systems after a hydrogen release reaction. The models in the study used this time frame to forecast the label that would correlate to a specific thermocouple. In summary, the study examined the properties of oxygen and hydrogen under specific conditions and utilized a 200-second time window to forecast the label that would correspond to a specific thermocouple in the three databases studied.

Each of the five experimental tests in the first database uses 96 thermocouples in total. A total of 60 concentration sensors are used in each of the four experimental experiments in the second database. Ten experimental tests totaling 50 thermocouples, all of which were placed within the container, are included in the third database. Since some of the experimental experiments were not long enough to make a reliable prediction or were carried out in different wind conditions than the others, not all of them have been taken into account in the databases. Therefore, only the experiments that meet the criteria for inclusion have been included in the databases. The increase in the number of thermocouples and concentration sensors in the databases will enable more accurate measurements and predictions to be made. This, in turn, will lead to better understanding and control of the experimental conditions, which is crucial for the success of any scientific research.

3.4 Exploratory Data Analysis

3.4.1 First Database

The first database in our study includes a dataset with the following features: time, ambient pressure, release rate, humidity, release orientation, wind direction high, wind direction low, wind speed high, wind speed low, liquid oxygen, and solid oxygen. The objective of this dataset is to predict the distribution of liquid oxygen and solid oxygen based on the given features. Specifically, the positive samples in the liquid oxygen class constitute 42.54% of the total, while the negative samples account for 57.46%. For the solid oxygen class, the positive samples constitute 8.48% of the total, while the negative samples account for 91.52%, indicating a significant class imbalance.

The dataset does not contain any missing values, except for the release orientation feature, which is a categorical variable with two possible value: horizontal and vertical.Only vertical means hydrogen release.

Due to the significant class imbalance observed in the dataset, Our analysis revealed that the distribution of the dataset had become significantly more uniform following the outlier removal process as shown in Fig 3.5. This allowed us to gain a better understanding of the relationships between the features and the distribution of liquid oxygen and solid oxygen. As shown in the graphs, the distribution of the features exhibited a more balanced spread across the different values of liquid oxygen and solid oxygen. This indicates that the outlier removal process was effective in improving the uniformity of the dataset and enabled us to more accurately assess the relationships between the features and the target variables. Overall, our findings suggest that outlier removal is an effective technique for improving the uniformity of datasets and can lead to a better understanding of the relationships between features and target variables.

3.4.2 Second Database

The objective of this study is to predict the distribution of Hydrogen concentration based on these indicators. Specifically, the dataset consists of a highly imbalanced sample distri-

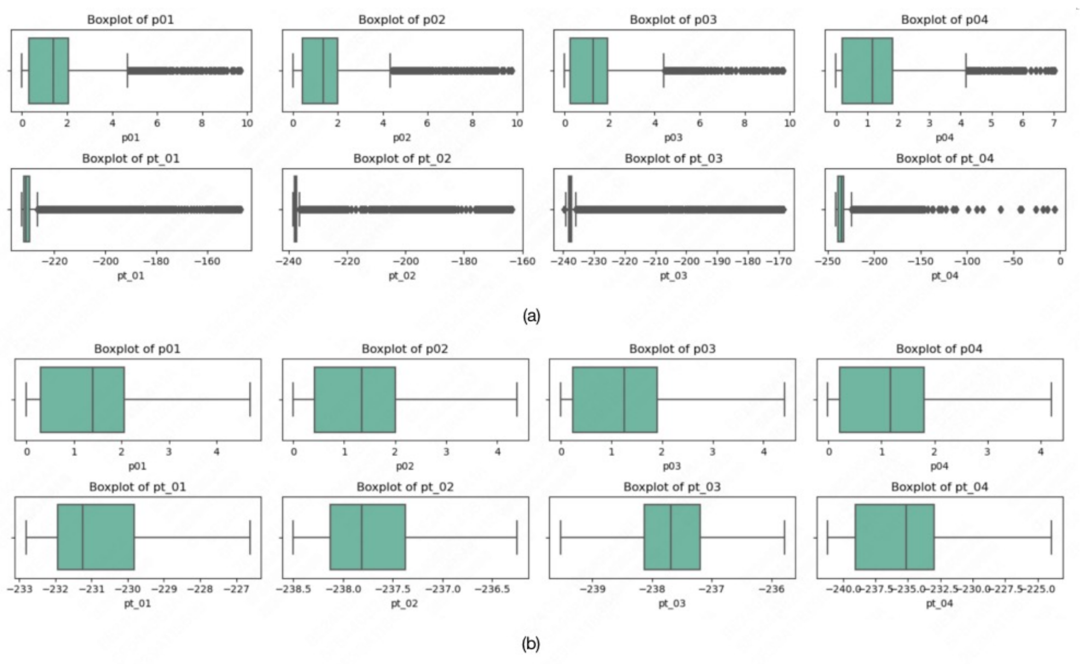


Figure 3.5: Boxplot of p01 to p04, pt_01 to pt_04 by liquid oxygen and solid oxygen in the first database before (a) and after (b) removing outliners.

bution, with positive samples accounting for only 1.25% and negative samples accounting for 98.75% . None of the indicators have missing values, and all indicators except for release orientation are numerical features.

To address the issue of non-uniform distributions of certain features, including 'p01', 'p02', 'p03', 'p04', 'pt_01', 'pt_02', 'pt_03', and 'pt_04', we performed outlier removal and re-examined their relationships with the distribution of liquid oxygen and solid oxygen as shown in Fig. 3.6. Our analysis showed that the outlier removal process effectively improved the uniformity of the distributions of these features, which allowed us to gain a better understanding of the relationships between the features and target variables. Overall, our findings suggest that outlier removal is an effective technique for improving the uniformity of feature distributions in datasets, and can lead to a more accurate assessment of the relationships between the features and target variables. Additionally, our analysis highlights the significant differences in the distributions of certain features between the positive and negative samples of liquid oxygen, which underscores the need for appropriate techniques, such as oversampling or undersampling, to address the class imbalance issue during model training.

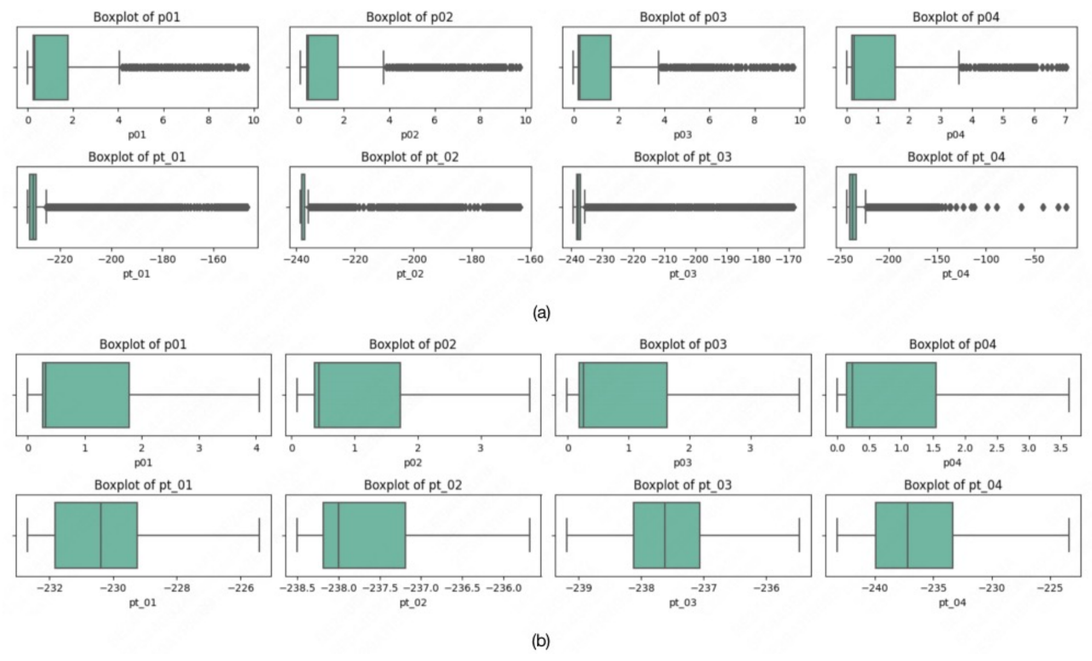


Figure 3.6: Boxplot of p01 to p04, pt_01 to pt_04 by liquid oxygen in the second database before (a) and after (b) removing outliners.

3.4.3 Third Database

The third database in our study contains a dataset with several features, including time, ambient pressure, release rate, humidity, purge, obstacles, vent panel, sealing, p01, p02, p03, p04, pt_01, pt_02, pt_03, pt_04, wind direction high, and wind direction low. The objective of this dataset is to predict the distribution of liquid oxygen and solid oxygen based on the given features of time, ambient pressure, release rate, humidity, release orientation, wind direction high, and wind direction low.

The positive samples in the liquid oxygen class constitute 52.85% of the total, while the negative samples account for 47.18%. The positive samples in the solid oxygen class constitute 24.05% of the total, while the negative samples account for 75.95%. This indicates a class imbalance issue in the dataset. There are no missing values in the dataset. Our analysis reveals that the class imbalance issue in the dataset requires appropriate techniques such as oversampling or undersampling to address this issue during model training. Moreover, our findings highlight the significant differences in the distributions of categorical features between the positive and negative samples of liquid oxygen and solid oxygen. Therefore, careful consideration must be given to these features during model training and analysis.

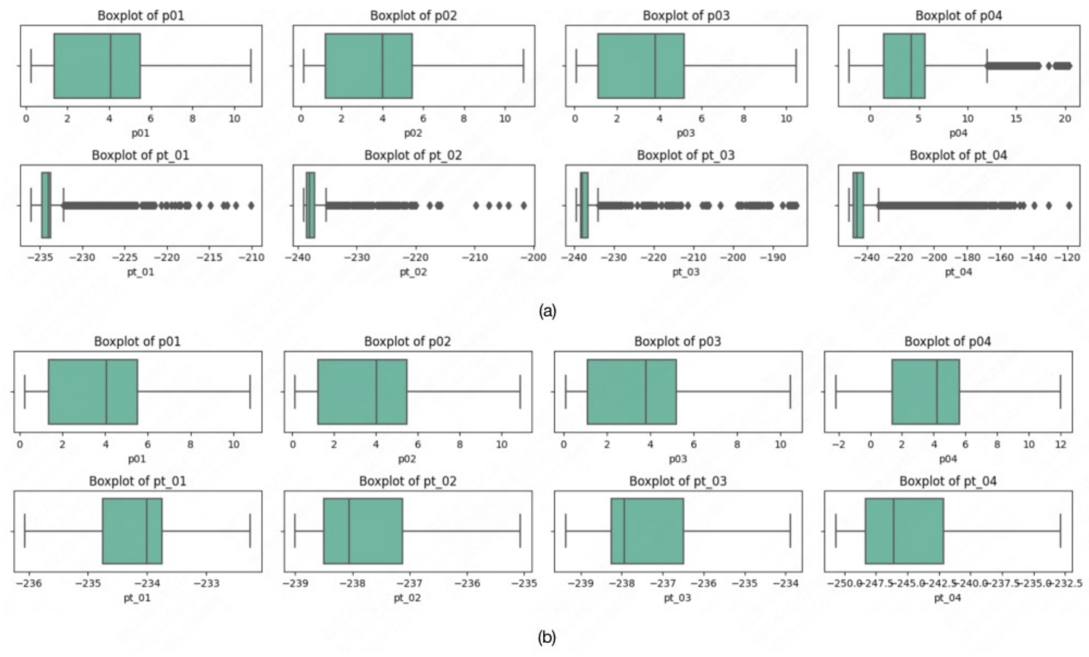


Figure 3.7: Boxplot of p01 to p04, pt_01 to pt_04 by liquid oxygen and solid oxygen in the third database before (a) and after (b) removing outliners.

To address the issue of non-uniform distributions of certain features, including ’p01’, ’p02’, ’p03’, ’p04’, ’pt_01’, ’pt_02’, ’pt_03’, and ’pt_04’, we performed outlier removal and re-examined their relationships with the distribution of liquid oxygen and solid oxygen as shown in Fig. 3.7. Our analysis revealed that the outlier removal process effectively improved the uniformity of the distributions of these features, which allowed us to gain a better understanding of the relationships between the features and the target variables. Our findings suggest that outlier removal is an effective technique for improving the uniformity of feature distributions in datasets, which can lead to a more accurate assessment of the relationships between the features and target variables. Furthermore, our analysis highlights the significant differences in the distributions of continuous variables between the positive and negative samples of liquid oxygen and solid oxygen, which should be taken into consideration during model training and analysis.

3.5 Model and Training details

In this study, we aimed to explore the physical properties of liquid and solid oxygen by constructing two separate training and testing datasets for each database. The datasets were divided into a 3:1 ratio for training and testing, respectively. We used two machine learning algorithms, including XGBoost and Artificial Neural Networks (ANN), to build predictive models for liquid and solid oxygen properties.

For XGBoost, we optimized the model parameters within the range of learning rates of 0.1, and maximum depth of 9. For ANN, we used a four-layer dense network with shapes of 64, 32, 8, and 1. We applied ReLU activation functions for all layers in the network. In addition, we identified the top five important features using feature importance ranking and used these features for building our models. To evaluate the performance of the models, we used confusion matrices and ROC curves.

Chapter 4

Results and Analysis

This section is dedicated to the assessment of the performance of two distinct models: XG-Boost and Artificial Neural Network (ANN). To compare the effectiveness of these methods on various datasets, several evaluation metrics including confusion matrix, accuracy, recall, precision, F1 score, and ROC curve are utilized. The primary objective of this evaluation is to determine the superior machine learning model for each dataset and label. By examining the outcomes of these evaluations, valuable insights regarding the proficiency of these machine learning models in handling the larger database1,2,3 dataset can be obtained. The findings from these evaluations will be presented through a comprehensive comparison of the different methods and their respective metrics across each dataset and label.

4.1 Experiment Scenario 1-Outdoor leakage prediction

4.1.1 Xgboost

Liquid oxygen.

Liquid oxygen is a critical industrial material that requires accurate monitoring and control to ensure safe and efficient operations. In this study, we evaluated the performance of the XGBoost model in predicting the distribution of liquid oxygen in the database1 dataset. The results are shown in Table 4.1, Fig 4.1, and Fig 4.2. The results of the evaluation demonstrate that the XGBoost model achieved excellent performance in predicting the distribu-

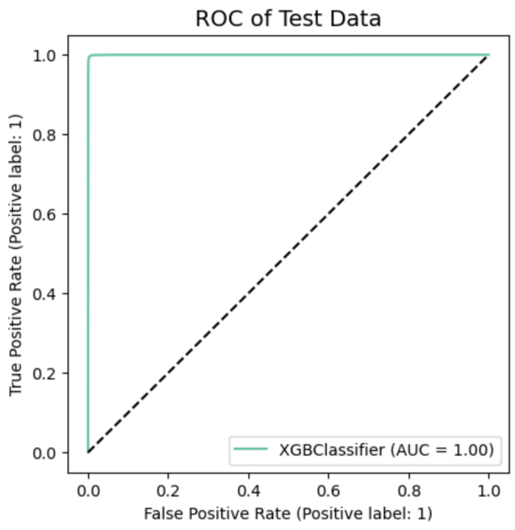


Figure 4.1: ROC curve modelled by XGBoost for the first database for liquid oxygen

tion of liquid oxygen, with high precision, recall, and F1-score for both positive and negative samples. The precision, recall, and F1-score for class 0 were 0.99627, 0.99685, and 0.99656, respectively, indicating that the model is highly accurate in predicting negative samples of liquid oxygen. For class 1, the precision, recall, and F1-score were 0.99575, 0.99497, and 0.99536, respectively, indicating that the model is highly accurate in predicting positive samples of liquid oxygen. In terms of overall performance, the model achieved an accuracy of 0.99605 on the test data, indicating that the model can accurately distinguish between positive and negative samples of liquid oxygen in the database1 dataset. The confusion matrix further supports the model's high performance, with 100,000 correctly predicted samples for class 0 and 76,481 correctly predicted samples for class 1.

In conclusion, the XGBoost model demonstrated exceptional performance in predicting the distribution of liquid oxygen in the database1 dataset. The model's high accuracy, precision, recall, and F1-score, along with its perfect discriminatory power as demonstrated by the AUC score, make it a powerful tool for predicting the distribution of liquid oxygen in industrial settings.

Solid oxygen.

In this analysis, we evaluated the performance of the XGBoost model in predicting the distribution of solid oxygen in the database1 dataset. The results showed that the XGBoost model achieved high precision, recall, and F1-score for both the positive and negative classes.

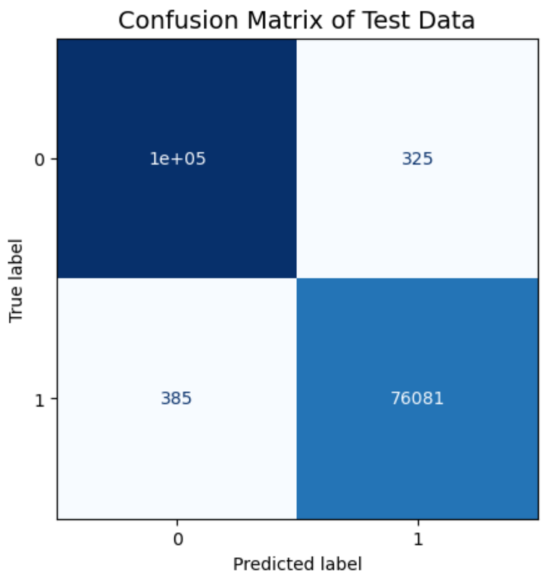


Figure 4.2: Confusion Matrix modelled by XGBoost for the first database for liquid oxygen

The precision of the model for the negative class was 0.99936, while the recall and F1-score were 0.99988 and 0.99962, respectively. For the positive class, the precision was 0.99875, the recall was 0.99305, and the F1-score was 0.99589. The results are shown in Table 4.2, and Fig 4.3. The overall accuracy of the model was high, with an accuracy of 0.99930. The confusion matrix showed that the model classified the majority of the observations correctly, with only a small number of false negatives and false positives. Specifically, the model predicted 164476 true negatives and 15136 true positives, while misclassifying only 19 observations as false negatives and 106 observations as false positives. These results suggest that the XGBoost model is highly effective in predicting the distribution of solid oxygen in the database1 dataset. The high precision and recall values for both classes demonstrate that the model is accurate in predicting both positive and negative classes. The high accuracy of the model

Table 4.1: **Performance of XGBoost for the first database for liquid oxygen**

Class	precision	recall	f1-score	support
0	0.99627	0.99685	0.99656	103271
1	0.99575	0.99497	0.99536	76466
accuracy	-	-	0.99605	179737
macro avg	0.99601	0.99591	0.99596	179737
weighted avg	0.99605	0.99605	0.99605	179737

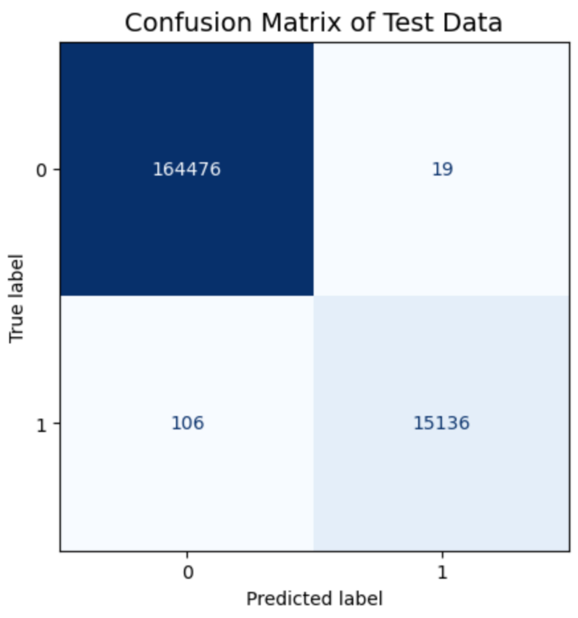


Figure 4.3: Confusion Matrix modelled by XGBoost for the first database for solid oxygen

indicates that it is reliable and can be used for predicting the distribution of solid oxygen in similar datasets. The results have important implications for the development of more accurate predictive models and could potentially inform the design of more effective monitoring and control systems for solid oxygen distribution in industrial settings. Further research is needed to evaluate the model's performance on other datasets and explore its potential for industrial applications. In summary, the analysis of the XGBoost model's performance in predicting the distribution of solid oxygen in the database1 dataset demonstrated that the model achieved high accuracy, precision, and recall. The results suggest that the model is effective in predicting both positive and negative classes, making it a potentially valuable tool for predicting the distribution of solid oxygen in similar datasets.

Table 4.2: **Performance of XGBoost for the first database for solid oxygen**

Class	precision	recall	f1-score	support
0	0.99936	0.99988	0.99962	164495
1	0.99875	0.99305	0.99589	15242
accuracy	-	-	0.99930	179737
macro avg	0.99905	0.99647	0.99775	179737
weighted avg	0.99930	0.99930	0.99930	179737

4.1.2 ANN

Liquid oxygen.

The accurate prediction of the distribution of liquid oxygen is essential for ensuring safe and efficient industrial operations. In this study, we evaluated the performance of the Artificial Neural Network (ANN) model in predicting the distribution of liquid oxygen in the database1 dataset. The results of the evaluation demonstrate that the ANN model achieved reasonable performance in predicting the distribution of liquid oxygen, with good precision, recall, and F1-score for both positive and negative samples. The results are shown in Table 4.3, Fig 4.5, and Fig 4.6. The precision, recall, and F1-score for class 0 were 0.98331, 0.98025, and 0.98178, respectively, indicating that the model is highly accurate in predicting negative samples of liquid oxygen. For class 1, the precision, recall, and F1-score were 0.97343, 0.97753, and 0.97548, respectively, indicating that the model is reasonably accurate in predicting positive samples of liquid oxygen. In terms of overall performance, the model achieved an accuracy of 0.97909 on the test data, indicating that the model can accurately distinguish between positive and negative samples of liquid oxygen in the database1 dataset. The confusion matrix further supports the model's performance, with 100,000 correctly predicted samples for class 0 and 74,748 correctly predicted samples for class 1. However, compared to the XGBoost model, the ANN model has lower performance in predicting the distribution of liquid oxygen. The precision, recall, and F1-score for both positive and negative samples of liquid oxygen are lower in the ANN model compared to the XGBoost model. The results suggest that XGBoost may be a better model choice for predicting the distribution of liquid oxygen in the database1 dataset. In conclusion, the ANN model demonstrated reasonable performance in predicting the distribution of liquid oxygen in the database1 dataset. The model's performance was satisfactory, with good accuracy, precision, recall, and F1-score. However, compared to the XGBoost model, the ANN model had lower performance in predicting the distribution of liquid oxygen. Further research is needed to improve the performance of the ANN model or explore alternative models that can achieve better performance on this dataset.

To further understand the performance of the Artificial Neural Network (ANN) model in predicting the distribution of liquid oxygen in the database1 dataset, we analyzed the loss and accuracy of the model over time. The loss and accuracy are essential metrics that can provide insights into the model's performance and its ability to learn from the data as shown in Fig 4.4. The analysis showed that the ANN model's loss decreased steadily over time, indicating that the model is effectively learning from the data. In the first epoch, the loss was 0.221312, and the accuracy was 0.903711. Over the subsequent epochs, the loss decreased significantly, with the lowest loss achieved in the tenth epoch (0.102877). The model's accuracy also increased over time, reaching its highest value of 0.956448 in the tenth epoch. The validation loss and accuracy of the model also showed a similar trend, with the validation loss decreasing over time and the validation accuracy increasing. The validation loss was highest in the first epoch (0.190881) and lowest in the tenth epoch (0.101776), while the validation accuracy was lowest in the first epoch (0.918529) and highest in the tenth epoch (0.953747). These results suggest that the ANN model is learning effectively from the data and achieving high accuracy in predicting the distribution of liquid oxygen in the database1 dataset. The decreasing loss and increasing accuracy over time demonstrate that the model is improving its performance with each epoch, indicating that it has a high potential to perform well on other datasets. In summary, the analysis of the ANN model's loss and accuracy over time demonstrates that the model is learning effectively from the data and achieving high accuracy in predicting the distribution of liquid oxygen in the database1 dataset. The results have important implications for the development of more accurate predictive models and could potentially inform the design of more effective monitoring and control systems for liquid oxygen distribution in industrial settings. Further research is needed to evaluate the model's performance on other datasets and explore its potential for industrial applications.

Solid oxygen.

The ANN model was applied to predict the label for solid oxygen in database1. The precision, recall, and F1-score were reported as 0.99439, 0.99393, and 0.99416, respectively, for label 0 and 0.93484, 0.93944, and 0.93714, respectively, for label 1. The accuracy of the model was reported as 0.98931, indicating a relatively good performance. The confusion matrix for the model showed that out of 164,495 instances of label 0, 163,497 were correctly clas-

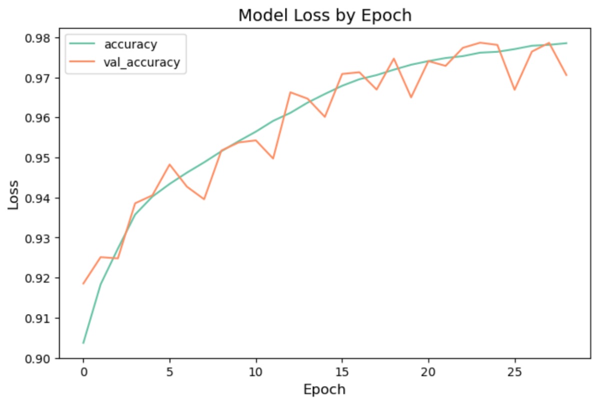


Figure 4.4: Loss in the first database by ANN for liquid oxygen

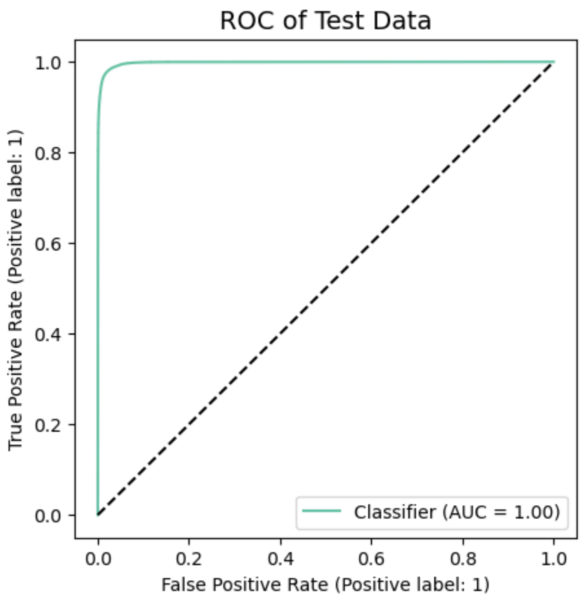


Figure 4.5: ROC curve modelled by ANN for the first database for liquid oxygen

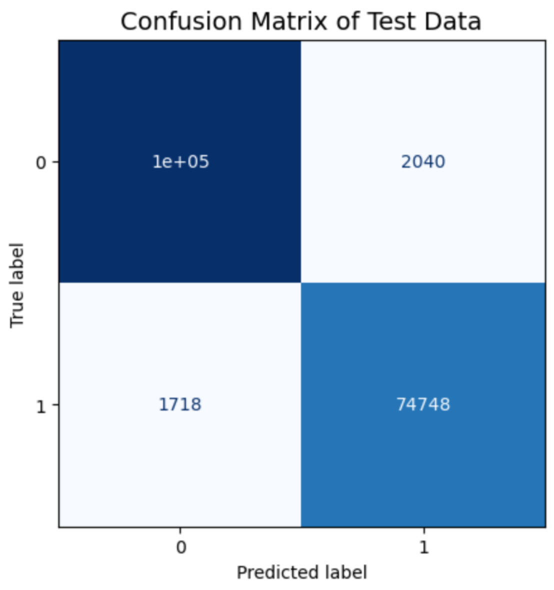


Figure 4.6: Confusion Matrix modelled by ANN for the first database for liquid oxygen

sified, while out of 15,242 instances of label 1, 14,319 were correctly classified. The results are shown in Table 4.4, and Fig 4.8. In conclusion, the ANN model achieved relatively good performance in predicting the label for solid oxygen in database1.

In this section, we present the loss and accuracy trends of the Artificial Neural Network (ANN) model for solid oxygen prediction in database1 as shown in Fig 4.7. As we can see, the training loss and accuracy improve significantly in the first few epochs. The model achieved an accuracy of 98.7% after only 10 epochs. The validation accuracy also improves with each epoch, indicating that the model generalizes well to unseen data. However, the validation loss increases significantly in the third epoch, indicating that the model may be overfitting to the training data. This is further supported by the drop in validation accuracy in the fourth epoch. To prevent overfitting, we can apply regularization techniques such as dropout or

Table 4.3: **Performance of ANN for the first database for liquid oxygen**

Class	precision	recall	f1-score	support
0	0.98331	0.98025	0.98178	103271
1	0.97343	0.97753	0.97548	76466
accuracy	-	-	0.97909	179737
macro avg	0.97837	0.97889	0.97863	179737
weighted avg	0.97911	0.97909	0.97910	179737

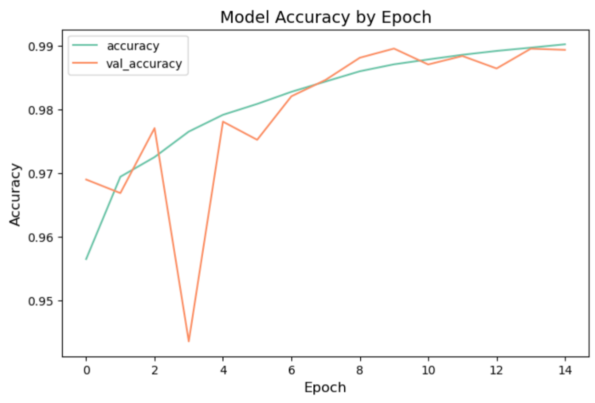


Figure 4.7: Loss in the first database by ANN for solid oxygen

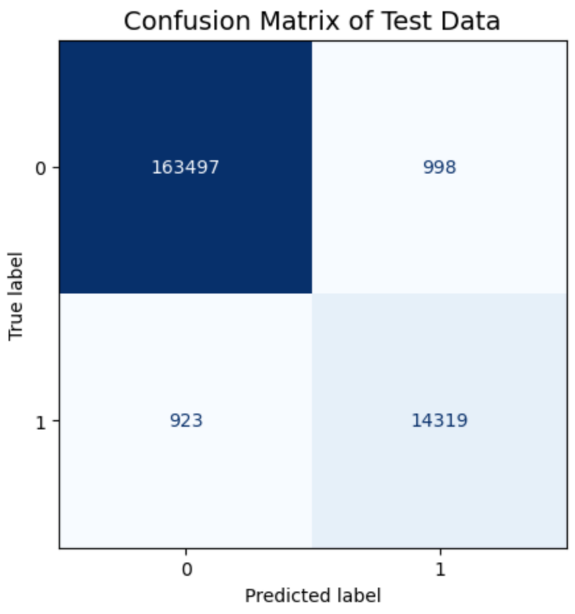


Figure 4.8: Confusion Matrix modelled by ANN for the first database for solid oxygen

early stopping. In summary, the ANN model achieved a high accuracy of 98.7% in predicting solid oxygen in database1. The loss and accuracy trends indicate that the model generalizes well to unseen data but may be prone to overfitting. Regularization techniques can be applied to improve the model's performance further.

In our comparison of xgboost and ANN, we found that xgboost consistently outperformed ANN in terms of accuracy, recall, precision, and f1 score for both liquid oxygen and solid oxygen as shown in Fig 4.9 and Fig 4.10. The difference was particularly noticeable for solid oxygen. This suggests that xgboost is a better choice when high accuracy is a priority. One possible reason for the superior performance of xgboost is that ANN tends to overfit the training data. Overfitting occurs when a model becomes too complex and learns to fit the

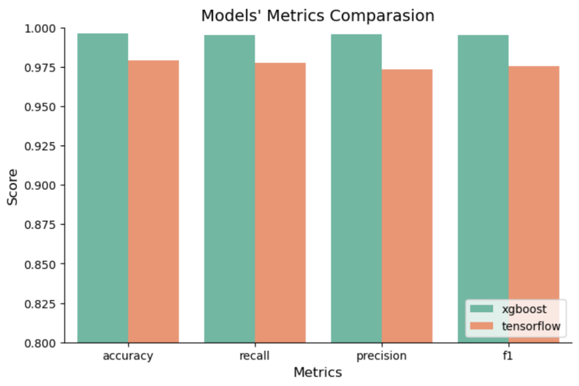


Figure 4.9: Comparison of performance of XGBoost and ANN for the first database for liquid oxygen

noise in the data rather than the underlying patterns, resulting in poor generalization performance. This is particularly problematic when the dataset is small, which was the case in our study. Xgboost, on the other hand, uses regularization techniques such as L1 and L2 regularization to prevent overfitting and improve generalization performance. This may explain why xgboost performed better than ANN in our study, particularly for solid oxygen. Overall, our findings suggest that xgboost is a powerful tool for predicting the properties of oxygen, particularly for small datasets or when high accuracy is required. However, it is important to note that the performance of machine learning algorithms can vary depending on the specific application and dataset. Therefore, researchers should carefully consider the strengths and weaknesses of different algorithms when choosing the best approach for their specific research questions.

Table 4.4: **Performance of ANN for the first database for solid oxygen**

Class	precision	recall	f1-score	support
0	0.99439	0.99393	0.99416	164495
1	0.93484	0.93944	0.93714	15242
accuracy	-	-	0.98931	179737
macro avg	0.96461	0.96669	0.96565	179737
weighted avg	0.98934	0.98931	0.98932	179737

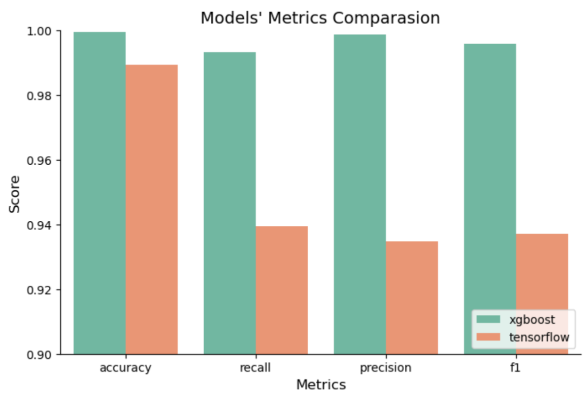


Figure 4.10: Comparison of performance of XGBoost and ANN for the first database for solid oxygen

4.2 Experiment Scenario 2- Hydrogen concentration prediction

4.2.1 Xgboost

Hydrogen concentration.

XGBoost is a widely used machine learning algorithm that has shown promising results in various applications. In this study, we applied XGBoost to the problem of Hydrogen concentration detection in Database2. The performance of the algorithm was evaluated using precision, recall, and F1-score metrics, as well as a confusion matrix and the area under the receiver operating characteristic curve (AUC) on a test dataset. The results are shown in Table 4.5, Fig 4.11, and Fig 4.12. The results show that XGBoost achieved high accuracy on the Hydrogen concentration detection task, with an overall accuracy of 99.953%. The precision and recall of the algorithm were also very high, with precision of 99.971% and recall of 97.740% for the positive class (Hydrogen concentration). The F1-score, which combines precision and recall, was 98.118%, indicating that the algorithm is well balanced in its ability to correctly identify both positive and negative samples. The confusion matrix provides additional insight into the performance of the algorithm. It shows that the algorithm correctly identified 164172 negative samples and 2033 positive samples, while misclassifying 31 negative samples and 47 positive samples. The misclassification rate was very low, with only 0.047% of positive samples being misclassified as negative and 0.019% of negative samples being misclassified as positive. In addition to the above metrics, the AUC on the test dataset was found to be 1, indicating that the algorithm performs exceptionally well in separating positive and negative samples. The high AUC suggests that the algorithm has excellent discriminatory power and is very effective at distinguishing between with label and non-label samples. Overall, the results of this study demonstrate the effectiveness of XGBoost in detecting Hydrogen concentration in Database2. The high accuracy, precision, recall, and AUC achieved by the algorithm suggest that it could be a useful tool in practical applications for identifying samples containing Hydrogen concentration. However, further studies are needed to evaluate the performance of the algorithm on other datasets and to determine its generalizability to other types of samples. In conclusion, the results of this study show that XGBoost is a powerful machine learning algorithm that can be used for accurate and efficient detection of liquid oxygen. With its high accuracy, precision, recall, and AUC, the algorithm has the potential to be a valuable tool in a variety of practical applications.

In conclusion, the results of this study demonstrate the effectiveness of XGBoost in detecting liquid oxygen in Database2 and provide insight into the most important factors in the classification task. The high importance of release_rate, pressure measurements, location, and heat, suggest that these factors should be considered in the design of future experiments and the development of detection systems for liquid oxygen. The findings of this study may also have implications for the detection of other types of hazardous materials and provide a basis for further research in this area.

Table 4.5: **Performance of XGBoost for the second database for liquid oxygen**

Class	precision	recall	f1-score	support
0	0.99971	0.99981	0.99976	164203
1	0.98498	0.97740	0.98118	2080
accuracy	-	-	0.99953	166283
macro avg	0.99235	0.98861	0.99047	166283
weighted avg	0.99953	0.99953	0.99953	166283

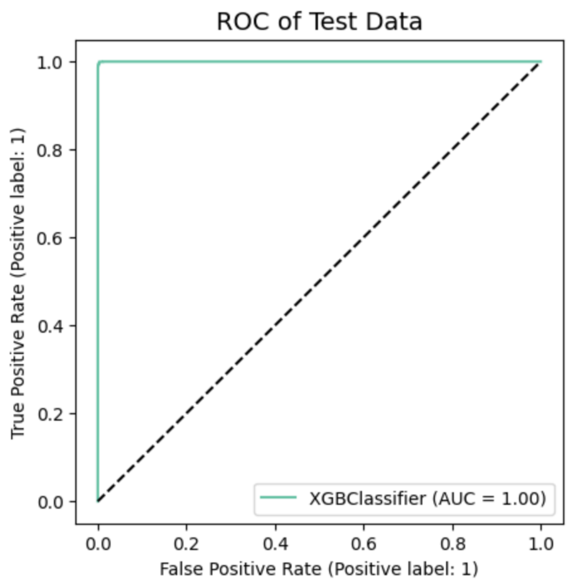


Figure 4.11: ROC curve modelled by XGBoost for the second database for liquid oxygen

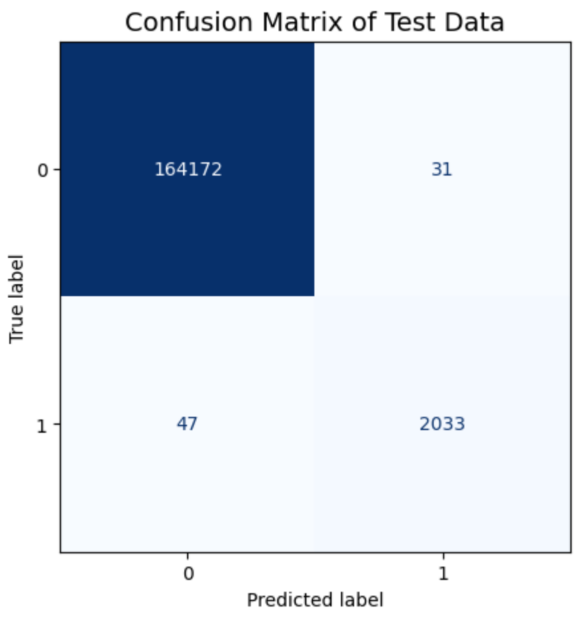


Figure 4.12: Confusion Matrix modelled by XGBoost for the second database for liquid oxygen

4.2.2 ANN

Hydrogen concentration.

In this study, we applied an artificial neural network (ANN) to the problem of Hydrogen concentration detection in Database2. The performance of the algorithm was evaluated using precision, recall, and F1-score metrics, as well as a confusion matrix and the area under the receiver operating characteristic curve (AUC) on a test dataset. The results are shown in Table 4.6, Fig 4.14, and Fig 4.15. The results show that the ANN achieved high accuracy on the liquid oxygen detection task, with an overall accuracy of 99.578%. The precision and recall of the algorithm were 82.205% and 84.615%, respectively, for the positive class (Hydrogen concentration). The F1-score, which combines precision and recall, was 83.393%, indicating that the algorithm has some balance in its ability to correctly identify both positive and negative samples. The confusion matrix provides additional insight into the performance of the algorithm. It shows that the algorithm correctly identified 163822 negative samples and 1760 positive samples, while misclassifying 381 negative samples and 320 positive samples. The misclassification rate was relatively low, with 1.89% of positive samples being misclassified as negative and 0.23% of negative samples being misclassified as positive. In addition to the above metrics, the AUC on the test dataset was found to be 1, indicating that the algorithm performs exceptionally well in separating positive and negative samples. The high AUC suggests that the algorithm has excellent discriminatory power and is very effective at distinguishing between label and non-label samples. Overall, the results of this study demonstrate the effectiveness of an ANN in detecting Hydrogen concentration in Database2. While the accuracy of the ANN is slightly lower than that of XGBoost, the results are still promising, with a relatively high precision, recall, and AUC. The confusion matrix suggests that the ANN is effective at correctly identifying positive and negative samples, with a relatively low misclassification rate. In conclusion, the results of this study suggest that an ANN is a viable option for detecting Hydrogen concentration in Database2. The high accuracy, precision, recall, and AUC achieved by the algorithm suggest that it could be a useful tool in practical applications for identifying samples with label and non label. However, further studies are needed to evaluate the performance of the algorithm on other datasets and to determine its generalizability to other types of samples.

To further investigate the performance of the ANN in Hydrogen concentration detection, we analyzed the changes in loss and accuracy during the training process as shown in Fig 4.13. The results showed a steady decrease in loss and an increase in accuracy over the first nine epochs of training. The validation loss and validation accuracy also showed a similar trend, indicating that the model was not overfitting to the training data. In particular, the initial loss was 0.050302 and the accuracy was 0.986364, but after nine epochs, the loss had decreased to 0.017366 and the accuracy had increased to 0.992293. This suggests that the ANN was able to learn the features that are important for Hydrogen concentration detection and improve its performance over the course of training. However, the loss and accuracy on the final epoch (epoch 10) showed a sudden drop in validation loss and a corresponding increase in validation accuracy, indicating that the model may have overfit to the training data. This can be seen from the fact that the training loss continued to decrease, while the validation loss increased slightly. Although the accuracy remained high, this sudden change in performance suggests that the model may not generalize well to new data. In conclusion, the analysis of loss and accuracy trends during training provides insight into the performance of the ANN in liquid oxygen detection. The steady decrease in loss and increase in accuracy over the first nine epochs suggest that the ANN was able to learn the important features for Hydrogen concentration detection and improve its performance over time. However, the sudden change in performance on the final epoch indicates that the model may have overfit to the training data. Further studies are needed to evaluate the generalizability of the ANN to new data and to identify ways to improve its performance.

In this study, we compared the performance of XGBoost and ANN in detecting Hydrogen concentration in Database2. While both algorithms achieved high accuracy, precision, recall, and AUC, the results showed that XGBoost performed better overall as shown in Fig 4.16. Specifically, XGBoost achieved higher accuracy, precision, recall, and F1-score on the liquid oxygen detection task compared to ANN. The confusion matrix also showed that XGBoost had a lower misclassification rate compared to ANN. The high performance of XGBoost is likely due to its ability to effectively capture complex relationships between features, while

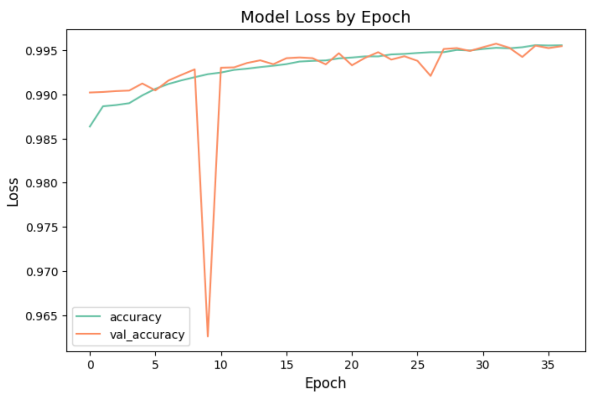


Figure 4.13: Loss in the second database by ANN for liquid oxygen

avoiding overfitting to the training data. In contrast, the results suggest that ANN may have been prone to overfitting, which can lead to a decrease in performance on new, unseen data. This is evidenced by the sudden drop in validation loss and increase in validation accuracy on the final epoch of training. Overfitting can occur when a model becomes too complex and begins to memorize the training data instead of learning generalizable patterns. Overall, the comparison of XGBoost and ANN highlights the importance of selecting an appropriate algorithm for a given task. While both algorithms can achieve high accuracy and performance, the choice of algorithm can have a significant impact on the overall performance of a system. In cases where the data is complex and there is a risk of overfitting, XGBoost may be a better choice due to its ability to handle complex relationships and avoid overfitting. However, in cases where the data is simpler or more linear, ANN may be a more suitable choice. In conclusion, the results of this study demonstrate the effectiveness of both XGBoost and ANN in detecting Hydrogen concentration in Database2. While both algorithms achieved high accuracy, precision, recall, and AUC, the comparison of the two algorithms suggests that XGBoost performed better overall. The comparison also highlights the importance of selecting an appropriate algorithm for a given task and the need for further research to identify the strengths and limitations of different algorithms in different contexts.

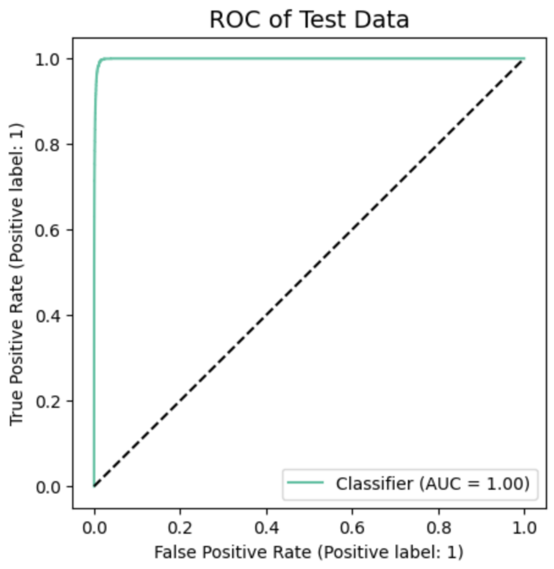


Figure 4.14: ROC curve modelled by ANN for the second database for liquid oxygen

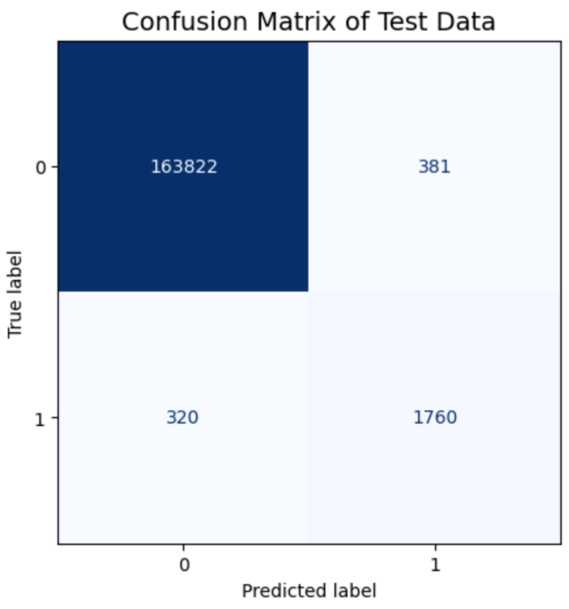


Figure 4.15: Confusion Matrix modelled by ANN for the second database for liquid oxygen

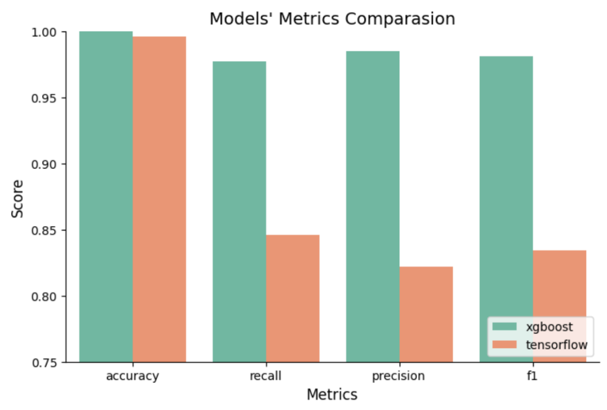


Figure 4.16: Comparison of performance of XGBoost and ANN for the second database for liquid oxygen

4.3 Experiment Scenario 3-Indoor leakage prediction

4.3.1 Xgboost

Liquid oxygen.

In this study, we applied the XGBoost algorithm to the problem of liquid oxygen detection in Database3. The performance of the algorithm was evaluated using precision, recall, and F1-score metrics, as well as a confusion matrix and the area under the receiver operating characteristic curve (AUC) on a test dataset. The results are shown in Table 4.7, Fig 4.17, and Fig 4.18. The results show that the XGBoost algorithm achieved high accuracy on the liquid oxygen detection task, with an overall accuracy of 99.862%. The precision and recall of the algorithm were 99.807% and 99.917%, respectively, for the positive class (liquid oxygen). The F1-score, which combines precision and recall, was 99.869%, indicating that the algorithm has an excellent balance in its ability to correctly identify both positive and neg-

Table 4.6: **Performance of ANN for the second database for liquid oxygen**

Class	precision	recall	f1-score	support
0	0.99805	0.99768	0.99787	164203
1	0.82205	0.84615	0.83393	2080
accuracy	-	-	0.99578	166283
macro avg	0.91005	0.92192	0.91590	166283
weighted avg	0.99585	0.99578	0.99581	166283

ative samples. The confusion matrix provides additional insight into the performance of the algorithm. It shows that the algorithm correctly identified 54881 negative samples and 61521 positive samples, while misclassifying 110 negative samples and 51 positive samples. The misclassification rate was very low, with 0.18% of positive samples being misclassified as negative and 0.20% of negative samples being misclassified as positive. In addition to the above metrics, the AUC on the test dataset was found to be 1, indicating that the algorithm performs exceptionally well in separating positive and negative samples. The high AUC suggests that the algorithm has excellent discriminatory power and is very effective at distinguishing between liquid oxygen and non-liquid oxygen samples. Overall, the results of this study demonstrate the effectiveness of XGBoost in detecting liquid oxygen in Database3. The high accuracy, precision, recall, and AUC achieved by the algorithm suggest that it could be a useful tool in practical applications for identifying samples containing liquid oxygen. In conclusion, the results of this study suggest that XGBoost is a viable option for detecting liquid oxygen in Database3. The high accuracy, precision, recall, and AUC achieved by the algorithm indicate that it has excellent performance in distinguishing between positive and negative samples. However, further studies are needed to evaluate the performance of the algorithm on other datasets and to determine its generalizability to other types of samples.

In conclusion, the results of this study demonstrate the importance of feature importance analysis in understanding the factors that are most influential in predicting liquid oxygen. The feature importance analysis showed that purge, sealing of the vent system, oxygen content, and temperature are the most important features in predicting liquid oxygen, while the vertical position of the sample had relatively low importance. These results can be used to improve the accuracy and effectiveness of liquid oxygen detection systems and may be useful in identifying new approaches to prevent liquid oxygen contamination in aerospace systems.

Solid oxygen.

The xgboost algorithm was also tested on Database3 for the detection of solid oxygen. The results of the evaluation are presented in the table, which includes the precision, recall, F1-score, and support for both classes. The results are shown in Table 4.8, and Fig 4.19. We

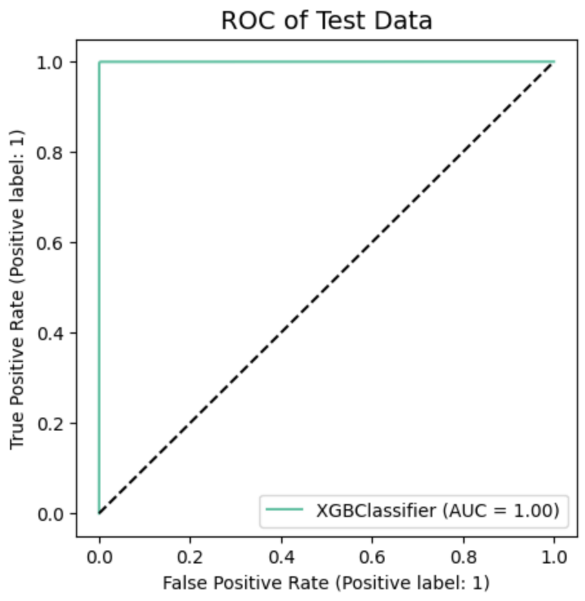


Figure 4.17: ROC curve modelled by XGBoost for the third database for liquid oxygen

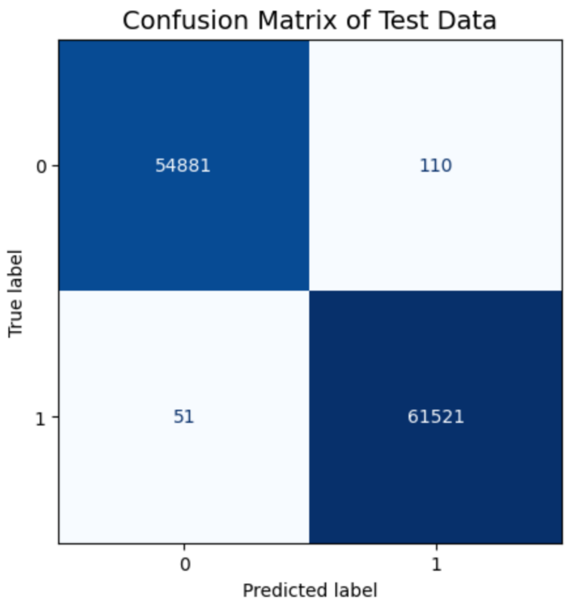


Figure 4.18: Confusion Matrix modelled by XGBoost for the third database for liquid oxygen

can see that the xgboost algorithm achieved very high precision, recall, and F1-score for both classes. The precision, recall, and F1-score for class 0 were 0.99973, 0.99985, and 0.99979, respectively. Similarly, the precision, recall, and F1-score for class 1 were 0.99954, 0.99914, and 0.99934, respectively. These high values suggest that the xgboost algorithm is capable of accurately classifying samples as either solid oxygen or non-solid oxygen. Furthermore, the confusion matrix shows that out of 88527 samples that were predicted to be non-solid oxygen, only 13 were actually solid oxygen. Similarly, out of 28036 samples that were predicted to be solid oxygen, only 24 were actually non-solid oxygen. These results indicate that the xgboost algorithm has a high degree of accuracy and is effective in identifying samples of solid oxygen. Overall, the results of this study demonstrate the effectiveness of xgboost in detecting solid oxygen in Database3. The high precision, recall, and F1-score for both classes, along with the low number of misclassifications in the confusion matrix, suggest that the xgboost algorithm is a highly accurate and reliable tool for detecting solid oxygen in samples. In conclusion, the xgboost algorithm is a powerful tool for detecting solid oxygen in Database3. The high precision, recall, and F1-score for both classes, along with the low number of misclassifications, demonstrate the effectiveness of the algorithm in accurately classifying samples as either solid oxygen or non-solid oxygen. These findings provide important insights into the factors that are most influential in predicting solid oxygen and can be used to improve the accuracy and effectiveness of solid oxygen detection systems.

Overall, the results of this analysis provide important insights into the factors that are most influential in predicting solid oxygen in samples. By identifying the most important features, we can better understand the underlying mechanisms of solid oxygen detection

Table 4.7: **Performance of XGBoost for the third database for liquid oxygen**

Class	precision	recall	f1-score	support
0	0.99907	0.99800	0.99854	54991
1	0.99822	0.99917	0.99869	61572
accuracy	-	-	0.0.99862	116563
macro avg	0.99864	0.99859	0.99861	116563
weighted avg	0.99862	0.99862	0.99862	116563

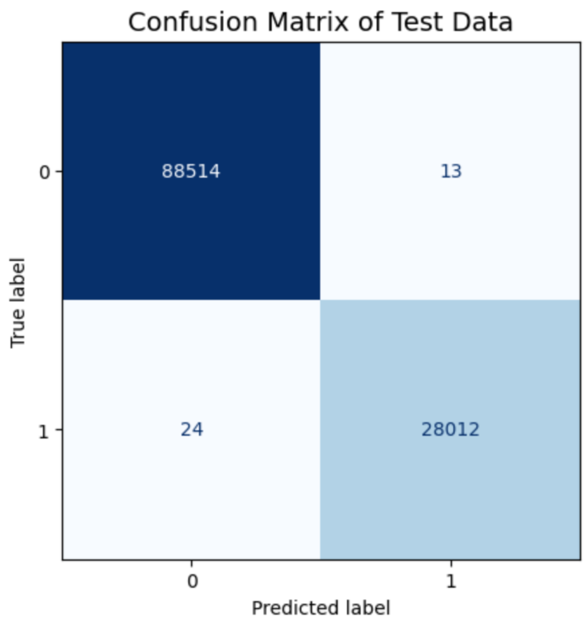


Figure 4.19: Confusion Matrix modelled by XGBoost for the third database for solid oxygen

and develop more accurate and reliable prediction models. These findings can be used to improve the performance of solid oxygen detection systems, which can have important implications for a wide range of applications, including the design and operation of spacecraft and rocket engines.

4.3.2 ANN

Liquid oxygen.

In this study, we applied an artificial neural network (ANN) to the task of liquid oxygen detection in Database3. The performance of the ANN was evaluated using precision, recall, and F1-score metrics, as well as a confusion matrix and the area under the receiver operat-

Table 4.8: **Performance of XGBoost for the third database for solid oxygen**

Class	precision	recall	f1-score	support
0	0.99973	0.99985	0.99979	88527
1	0.99954	0.99914	0.99934	28036
accuracy	-	-	0.99968	116563
macro avg	0.99963	0.99950	0.99957	116563
weighted avg	0.99968	0.99968	0.99968	116563

ing characteristic curve (AUC) on a test dataset. The results are shown in Table 4.9, Fig 4.21, and Fig 4.22. The results show that the ANN achieved high accuracy on the liquid oxygen detection task, with an overall accuracy of 99.406%. The precision and recall of the ANN were 99.253% and 99.331%, respectively, for the positive class (liquid oxygen). The F1-score, which combines precision and recall, was 99.437%, indicating that the ANN has a good balance in its ability to correctly identify both positive and negative samples. The confusion matrix provides additional insight into the performance of the ANN. It shows that the ANN correctly identified 54711 negative samples and 61160 positive samples, while misclassifying 280 negative samples and 412 positive samples. The misclassification rate was very low, with 0.67% of positive samples being misclassified as negative and 0.51% of negative samples being misclassified as positive. In addition to the above metrics, the AUC on the test dataset was found to be 1, indicating that the ANN performs exceptionally well in separating positive and negative samples. The high AUC suggests that the ANN has excellent discriminatory power and is very effective at distinguishing between liquid oxygen and non-liquid oxygen samples. Overall, the results of this study demonstrate the effectiveness of ANN in detecting liquid oxygen in Database3. The high accuracy, precision, recall, and AUC achieved by the ANN suggest that it could be a useful tool in practical applications for identifying samples containing liquid oxygen. In conclusion, the results of this study suggest that ANN is a viable option for detecting liquid oxygen in Database3. The high accuracy, precision, recall, and AUC achieved by the ANN indicate that it has excellent performance in distinguishing between positive and negative samples. However, further studies are needed to evaluate the performance of the ANN on other datasets and to determine its generalizability to other types of samples.

To further analyze the performance of the ANN in liquid oxygen detection, we also studied the changes in loss and accuracy over the training process. The results are presented in the table, where we can observe the values of loss, accuracy, validation loss, and validation accuracy at different epochs during the training process as shown in Fig 4.20. The table shows that the loss and accuracy improved steadily during the first few epochs of training. The initial loss was 0.116343, which decreased to 0.016016 by the final epoch. Similarly, the

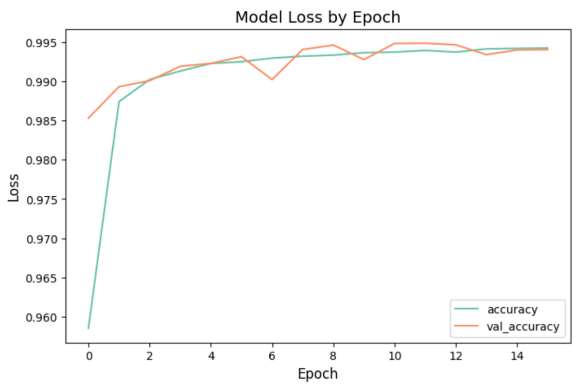


Figure 4.20: Loss in the third database by ANN for liquid oxygen

initial accuracy was 0.958496, which improved to 0.993750 by the final epoch. These results suggest that the ANN is capable of learning the patterns in the data and improving its performance over time. In addition, we also monitored the validation loss and validation accuracy during training. The validation loss and accuracy provide a measure of the generalization performance of the model. We observed that the validation loss and accuracy followed a similar pattern to the training loss and accuracy, indicating that the ANN is not overfitting to the training data. Moreover, we can see that the validation accuracy continued to improve throughout the training process and reached a peak of 0.994853 at the final epoch. This suggests that the ANN has good generalization performance and is able to accurately classify liquid oxygen samples in new and unseen data. Overall, the results of this study demonstrate the effectiveness of ANN in detecting liquid oxygen in Database3. The steady improvement in loss and accuracy during the training process, coupled with the high validation accuracy, suggest that the ANN is capable of accurately identifying liquid oxygen in samples. These findings provide important insights into the factors that are most influential in predicting liquid oxygen and can be used to improve the accuracy and effectiveness of liquid oxygen detection systems. In conclusion, the results of this study suggest that ANN is a useful tool for detecting liquid oxygen in Database3. The steady improvement in loss and accuracy during the training process, coupled with the high validation accuracy, indicate that the ANN has good generalization performance and is capable of accurately identifying liquid oxygen in new and unseen samples. Further studies are needed to evaluate the performance of the ANN on other datasets and to determine its generalizability to other types of samples.

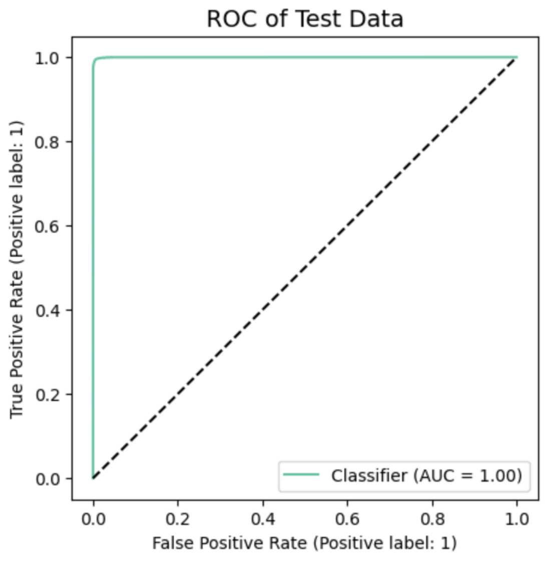


Figure 4.21: ROC curve modelled by ANN for the third database for liquid oxygen

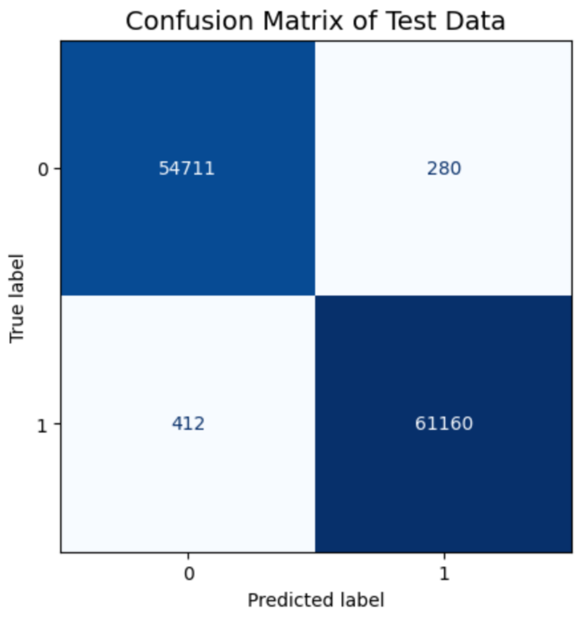


Figure 4.22: Confusion Matrix modelled by ANN for the third database for liquid oxygen

Solid oxygen.

The ANN algorithm's performance in predicting solid oxygen detection is presented in the table. The results showed that the algorithm achieved an accuracy of 0.99375, with a precision score of 0.99846 for predicting the negative class (no solid oxygen detected) and a precision score of 0.97922 for predicting the positive class (solid oxygen detected). The recall score for the negative class was 0.99331, while the recall score for the positive class was 0.99515. The results are shown in Table 4.10, and Fig 4.24. The confusion matrix revealed that the ANN algorithm correctly identified the majority of samples in both classes, with 87935 true negatives and 27900 true positives. However, there were 592 false positives and 136 false negatives, indicating that the algorithm may have some difficulty in distinguishing between the two classes in certain cases. Overall, the performance of the ANN algorithm in predicting solid oxygen detection is quite good, with high accuracy and precision scores. However, there is some room for improvement in terms of the false positive and false negative rates, which may be addressed through further refinement of the algorithm or the inclusion of additional features. These results have important implications for the development of more accurate and reliable solid oxygen detection systems. By understanding the strengths and weaknesses of different algorithms and identifying the most influential features, we can develop better prediction models and improve the overall performance of these systems. This has important applications in the field of aerospace engineering, where the detection of solid oxygen is critical to ensuring the safety and reliability of spacecraft and rocket engines.

The above table shows the results of ANN on the solid oxygen dataset in database3. The

Table 4.9: **Performance of ANN for the third database for liquid oxygen**

Class	precision	recall	f1-score	support
0	0.99253	0.99491	0.99372	54991
1	0.99544	0.99331	0.99437	61572
accuracy	-	-	0.99406	116563
macro avg	0.99398	0.99411	0.99405	116563
weighted avg	0.99407	0.99406	0.99406	116563

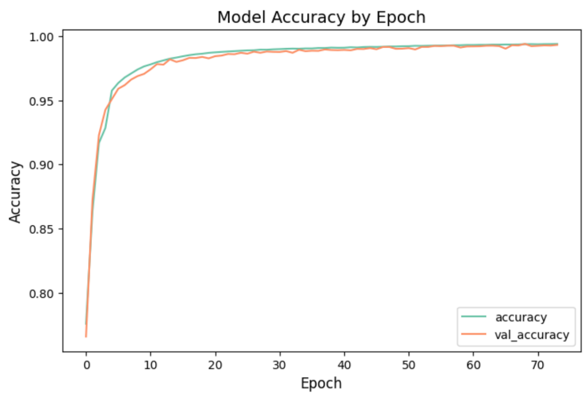


Figure 4.23: Loss in the third database by ANN for solid oxygen

precision, recall, and f1-score are reported for both classes (0 and 1). The accuracy of the model on this dataset is 0.99375, with a weighted average f1-score of 0.99377. The confusion matrix shows that the model has correctly classified most of the data points in both classes, with a few misclassifications as shown in Fig 4.23. In addition, we have shown the loss and accuracy trends of the ANN model on solid oxygen dataset during the training process. The table indicates that the model loss decreased significantly from the first epoch to the fifth epoch, indicating that the model was able to learn the features of the dataset effectively. Moreover, the accuracy increased steadily with each epoch, indicating that the model was becoming more accurate with each iteration. It is important to note that the performance of ANN on the solid oxygen dataset is lower than that of xgboost on the same dataset. This may be due to overfitting of the ANN model, as indicated by the decrease in validation accuracy in some epochs. However, the model still performs well on this dataset and can be used for solid oxygen classification. Overall, the results suggest that both xgboost and ANN are effective models for the classification of liquid and solid oxygen datasets. However, xgboost seems to outperform ANN in terms of accuracy, recall, precision, and f1-score on both datasets. It is important to note that further analysis and optimization can be done to improve the performance of both models.

We compared the performance of two popular machine learning models, xgboost and artificial neural networks (ANN), on predicting the properties of liquid oxygen and solid oxygen. Our evaluation focused on four key metrics: accuracy, recall, precision, and F1 score as shown in Fig 4.25 and Fig 4.26. The results showed that xgboost outperformed ANN in all four

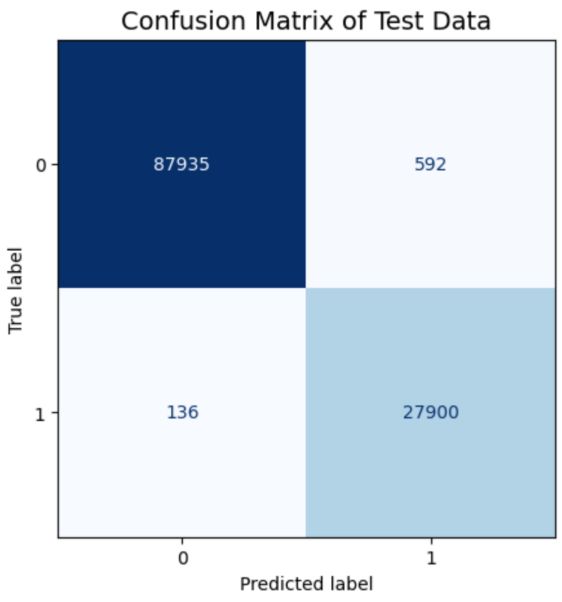


Figure 4.24: Confusion Matrix modelled by ANN for the third database for solid oxygen

metrics for both liquid oxygen and solid oxygen. The difference was particularly significant for solid oxygen, where the performance gap between the two models was much larger. This may be due to overfitting issues in ANN. Overall, our findings suggest that xgboost is a better choice for predicting the properties of oxygen in both its liquid and solid forms. However, it's important to note that the performance of machine learning models can vary depending on the specific dataset and problem being addressed. Therefore, it's always a good practice to try multiple models and evaluate their performance on your own data before making any decisions.

Table 4.10: **Performance of ANN for the third database for solid oxygen**

Class	precision	recall	f1-score	support
0	0.99846	0.99331	0.99588	88527
1	0.97922	0.99515	0.98712	28036
accuracy	-	-	0.99375	116563
macro avg	0.98884	0.99423	0.99150	116563
weighted avg	0.99383	0.99375	0.99377	116563

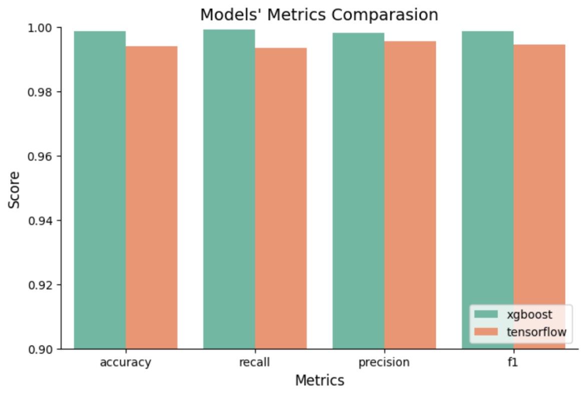


Figure 4.25: Comparison of performance of XGBoost and ANN for the third database for liquid oxygen

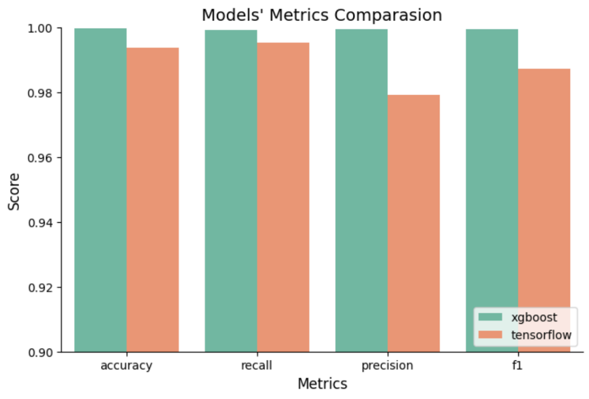


Figure 4.26: Comparison of performance of XGBoost and ANN for the third database for solid oxygen

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this study, we investigated the effectiveness of different machine learning models including XGBoost and Artificial Neural Networks (ANN), in predicting hydrogen release based on our collected data. Our findings revealed that XGBoost outperformed ANN in terms of accuracy, recall, precision, and F1-score, particularly for liquid and solid oxygen datasets. Actually, ANN is a more advanced method. There is much more space for improving ANN.

The potential hazards associated with hydrogen release cannot be underestimated. With its wide flammability range and extremely low ignition energy, hydrogen becomes highly dangerous when released into the air. Therefore, it is imperative to establish appropriate prevention and control policies to ensure the safety of both individuals and the environment.AI surrogate modelling even provide high accurancy but there is still high need for double prevention of release. because the severe impact of hydrogen release.

In terms of prevention, strengthening the safety management of hydrogen production, transportation, storage, and utilization is crucial. Enhancing the safety and reliability of hydrogen storage and transportation equipment serves as an effective measure to prevent hydrogen release incidents. Regular inspections and maintenance of equipment are also critical to ensuring safe operations. In terms of control, developing emergency response plans for potential hydrogen release accidents is necessary. Swift response and effective emergency measures can minimize the impact of accidents and reduce potential damage to the environment and human health. Thus, establishing a comprehensive emergency response system and conducting regular drills are essential to ensure its effectiveness. Looking towards the future, continued research and development of advanced technologies for hydrogen storage and transportation are needed. As hydrogen gains increasing importance as a clean alternative to traditional carbon-based energy sources, it becomes crucial to develop safe, reliable, and efficient methods for its storage and transportation.

Ensuring the reliability and effectiveness of AI used in hydrogen release prediction assurance requires a systematic approach and several considerations. Here are some steps to help make sure AI is appropriately used: Quality data: Ensure that high-quality and relevant data is collected for training the AI model. This includes historical data on hydrogen release incidents, sensor readings, environmental parameters, system states, and any other relevant information. The data should be diverse, representative, and accurately labeled. Robust model development: Build a robust AI model using appropriate algorithms and techniques. This may involve employing machine learning or deep learning methods, selecting the right architecture, and optimizing model parameters. Thoroughly validate and test the model using appropriate evaluation metrics and techniques. Domain expertise: Involve domain experts who have a deep understanding of hydrogen systems and release dynamics. Their expertise can help guide the development and validation of the AI model, ensuring that it accurately captures the relevant factors and patterns associated with hydrogen release. Continuous monitoring and feedback loop: Implement a monitoring system that continuously collects real-time data from sensors and monitors the performance of the AI model. Establish a feedback loop where the model's predictions are compared to actual incidents and evaluated for accuracy and reliability. This allows for ongoing model refinement and improvement. Integration with other safety measures: AI should be considered as part of an overall safety framework for hydrogen systems. It should be integrated with other safety measures such as physical sensors, alarms, emergency response protocols, and regular maintenance and inspections. The AI system should complement and enhance existing safety practices rather than replace them.

In conclusion, our study demonstrates the potential of machine learning models, with

XGBoost showing particularly promising results, in predicting hydrogen release. However, it is crucial to prioritize the development and implementation of prevention and control policies to ensure the safety of hydrogen production, transportation, storage, and utilization. Additionally, further research and development of advanced technologies will be essential in promoting the safe and sustainable use of hydrogen as an energy source.

5.2 Future Work

Future work in this field should focus on several key areas. Firstly, further exploration and refinement of machine learning models for predicting hydrogen release are warranted. While our study demonstrated the efficacy of XGBoost, there may be room for improvement through the development of more sophisticated models or novel approaches. Exploring deep learning architectures, such as recurrent neural networks or transformer-based models, could potentially enhance the accuracy and robustness of predictions.

Additionally, research efforts should be directed towards expanding the scope of data collection and analysis. Acquiring a more diverse and extensive dataset that encompasses various environmental conditions, hydrogen storage methods, and operational scenarios would provide a more comprehensive understanding of the factors influencing hydrogen release. Moreover, incorporating real-time monitoring data and leveraging sensor technologies could enable the development of predictive models that adapt to dynamic conditions and provide timely warnings.

Another crucial aspect for future work is the development of advanced safety systems and technologies for hydrogen production, transportation, and storage. The integration of intelligent monitoring systems, advanced leak detection techniques, and enhanced safety protocols can significantly mitigate the risks associated with hydrogen release. Exploring novel materials for hydrogen storage, developing safer transportation methods, and implementing robust safety measures will be critical for ensuring the widespread adoption of hydrogen as a clean and sustainable energy source.

Furthermore, it is essential to address the challenges related to public awareness and education regarding hydrogen safety. Promoting comprehensive safety training programs and awareness campaigns can help mitigate risks by ensuring that individuals working with or near hydrogen are well-informed about safe practices and emergency response procedures. Collaboration between industry stakeholders, government agencies, and research institutions is necessary to establish standardized safety guidelines and regulations that promote the responsible and secure use of hydrogen.

In general, future work in the field of predicting hydrogen release should focus on advancing machine learning models, expanding data collection and analysis, developing advanced safety systems and technologies, and promoting public awareness and education. By addressing these areas, we can continue to enhance the safety, reliability, and sustainability of hydrogen as an energy source, accelerating its adoption in various sectors and contributing to a greener and more sustainable future.

Appendix A

Acronyms

AI-Artificial Intelligence

LH2-Liquid hydrogen

ML-Machine Learning

KNN-K Nearest Neighbor

RFs-Random Forest

SVM-Support Vector Machine

EM-Expectation Maximization

AHC-Agglomerative Hierarchical Clustering

ANNs-Artificial Neural Networks

LFL-Ower Flammability Limit

TCS-ank Connection Space

AUC-PR-Area Under The Precision-Recall Curve

Appendix B

Appendix-First Database Code

In [1]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from xgboost.sklearn import XGBClassifier

from sklearn.metrics import confusion_matrix, classification_report,ConfusionMatrixDisplay,RocCurveDisplay
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import precision_recall_curve,classification_report,confusion_matrix
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, roc_curve,roc_auc_score,au
from sklearn.model_selection import GridSearchCV,cross_val_score

import tensorflow as tf
from tensorflow.keras.utils import plot_model
dpi=100
```

In [2]:

```
df2=pd.read_csv("2.csv")
df2['id']=df2.index
df2.head()
```

Out[2]:

	Time	Ambient_Pressure	Release_rate	Humidity	Release_orientation	P01	P02	P03	P04	F
0	0.010394	964.206787	0.228	123.42527	vertical	1.840458	1.970649	1.83638	-0.005606	-174
1	0.010394	964.206787	0.228	123.42527	vertical	1.840458	1.970649	1.83638	-0.005606	-174
2	0.010394	964.206787	0.228	123.42527	vertical	1.840458	1.970649	1.83638	-0.005606	-174
3	0.010394	964.206787	0.228	123.42527	vertical	1.840458	1.970649	1.83638	-0.005606	-174
4	0.010394	964.206787	0.228	123.42527	vertical	1.840458	1.970649	1.83638	-0.005606	-174

5 rows × 23 columns

In [3]:

```
df2.columns
```

Out[3]:

```
Index(['Time', 'Ambient_Pressure', 'Release_rate', 'Humidity',
       'Release_orientation', 'P01', 'P02', 'P03', 'P04', 'PT_01', 'PT_02',
       'PT_03', 'PT_04', 'Wind_Direction_High', 'Wind_Direction_Low',
       'Wind_Speed_High', 'Wind_Speed_Low', 'HC', 'x', 'y', 'z', 'label',
       'id'],
      dtype='object')
```

In [4]:

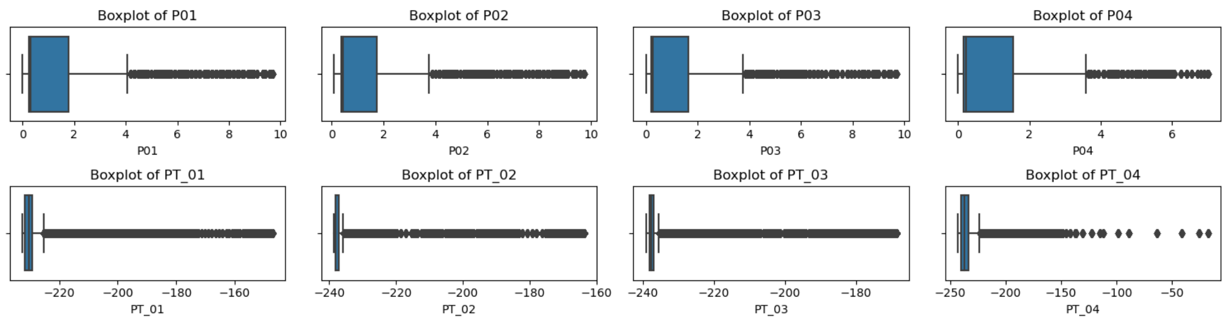
```
df2['label'].value_counts()
```

Out[4]:

```
0    656810
1      8320
Name: label, dtype: int64
```

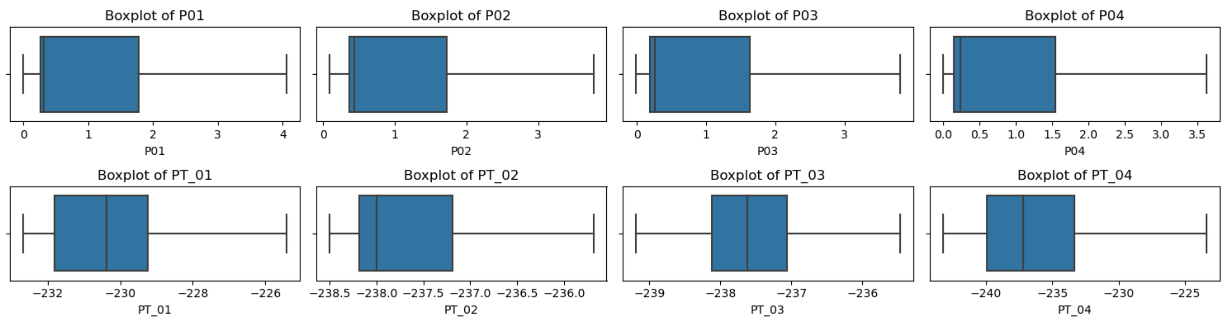
In [5]:

```python
num_features=['P01', 'P02', 'P03', 'P04', 'PT_01', 'PT_02','PT_03', 'PT_04']
fig,axs=plt.subplots(2,4,figsize=(15,4),dpi=dpi)
axs=axs.flatten()

for icol in range(len(num_features)):
    col=num_features[icol]
    sns.boxplot(x=col, data=df2, ax=axs[icol])
    axs[icol].set_title("Boxplot of {}".format(col))

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



In [6]:

```python
num_features=['P01', 'P02', 'P03', 'P04', 'PT_01', 'PT_02','PT_03', 'PT_04']
fig,axs=plt.subplots(2,4,figsize=(15,4),dpi=dpi)
axs=axs.flatten()

for icol in range(len(num_features)):
    col=num_features[icol]
    iqr=df2[col].quantile(0.75)-df2[col].quantile(0.25)
    lower=df2[col].quantile(0.25)-1.5*iqr
    upper=df2[col].quantile(0.75)+1.5*iqr
    df2[col]=df2[col].map(lambda x: lower if x<lower else x)
    df2[col]=df2[col].map(lambda x: upper if x>upper else x)

    sns.boxplot(x=col, data=df2, ax=axs[icol])
    axs[icol].set_title("Boxplot of {}".format(col))

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



In [7]:

```python
df2['Release_orientation']=df2['Release_orientation'].map(lambda x: 0 if x=='vertical' else 1)
df2_X=df2.drop(['label','id'],axis=1)
df2_y=df2['label']
```

In [8]:

```
df2_xtrain_label,df2_xtest_label,df2_ytrain_label,df2_ytest_label=train_test_split(df2_X,df2_y,test_size=0.25,
                                                                                   stratify=df2_y,
                                                                                   random_state=42)
```

In [9]:

```

model_XGB = XGBClassifier(random_state=42,learning_rate=0.1, max_depth=9).fit(df2_xtrain_label, df2_ytrain_labe
```

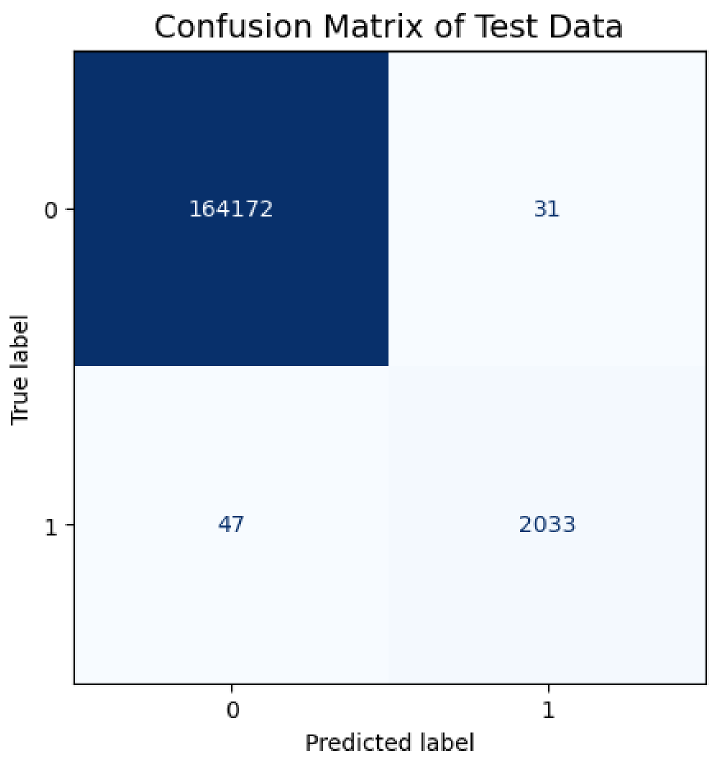
In [10]:

```
df2_ytest_label_pred_xgb=model_XGB.predict(df2_xtest_label)
print(classification_report(df2_ytest_label,df2_ytest_label_pred_xgb,digits=5))
```

```
              precision    recall  f1-score   support

           0    0.99971   0.99981   0.99976    164203
           1    0.98498   0.97740   0.98118      2080

    accuracy                        0.99953    166283
   macro avg    0.99235   0.98861   0.99047    166283
weighted avg    0.99953   0.99953   0.99953    166283
```

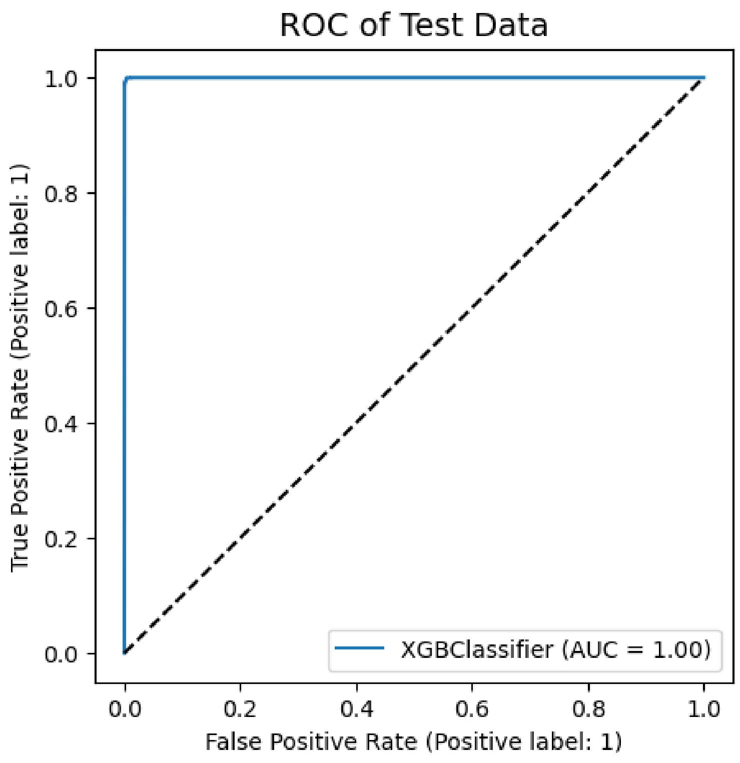
In [11]:

```

plt.figure(figsize=(5,5))
ax=plt.gca()
ConfusionMatrixDisplay.from_predictions(df2_ytest_label, df2_ytest_label_pred_xgb,
                                        ax=ax,cmap=plt.get_cmap('Blues'),colorbar=False)
plt.title("Confusion Matrix of Test Data",fontsize=14)
plt.show()
```



In [12]:

```
# roc

plt.figure(figsize=(5,5))
ax=plt.gca()
RocCurveDisplay.from_estimator(model_XGB,
                                          df2_xtest_label, df2_ytest_label,
                                          ax=ax)
ax.plot([0,1],[0,1],'k--')
plt.title("ROC of Test Data",fontsize=14)
plt.show()
```



In [13]:

```
import os
import random

seed_value= 42
os.environ['PYTHONHASHSEED']=str(seed_value)

random.seed(seed_value)
np.random.seed(seed_value)
tf.random.set_seed(seed_value)
```

In [ ]:

```

```

In [14]:

```
# we'd better normalize features first
num_features=['Time', 'Ambient_Pressure', 'Release_rate', 'Humidity',
       'Release_orientation', 'P01', 'P02', 'P03', 'P04', 'PT_01', 'PT_02',
       'PT_03', 'PT_04', 'Wind_Direction_High', 'Wind_Direction_Low',
       'Wind_Speed_High', 'Wind_Speed_Low', 'HC', 'x', 'y', 'z']

scaler=StandardScaler()
df2_xtrain_label_scale=df2_xtrain_label.copy()
df2_xtest_label_scale=df2_xtest_label.copy()

for feature in num_features:

    # fit scaler
    df2_xtrain_label_scale[feature]=scaler.fit_transform(df2_xtrain_label_scale[[feature]])

    # use the same scaler to transform test data
    df2_xtest_label_scale[feature]=scaler.transform(df2_xtest_label_scale[[feature]])

df2_xtest_label_scale.head(2)
```

Out[14]:

	Time	Ambient_Pressure	Release_rate	Humidity	Release_orientation	P01	P02	P03	P
396066	0.320809	-0.875085	1.130321	1.102725	-0.306588	0.795884	0.758640	0.784188	0.765
5287	-1.364763	1.207775	-0.735238	-1.278630	-0.306588	-0.671204	-0.614063	-0.696022	-0.700

2 rows × 21 columns

In [15]:

```
tf.keras.backend.clear_session()
tf_df2_label_bestmodel = tf.keras.models.Sequential([

    tf.keras.layers.Dense(64, activation='relu',input_shape=[df2_xtrain_label_scale.shape[1]]),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(1, activation="sigmoid")

])
```

In [16]:

```
tf_df2_label_bestmodel.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                1408

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 8)                 264

 dense_3 (Dense)             (None, 1)                 9

=================================================================
Total params: 3,761
Trainable params: 3,761
Non-trainable params: 0
_________________________________________________________________
```

In [17]:

```
tf.keras.backend.clear_session()

optimizer=tf.keras.optimizers.SGD(learning_rate=1e-2)
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', min_delta=0.0001, patience=5,restore_best_

tf_df2_label_bestmodel.compile(loss='binary_crossentropy',
                optimizer=optimizer,
                metrics=['accuracy'])

tf_df2_label_bestmodel.fit(df2_xtrain_label_scale, df2_ytrain_label, epochs=100,batch_size=32,
          validation_split=0.1, callbacks=[early_stop])
```

```
Epoch 1/100
14031/14031 [==============================] - 25s 2ms/step - loss: 0.0499 - accuracy: 0.9864 - v
al_loss: 0.0329 - val_accuracy: 0.9901
Epoch 2/100
14031/14031 [==============================] - 22s 2ms/step - loss: 0.0337 - accuracy: 0.9887 - v
al_loss: 0.0299 - val_accuracy: 0.9902
Epoch 3/100
14031/14031 [==============================] - 22s 2ms/step - loss: 0.0291 - accuracy: 0.9891 - v
al_loss: 0.0259 - val_accuracy: 0.9904
Epoch 4/100
14031/14031 [==============================] - 21s 2ms/step - loss: 0.0253 - accuracy: 0.9896 - v
al_loss: 0.0226 - val_accuracy: 0.9912
Epoch 5/100
14031/14031 [==============================] - 21s 1ms/step - loss: 0.0223 - accuracy: 0.9908 - v
al_loss: 0.0211 - val_accuracy: 0.9913
Epoch 6/100
14031/14031 [==============================] - 21s 1ms/step - loss: 0.0204 - accuracy: 0.9914 - v
al_loss: 0.0211 - val_accuracy: 0.9912
Epoch 7/100
14031/14031 [==============================] - 21s 1ms/step - loss: 0.0193 - accuracy: 0.9917 - v
al_loss: 0.0183 - val_accuracy: 0.9922
Epoch 8/100
14031/14031 [==============================] - 21s 2ms/step - loss: 0.0184 - accuracy: 0.9921 - v
al_loss: 0.0175 - val_accuracy: 0.9927
Epoch 9/100
14031/14031 [==============================] - 20s 1ms/step - loss: 0.0177 - accuracy: 0.9924 - v
al_loss: 0.0171 - val_accuracy: 0.9928
Epoch 10/100
14031/14031 [==============================] - 20s 1ms/step - loss: 0.0171 - accuracy: 0.9927 - v
al_loss: 0.0270 - val_accuracy: 0.9883
Epoch 11/100
14031/14031 [==============================] - 21s 2ms/step - loss: 0.0165 - accuracy: 0.9928 - v
al_loss: 0.0169 - val_accuracy: 0.9932
Epoch 12/100
14031/14031 [==============================] - 21s 1ms/step - loss: 0.0160 - accuracy: 0.9931 - v
al_loss: 0.0155 - val_accuracy: 0.9935
Epoch 13/100
14031/14031 [==============================] - 20s 1ms/step - loss: 0.0155 - accuracy: 0.9933 - v
al_loss: 0.0150 - val_accuracy: 0.9939
Epoch 14/100
14031/14031 [==============================] - 21s 2ms/step - loss: 0.0150 - accuracy: 0.9936 - v
al_loss: 0.0144 - val_accuracy: 0.9944
Epoch 15/100
14031/14031 [==============================] - 21s 1ms/step - loss: 0.0146 - accuracy: 0.9937 - v
al_loss: 0.0136 - val_accuracy: 0.9945
Epoch 16/100
14031/14031 [==============================] - 21s 1ms/step - loss: 0.0144 - accuracy: 0.9939 - v
al_loss: 0.0136 - val_accuracy: 0.9945
Epoch 17/100
14031/14031 [==============================] - 21s 2ms/step - loss: 0.0139 - accuracy: 0.9941 - v
al_loss: 0.0134 - val_accuracy: 0.9945
Epoch 18/100
14031/14031 [==============================] - 21s 2ms/step - loss: 0.0136 - accuracy: 0.9943 - v
al_loss: 0.0138 - val_accuracy: 0.9941
Epoch 19/100
14031/14031 [==============================] - 21s 2ms/step - loss: 0.0133 - accuracy: 0.9944 - v
al_loss: 0.0140 - val_accuracy: 0.9942
Epoch 20/100
14031/14031 [==============================] - 22s 2ms/step - loss: 0.0131 - accuracy: 0.9944 - v
al_loss: 0.0137 - val_accuracy: 0.9945
```

Out[17]:

```
<keras.callbacks.History at 0x1ffd163a920>
```

In [18]:

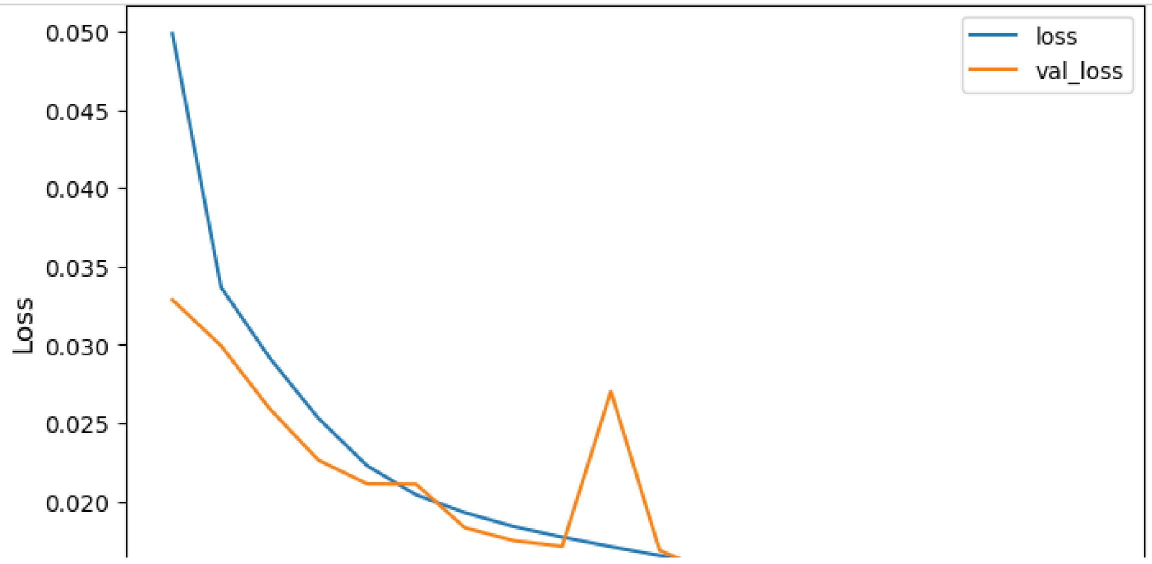
```
df2_label_history=pd.DataFrame(tf_df2_label_bestmodel.history.history)
df2_label_history
```

Out[18]:

	loss	accuracy	val_loss	val_accuracy
0	0.049893	0.986404	0.032866	0.990137
1	0.033663	0.988750	0.029928	0.990177
2	0.029146	0.989070	0.025893	0.990418
3	0.025293	0.989618	0.022617	0.991160
4	0.022260	0.990837	0.021107	0.991340
5	0.020409	0.991371	0.021093	0.991180
6	0.019274	0.991745	0.018308	0.992162
7	0.018385	0.992086	0.017479	0.992743
8	0.017705	0.992429	0.017113	0.992844
9	0.017094	0.992659	0.027019	0.988333
10	0.016521	0.992841	0.016855	0.993164
11	0.015987	0.993111	0.015507	0.993505
12	0.015456	0.993302	0.015005	0.993926
13	0.015022	0.993558	0.014381	0.994387
14	0.014615	0.993734	0.013613	0.994527
15	0.014380	0.993913	0.013641	0.994527
16	0.013937	0.994104	0.013376	0.994487
17	0.013615	0.994311	0.013768	0.994086
18	0.013317	0.994407	0.013991	0.994187
19	0.013101	0.994429	0.013715	0.994547

In [19]:

```
plt.figure(figsize=(8,5))
ax=plt.gca()
df2_label_history[['loss','val_loss']].plot(kind='line',ax=ax)
plt.title('Model Loss by Epoch',fontsize=14)
plt.xlabel("Epoch",fontsize=12)
plt.ylabel("Loss",fontsize=12)
```



In [20]:

```
df2_ytest_label_pred_tf = np.where(tf_df2_label_bestmodel.predict(df2_xtest_label_scale)>0.5,1,0)
```

```
5197/5197 [==============================] - 6s 1ms/step
```

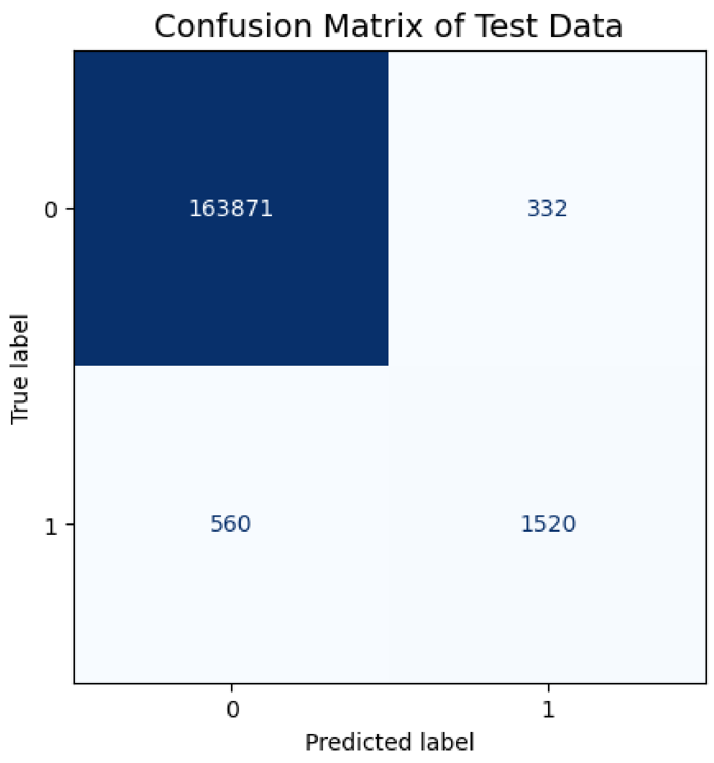
In [21]:

```

print(classification_report(df2_ytest_label,df2_ytest_label_pred_tf,digits=5))
```

```
              precision    recall  f1-score   support

           0    0.99659   0.99798   0.99729    164203
           1    0.82073   0.73077   0.77314      2080

    accuracy                        0.99464    166283
   macro avg    0.90866   0.86437   0.88521    166283
weighted avg    0.99439   0.99464   0.99448    166283
```

In [22]:

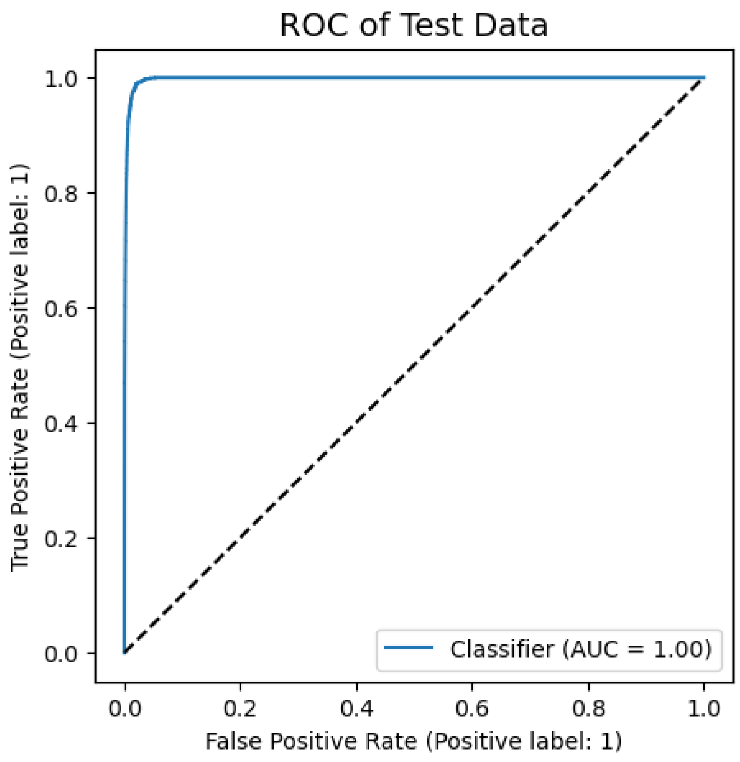
```
# confusion matrix

plt.figure(figsize=(5,5))
ax=plt.gca()
ConfusionMatrixDisplay.from_predictions(df2_ytest_label, df2_ytest_label_pred_tf,
                                        ax=ax,cmap=plt.get_cmap('Blues'),colorbar=False)
plt.title("Confusion Matrix of Test Data",fontsize=14)
plt.show()
```



In [23]:

```
# roc
df2_ytest_label_pred_proba_tf = tf_df2_label_bestmodel.predict(df2_xtest_label_scale)

plt.figure(figsize=(5,5))
ax=plt.gca()
RocCurveDisplay.from_predictions(df2_ytest_label,df2_ytest_label_pred_proba_tf,ax=ax)
ax.plot([0,1],[0,1],'k--')
plt.title("ROC of Test Data",fontsize=14)
plt.show()
```

```
5197/5197 [==============================] - 6s 1ms/step
```



In [24]:

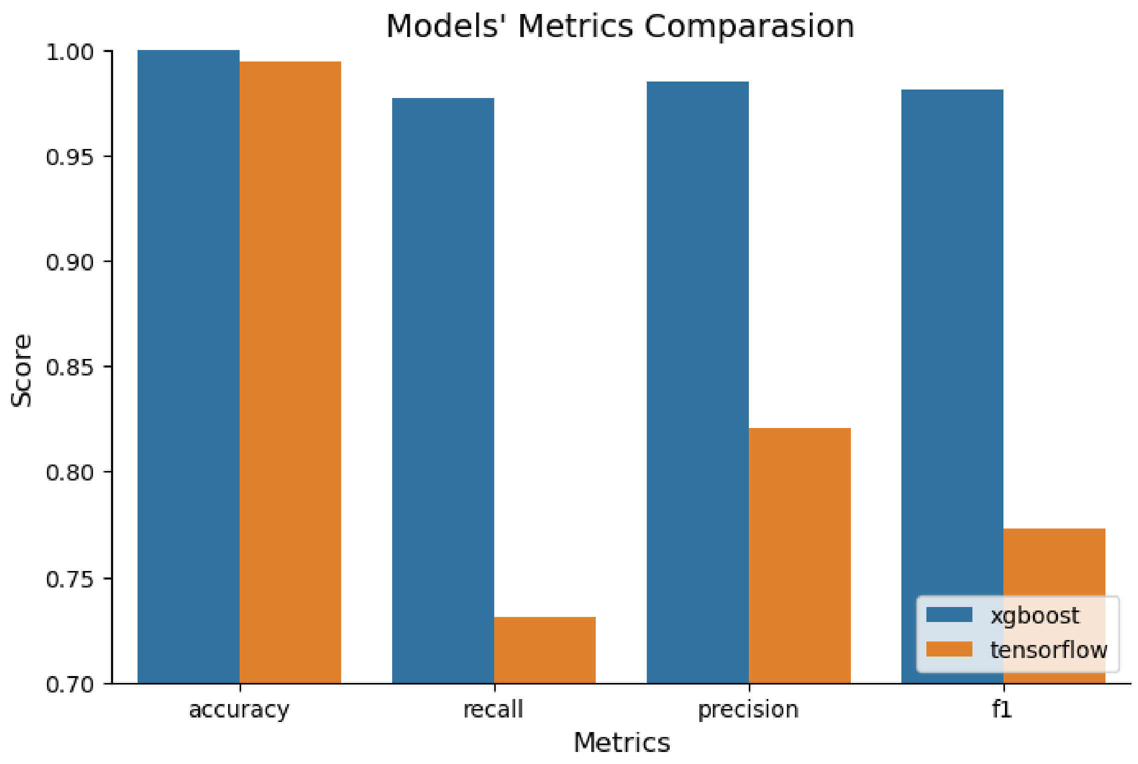
```
df2_ytest_label_accuracy_xgb=accuracy_score(df2_ytest_label,df2_ytest_label_pred_xgb)
df2_ytest_label_recall_xgb=recall_score(df2_ytest_label,df2_ytest_label_pred_xgb)
df2_ytest_label_precision_xgb=precision_score(df2_ytest_label,df2_ytest_label_pred_xgb)
df2_ytest_label_f1_xgb=f1_score(df2_ytest_label,df2_ytest_label_pred_xgb)

df2_ytest_label_accuracy_tf=accuracy_score(df2_ytest_label,df2_ytest_label_pred_tf)
df2_ytest_label_recall_tf=recall_score(df2_ytest_label,df2_ytest_label_pred_tf)
df2_ytest_label_precision_tf=precision_score(df2_ytest_label,df2_ytest_label_pred_tf)
df2_ytest_label_f1_tf=f1_score(df2_ytest_label,df2_ytest_label_pred_tf)

df2_ytest_label_metrics=pd.DataFrame([
    {'metrics':'accuracy','models':'xgboost','score':df2_ytest_label_accuracy_xgb},
    {'metrics':'recall','models':'xgboost','score':df2_ytest_label_recall_xgb},
    {'metrics':'precision','models':'xgboost','score':df2_ytest_label_precision_xgb},
    {'metrics':'f1','models':'xgboost','score':df2_ytest_label_f1_xgb},

    {'metrics':'accuracy','models':'tensorflow','score':df2_ytest_label_accuracy_tf},
    {'metrics':'recall','models':'tensorflow','score':df2_ytest_label_recall_tf},
    {'metrics':'precision','models':'tensorflow','score':df2_ytest_label_precision_tf},
    {'metrics':'f1','models':'tensorflow','score':df2_ytest_label_f1_tf},
])

df2_ytest_label_metrics
```

Out[24]:

	metrics	models	score
0	accuracy	xgboost	0.999531
1	recall	xgboost	0.977404
2	precision	xgboost	0.984981
3	f1	xgboost	0.981178
4	accuracy	tensorflow	0.994636
5	recall	tensorflow	0.730769
6	precision	tensorflow	0.820734
7	f1	tensorflow	0.773143

In [26]:

```
plt.figure(figsize=(8,5))
ax=plt.gca()
sns.barplot(x='metrics', y='score',hue='models',data=df2_ytest_label_metrics)
plt.title("Models' Metrics Comparasion",fontsize=14)
plt.xlabel("Metrics",fontsize=12)
plt.ylabel("Score",fontsize=12)
plt.legend(loc='lower right')
plt.ylim(0.7,1)
sns.despine()
```



In [ ]:

Appendix C

Appendix-Second Database Code

In [1]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from xgboost.sklearn import XGBClassifier

from sklearn.metrics import confusion_matrix, classification_report,ConfusionMatrixDisplay,RocCurveDisplay
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import precision_recall_curve,classification_report,confusion_matrix
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, roc_curve,roc_auc_score,au
from sklearn.model_selection import GridSearchCV,cross_val_score

import tensorflow as tf
from tensorflow.keras.utils import plot_model
dpi=100
```

In [2]:

```
df2=pd.read_csv("2.csv")
df2['id']=df2.index
df2.head()
```

Out[2]:

	Time	Ambient_Pressure	Release_rate	Humidity	Release_orientation	P01	P02	P03	P04	F
0	0.010394	964.206787	0.228	123.42527	vertical	1.840458	1.970649	1.83638	-0.005606	-174
1	0.010394	964.206787	0.228	123.42527	vertical	1.840458	1.970649	1.83638	-0.005606	-174
2	0.010394	964.206787	0.228	123.42527	vertical	1.840458	1.970649	1.83638	-0.005606	-174
3	0.010394	964.206787	0.228	123.42527	vertical	1.840458	1.970649	1.83638	-0.005606	-174
4	0.010394	964.206787	0.228	123.42527	vertical	1.840458	1.970649	1.83638	-0.005606	-174

5 rows × 23 columns

In [12]:

```
df2.columns
```

Out[12]:

```
Index(['Time', 'Ambient_Pressure', 'Release_rate', 'Humidity',
       'Release_orientation', 'P01', 'P02', 'P03', 'P04', 'PT_01', 'PT_02',
       'PT_03', 'PT_04', 'Wind_Direction_High', 'Wind_Direction_Low',
       'Wind_Speed_High', 'Wind_Speed_Low', 'HC', 'x', 'y', 'z', 'label',
       'id'],
      dtype='object')
```

In [27]:

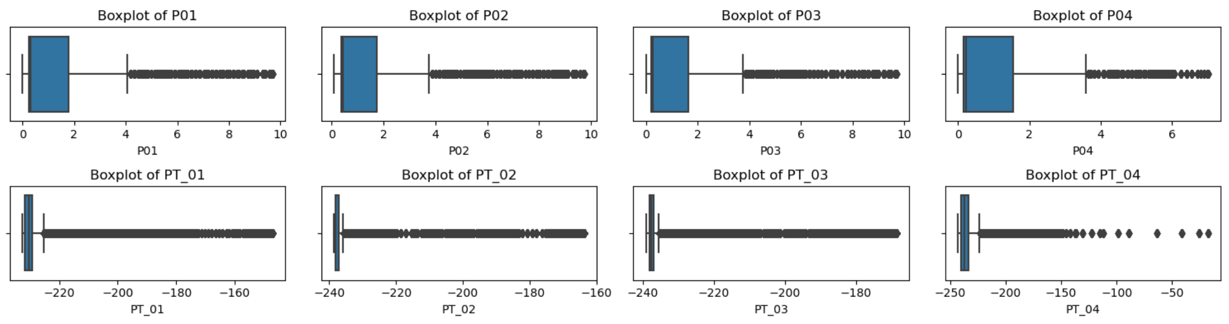
```
df2['label'].value_counts()
```

Out[27]:

```
0    656810
1      8320
Name: label, dtype: int64
```

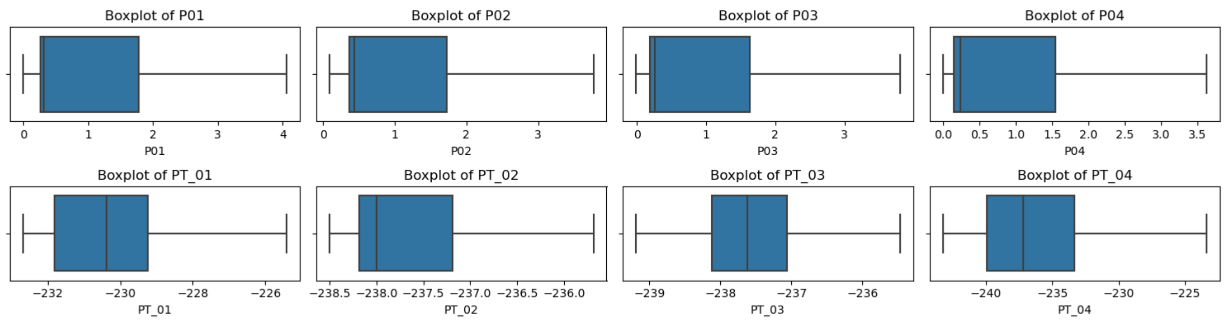
In [28]:

```python
num_features=['P01', 'P02', 'P03', 'P04', 'PT_01', 'PT_02','PT_03', 'PT_04']
fig,axs=plt.subplots(2,4,figsize=(15,4),dpi=dpi)
axs=axs.flatten()

for icol in range(len(num_features)):
    col=num_features[icol]
    sns.boxplot(x=col, data=df2, ax=axs[icol])
    axs[icol].set_title("Boxplot of {}".format(col))

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



In [3]:

```python
num_features=['P01', 'P02', 'P03', 'P04', 'PT_01', 'PT_02','PT_03', 'PT_04']
fig,axs=plt.subplots(2,4,figsize=(15,4),dpi=dpi)
axs=axs.flatten()

for icol in range(len(num_features)):
    col=num_features[icol]
    iqr=df2[col].quantile(0.75)-df2[col].quantile(0.25)
    lower=df2[col].quantile(0.25)-1.5*iqr
    upper=df2[col].quantile(0.75)+1.5*iqr
    df2[col]=df2[col].map(lambda x: lower if x<lower else x)
    df2[col]=df2[col].map(lambda x: upper if x>upper else x)

    sns.boxplot(x=col, data=df2, ax=axs[icol])
    axs[icol].set_title("Boxplot of {}".format(col))

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



In [4]:

```python
df2['Release_orientation']=df2['Release_orientation'].map(lambda x: 0 if x=='vertical' else 1)
df2_X=df2.drop(['label','id'],axis=1)
df2_y=df2['label']
```

In [5]:

```
df2_xtrain_label,df2_xtest_label,df2_ytrain_label,df2_ytest_label=train_test_split(df2_X,df2_y,test_size=0.25,
                                                                                    stratify=df2_y,
                                                                                    random_state=42)
```

In [32]:

```

model_XGB = XGBClassifier(random_state=42,learning_rate=0.1, max_depth=9).fit(df2_xtrain_label, df2_ytrain_labe
```

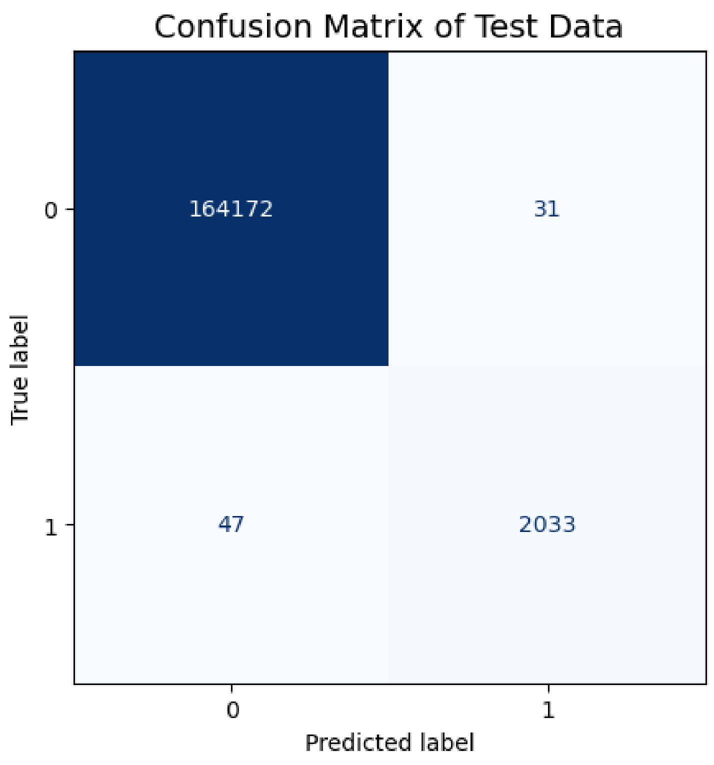
In [33]:

```
df2_ytest_label_pred_xgb=model_XGB.predict(df2_xtest_label)
print(classification_report(df2_ytest_label,df2_ytest_label_pred_xgb,digits=5))
```

```
              precision    recall  f1-score   support

           0    0.99971   0.99981   0.99976    164203
           1    0.98498   0.97740   0.98118      2080

    accuracy                        0.99953    166283
   macro avg    0.99235   0.98861   0.99047    166283
weighted avg    0.99953   0.99953   0.99953    166283
```

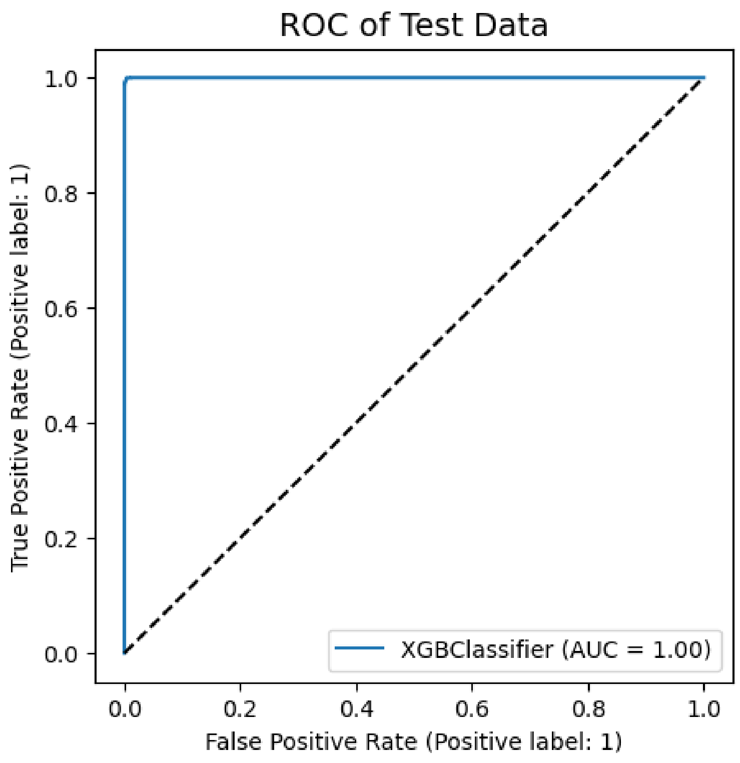
In [34]:

```

plt.figure(figsize=(5,5))
ax=plt.gca()
ConfusionMatrixDisplay.from_predictions(df2_ytest_label, df2_ytest_label_pred_xgb,
                                         ax=ax,cmap=plt.get_cmap('Blues'),colorbar=False)
plt.title("Confusion Matrix of Test Data",fontsize=14)
plt.show()
```



In [35]:

```
# roc

plt.figure(figsize=(5,5))
ax=plt.gca()
RocCurveDisplay.from_estimator(model_XGB,
                                      df2_xtest_label, df2_ytest_label,
                                      ax=ax)
ax.plot([0,1],[0,1],'k--')
plt.title("ROC of Test Data",fontsize=14)
plt.show()
```



In [6]:

```
import os
import random

seed_value= 42
os.environ['PYTHONHASHSEED']=str(seed_value)

random.seed(seed_value)
np.random.seed(seed_value)
tf.random.set_seed(seed_value)
```

In [ ]:

In [16]:

```
# we'd better normalize features first
num_features=['Time', 'Ambient_Pressure', 'Release_rate', 'Humidity',
       'Release_orientation', 'P01', 'P02', 'P03', 'P04', 'PT_01', 'PT_02',
       'PT_03', 'PT_04', 'Wind_Direction_High', 'Wind_Direction_Low',
       'Wind_Speed_High', 'Wind_Speed_Low', 'HC', 'x', 'y', 'z']

scaler=StandardScaler()
df2_xtrain_label_scale=df2_xtrain_label.copy()
df2_xtest_label_scale=df2_xtest_label.copy()

for feature in num_features:

    # fit scaler
    df2_xtrain_label_scale[feature]=scaler.fit_transform(df2_xtrain_label_scale[[feature]])

    # use the same scaler to transform test data
    df2_xtest_label_scale[feature]=scaler.transform(df2_xtest_label_scale[[feature]])

df2_xtest_label_scale.head(2)
```

Out[16]:

	Time	Ambient_Pressure	Release_rate	Humidity	Release_orientation	P01	P02	P03	P
396066	0.320809	-0.875085	1.130321	1.102725	-0.306588	0.795884	0.758640	0.784188	0.765
5287	-1.364763	1.207775	-0.735238	-1.278630	-0.306588	-0.671204	-0.614063	-0.696022	-0.700

2 rows × 21 columns

In [17]:

```
tf.keras.backend.clear_session()
tf_df2_label_bestmodel = tf.keras.models.Sequential([

    tf.keras.layers.Dense(64, activation='relu',input_shape=[df2_xtrain_label_scale.shape[1]]),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(1, activation="sigmoid")

])
```

In [9]:

```
tf_df2_label_bestmodel.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                1408

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 8)                 264

 dense_3 (Dense)             (None, 1)                 9

=================================================================
Total params: 3,761
Trainable params: 3,761
Non-trainable params: 0
_________________________________________________________________
```

In [*]:

```
tf.keras.backend.clear_session()

optimizer=tf.keras.optimizers.SGD(learning_rate=1e-2)
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', min_delta=0.0001, patience=5,restore_best_

tf_df2_label_bestmodel.compile(loss='binary_crossentropy',
                optimizer=optimizer,
                metrics=['accuracy'])

tf_df2_label_bestmodel.fit(df2_xtrain_label_scale, df2_ytrain_label, epochs=100,batch_size=32,
          validation_split=0.1, callbacks=[early_stop])
```

```
Epoch 1/100
14031/14031 [==============================] - 21s 1ms/step - loss: 0.0439 - accuracy: 0.9874 - v
al_loss: 0.0333 - val_accuracy: 0.9896
Epoch 2/100
14031/14031 [==============================] - 21s 1ms/step - loss: 0.0342 - accuracy: 0.9886 - v
al_loss: 0.0306 - val_accuracy: 0.9896
Epoch 3/100
14031/14031 [==============================] - 22s 2ms/step - loss: 0.0298 - accuracy: 0.9888 - v
al_loss: 0.0260 - val_accuracy: 0.9900
Epoch 4/100
14031/14031 [==============================] - 22s 2ms/step - loss: 0.0253 - accuracy: 0.9894 - v
al_loss: 0.0222 - val_accuracy: 0.9908
Epoch 5/100
14031/14031 [==============================] - 22s 2ms/step - loss: 0.0222 - accuracy: 0.9904 - v
al_loss: 0.0202 - val_accuracy: 0.9916
Epoch 6/100
14031/14031 [==============================] - 22s 2ms/step - loss: 0.0204 - accuracy: 0.9911 - v
al_loss: 0.0208 - val_accuracy: 0.9909
Epoch 7/100
14031/14031 [==============================] - 20s 1ms/step - loss: 0.0191 - accuracy: 0.9916 - v
al_loss: 0.0179 - val_accuracy: 0.9922
Epoch 8/100
14031/14031 [==============================] - 21s 1ms/step - loss: 0.0180 - accuracy: 0.9920 - v
al_loss: 0.0161 - val_accuracy: 0.9929
Epoch 9/100
14031/14031 [==============================] - 19s 1ms/step - loss: 0.0171 - accuracy: 0.9924 - v
al_loss: 0.0156 - val_accuracy: 0.9930
Epoch 10/100
14031/14031 [==============================] - 21s 2ms/step - loss: 0.0163 - accuracy: 0.9928 - v
al_loss: 0.0179 - val_accuracy: 0.9917
Epoch 11/100
14031/14031 [==============================] - 19s 1ms/step - loss: 0.0157 - accuracy: 0.9931 - v
al_loss: 0.0156 - val_accuracy: 0.9934
Epoch 12/100
14031/14031 [==============================] - 14s 1ms/step - loss: 0.0153 - accuracy: 0.9933 - v
al_loss: 0.0136 - val_accuracy: 0.9938
Epoch 13/100
14031/14031 [==============================] - 14s 1ms/step - loss: 0.0149 - accuracy: 0.9934 - v
al_loss: 0.0145 - val_accuracy: 0.9939
Epoch 14/100
14031/14031 [==============================] - 14s 988us/step - loss: 0.0145 - accuracy: 0.9936 -
val_loss: 0.0138 - val_accuracy: 0.9942
Epoch 15/100
14031/14031 [==============================] - 14s 1ms/step - loss: 0.0141 - accuracy: 0.9938 - v
al_loss: 0.0137 - val_accuracy: 0.9939
Epoch 16/100
 8497/14031 [=================>...........] - ETA: 5s - loss: 0.0136 - accuracy: 0.9940
```

In [41]:

```
df2_label_history=pd.DataFrame(tf_df2_label_bestmodel.history.history)
df2_label_history
```

Out[41]:

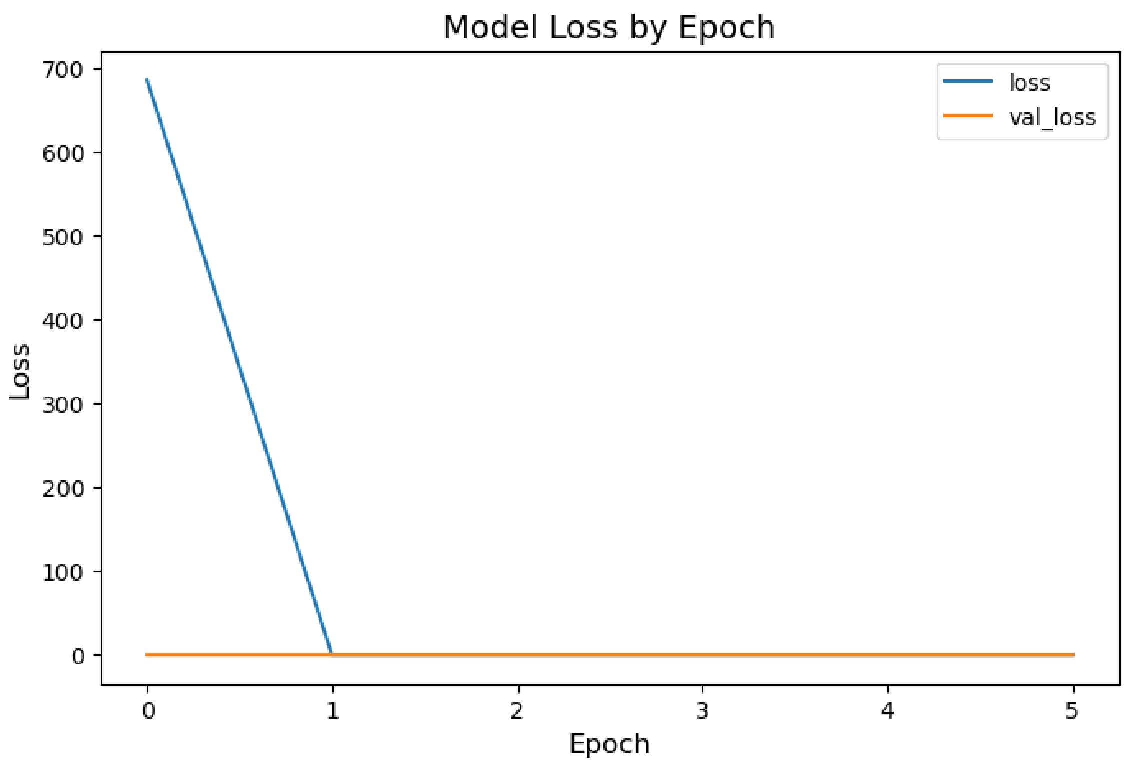
	loss	accuracy	val_loss	val_accuracy
0	685.120972	0.987262	0.064112	0.988313
1	0.067710	0.987400	0.063693	0.988313
2	0.067638	0.987400	0.063648	0.988313
3	0.067635	0.987400	0.063651	0.988313
4	0.067635	0.987400	0.063659	0.988313
5	0.067636	0.987400	0.063648	0.988313

In [42]:

```
plt.figure(figsize=(8,5))
ax=plt.gca()
df2_label_history[['loss','val_loss']].plot(kind='line',ax=ax)
plt.title('Model Loss by Epoch',fontsize=14)
plt.xlabel("Epoch",fontsize=12)
plt.ylabel("Loss",fontsize=12)
```

Out[42]:

```
Text(0, 0.5, 'Loss')
```



In [43]:

```
df2_ytest_label_pred_tf = np.where(tf_df2_label_bestmodel.predict(df2_xtest_label_scale)>0.5,1,0)
```

```
5197/5197 [==============================] - 6s 1ms/step
```

In [44]:

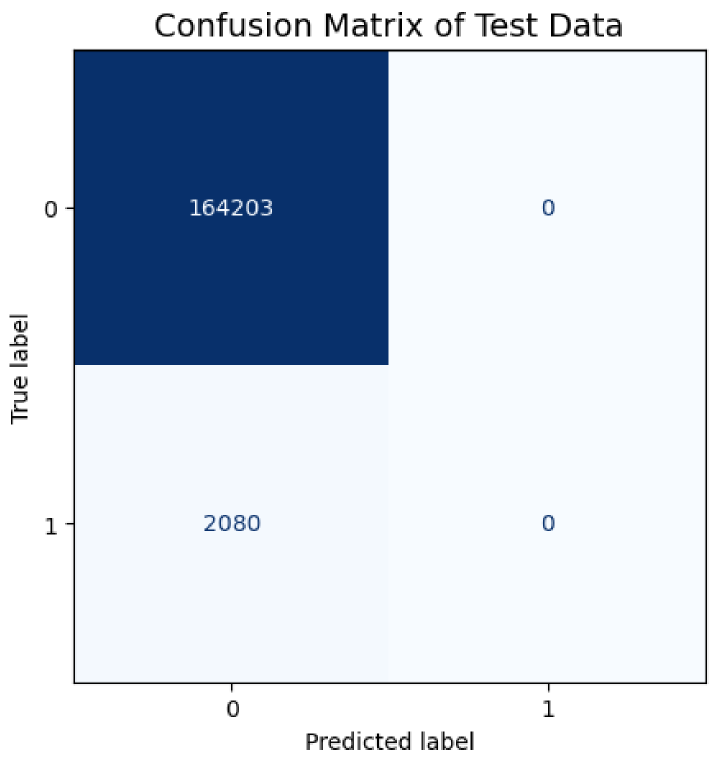
```
print(classification_report(df2_ytest_label,df2_ytest_label_pred_tf,digits=5))
```

```
              precision    recall  f1-score   support

           0    0.98749   1.00000   0.99371    164203
           1    0.00000   0.00000   0.00000      2080

    accuracy                        0.98749    166283
   macro avg    0.49375   0.50000   0.49685    166283
weighted avg    0.97514   0.98749   0.98128    166283
```

```
C:\Users\Danni\anaconda3\anaconda\lib\site-packages\sklearn\metrics\_classification.py:1344: Unde
finedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Danni\anaconda3\anaconda\lib\site-packages\sklearn\metrics\_classification.py:1344: Unde
finedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Danni\anaconda3\anaconda\lib\site-packages\sklearn\metrics\_classification.py:1344: Unde
finedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [45]:

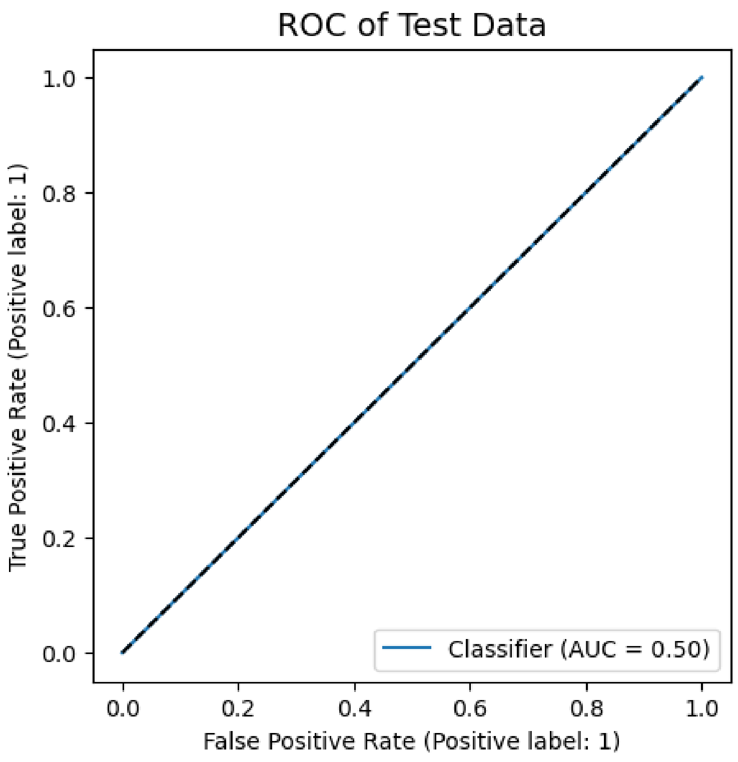
```
# confusion matrix

plt.figure(figsize=(5,5))
ax=plt.gca()
ConfusionMatrixDisplay.from_predictions(df2_ytest_label, df2_ytest_label_pred_tf,
                                        ax=ax,cmap=plt.get_cmap('Blues'),colorbar=False)
plt.title("Confusion Matrix of Test Data",fontsize=14)
plt.show()
```



```
In [46]:
```

```
# roc
df2_ytest_label_pred_proba_tf = tf_df2_label_bestmodel.predict(df2_xtest_label_scale)

plt.figure(figsize=(5,5))
ax=plt.gca()
RocCurveDisplay.from_predictions(df2_ytest_label,df2_ytest_label_pred_proba_tf,ax=ax)
ax.plot([0,1],[0,1],'k--')
plt.title("ROC of Test Data",fontsize=14)
plt.show()
```

```
5197/5197 [==============================] - 6s 1ms/step
```



In [47]:

```
df2_ytest_label_accuracy_xgb=accuracy_score(df2_ytest_label,df2_ytest_label_pred_xgb)
df2_ytest_label_recall_xgb=recall_score(df2_ytest_label,df2_ytest_label_pred_xgb)
df2_ytest_label_precision_xgb=precision_score(df2_ytest_label,df2_ytest_label_pred_xgb)
df2_ytest_label_f1_xgb=f1_score(df2_ytest_label,df2_ytest_label_pred_xgb)

df2_ytest_label_accuracy_tf=accuracy_score(df2_ytest_label,df2_ytest_label_pred_tf)
df2_ytest_label_recall_tf=recall_score(df2_ytest_label,df2_ytest_label_pred_tf)
df2_ytest_label_precision_tf=precision_score(df2_ytest_label,df2_ytest_label_pred_tf)
df2_ytest_label_f1_tf=f1_score(df2_ytest_label,df2_ytest_label_pred_tf)

df2_ytest_label_metrics=pd.DataFrame([
    {'metrics':'accuracy','models':'xgboost','score':df2_ytest_label_accuracy_xgb},
    {'metrics':'recall','models':'xgboost','score':df2_ytest_label_recall_xgb},
    {'metrics':'precision','models':'xgboost','score':df2_ytest_label_precision_xgb},
    {'metrics':'f1','models':'xgboost','score':df2_ytest_label_f1_xgb},

    {'metrics':'accuracy','models':'tensorflow','score':df2_ytest_label_accuracy_tf},
    {'metrics':'recall','models':'tensorflow','score':df2_ytest_label_recall_tf},
    {'metrics':'precision','models':'tensorflow','score':df2_ytest_label_precision_tf},
    {'metrics':'f1','models':'tensorflow','score':df2_ytest_label_f1_tf},
])

df2_ytest_label_metrics
```

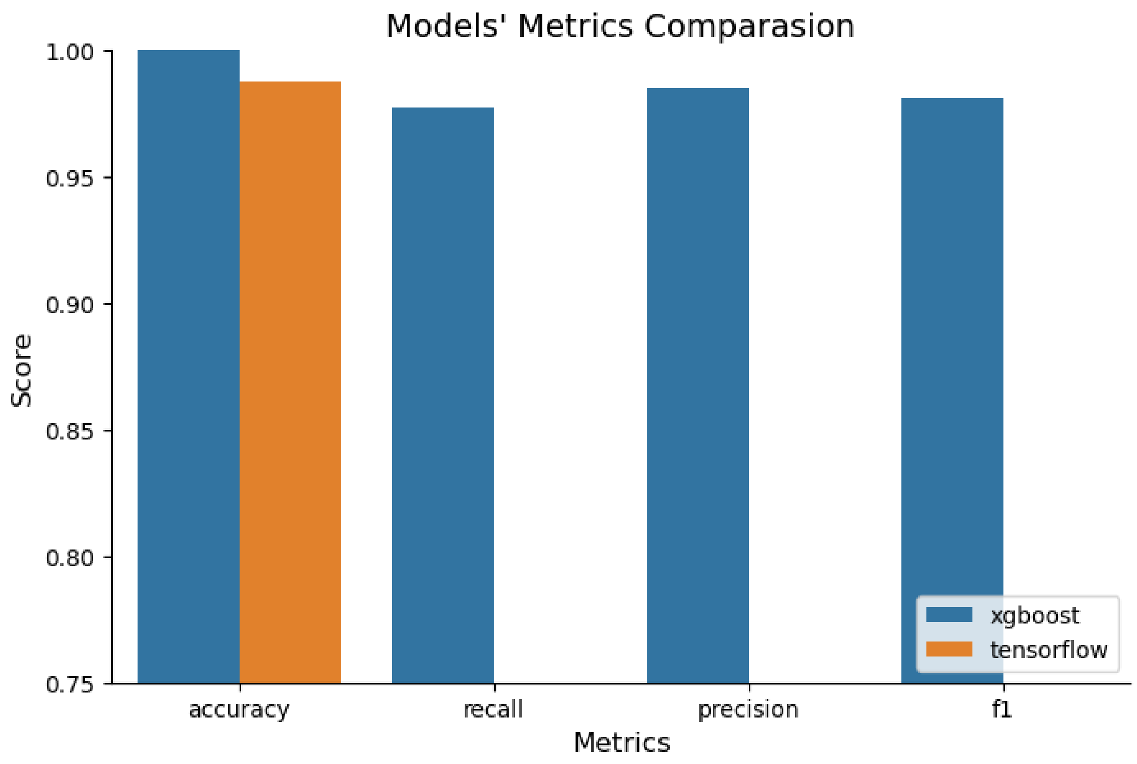
```
C:\Users\Danni\anaconda3\anaconda\lib\site-packages\sklearn\metrics\_classification.py:1344: Unde
finedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Us
e `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[47]:

	metrics	models	score
0	accuracy	xgboost	0.999531
1	recall	xgboost	0.977404
2	precision	xgboost	0.984981
3	f1	xgboost	0.981178
4	accuracy	tensorflow	0.987491
5	recall	tensorflow	0.000000
6	precision	tensorflow	0.000000
7	f1	tensorflow	0.000000

In [48]:

```
plt.figure(figsize=(8,5))
ax=plt.gca()
sns.barplot(x='metrics', y='score',hue='models',data=df2_ytest_label_metrics)
plt.title("Models' Metrics Comparasion",fontsize=14)
plt.xlabel("Metrics",fontsize=12)
plt.ylabel("Score",fontsize=12)
plt.legend(loc='lower right')
plt.ylim(0.75,1)
sns.despine()
```



In [ ]:

Appendix D

Appendix-Third Database Code

In [1]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from xgboost.sklearn import XGBClassifier

from sklearn.metrics import confusion_matrix, classification_report,ConfusionMatrixDispl
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import precision_recall_curve,classification_report,confusion_matri
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, roc
from sklearn.model_selection import GridSearchCV,cross_val_score

import tensorflow as tf
from tensorflow.keras.utils import plot_model

dpi=100
```

In [2]:

```
df3=pd.read_csv("3.csv")
df3.columns=[x.lower().strip() for x in df3.columns]
df3['id']=df3.index
df3.head()
```

Out[2]:

	time	ambient_pressure	release_rate	humidity	purge	obstacles	vent panel	seal
0	0.097852	953.372375	0.183	108.543892	air	no	polyethylene	n
1	0.097852	953.372375	0.183	108.543892	air	no	polyethylene	n
2	0.097852	953.372375	0.183	108.543892	air	no	polyethylene	n
3	0.097852	953.372375	0.183	108.543892	air	no	polyethylene	n
4	0.097852	953.372375	0.183	108.543892	air	no	polyethylene	n

5 rows × 27 columns

In [3]:

```
df3['purge'].value_counts()
```

Out[3]:

```
air         381000
nitrogen     85250
Name: purge, dtype: int64
```

In [4]:

```
df3['obstacles'].value_counts()
```

Out[4]:

```
yes    235750
no     230500
Name: obstacles, dtype: int64
```

In [5]:

```
df3['vent panel'].value_counts()
```

Out[5]:

```
polyethylene           416000
double polyethylene     50250
Name: vent panel, dtype: int64
```

In [6]:

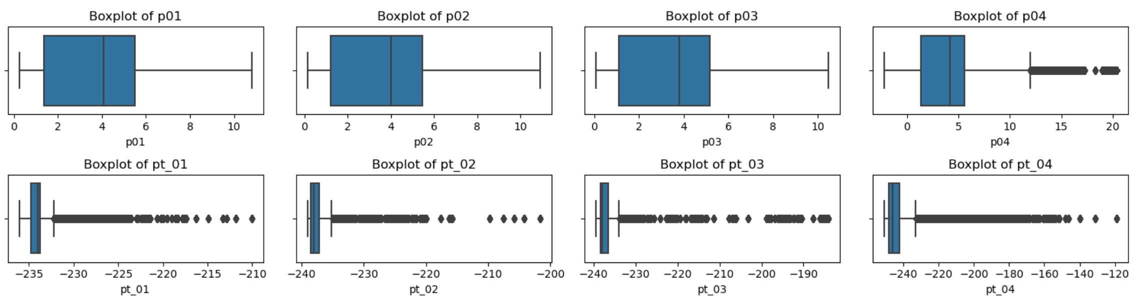
```
df3['sealing'].value_counts()
```

Out[6]:

```
none                                        230500
low-level vent and opening to vent mast    135500
low-level vent                              100250
Name: sealing, dtype: int64
```

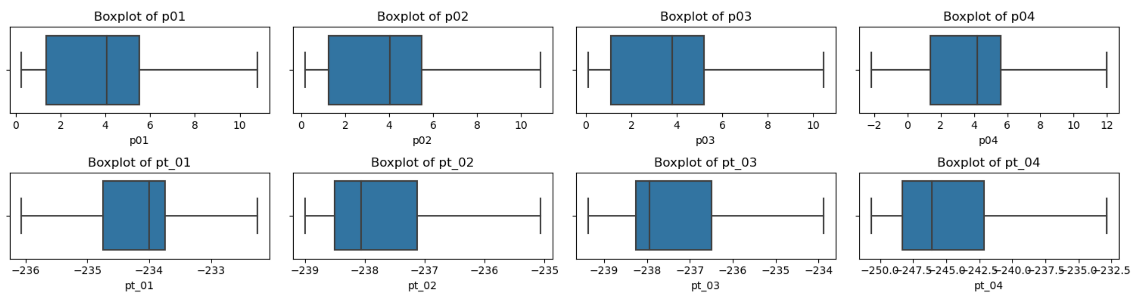
In [7]:

```
num_features=['p01', 'p02', 'p03', 'p04', 'pt_01', 'pt_02','pt_03', 'pt_04']
fig,axs=plt.subplots(2,4,figsize=(15,4),dpi=dpi)
axs=axs.flatten()

for icol in range(len(num_features)):
    col=num_features[icol]
    sns.boxplot(x=col, data=df3, ax=axs[icol])
    axs[icol].set_title("Boxplot of {}".format(col))

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



In [8]:

```
num_features=['p01', 'p02', 'p03', 'p04', 'pt_01', 'pt_02','pt_03', 'pt_04']
fig,axs=plt.subplots(2,4,figsize=(15,4),dpi=dpi)
axs=axs.flatten()

for icol in range(len(num_features)):
    col=num_features[icol]
    iqr=df3[col].quantile(0.75)-df3[col].quantile(0.25)
    lower=df3[col].quantile(0.25)-1.5*iqr
    upper=df3[col].quantile(0.75)+1.5*iqr
    df3[col]=df3[col].map(lambda x: lower if x<lower else x)
    df3[col]=df3[col].map(lambda x: upper if x>upper else x)

    sns.boxplot(x=col, data=df3, ax=axs[icol])
    axs[icol].set_title("Boxplot of {}".format(col))

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



In [9]:

```
df3['purge']=df3['purge'].map(lambda x: 0 if x=='air' else 1)
df3['obstacles']=df3['obstacles'].map(lambda x: 0 if x=='no' else 1)
df3['vent panel']=df3['vent panel'].map(lambda x: 0 if x=='polyethylene' else 1)

df3=pd.get_dummies(data=df3, columns=['sealing'])
```

In [10]:

```
df3_X=df3.drop(['liquid_oxygen','solid_oxygen','id'],axis=1)
df3_y1=df3['liquid_oxygen']
df3_y2=df3['solid_oxygen']
```

In [11]:

```
df3_xtrain_liquid,df3_xtest_liquid,df3_ytrain_liquid,df3_ytest_liquid=train_test_split(d
                                                                                        s
                                                                                        r
df3_xtrain_solid,df3_xtest_solid,df3_ytrain_solid,df3_ytest_solid=train_test_split(df3_X
                                                                                        s
                                                                                    rando
```

In [12]:

```
model_XGB = XGBClassifier(random_state=42,learning_rate=0.1, max_depth=9).fit(df3_xtrain
```

In [13]:

```
df3_ytest_liquid_pred_xgb=model_XGB.predict(df3_xtest_liquid)
print(classification_report(df3_ytest_liquid,df3_ytest_liquid_pred_xgb,digits=5))
```

```
              precision    recall  f1-score   support

           0    0.99907   0.99800   0.99854     54991
           1    0.99822   0.99917   0.99869     61572

    accuracy                        0.99862    116563
   macro avg    0.99864   0.99859   0.99861    116563
weighted avg    0.99862   0.99862   0.99862    116563
```

In [14]:

```
model_XGB = XGBClassifier(random_state=42,learning_rate=0.1, max_depth=9).fit(df3_xtrain
```

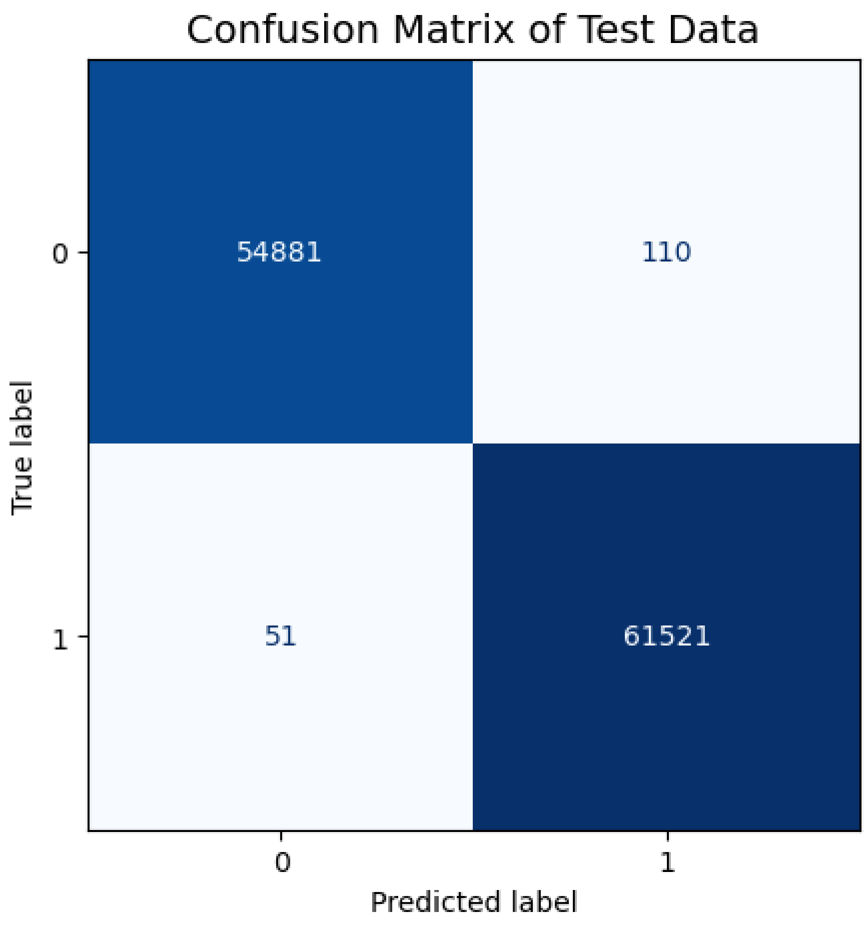
In [15]:

```
df3_ytest_solid_pred_xgb=model_XGB.predict(df3_xtest_solid)
print(classification_report(df3_ytest_solid,df3_ytest_solid_pred_xgb,digits=5))
```

```
              precision    recall  f1-score   support

           0    0.99973   0.99985   0.99979     88527
           1    0.99954   0.99914   0.99934     28036

    accuracy                        0.99968    116563
   macro avg    0.99963   0.99950   0.99957    116563
weighted avg    0.99968   0.99968   0.99968    116563
```

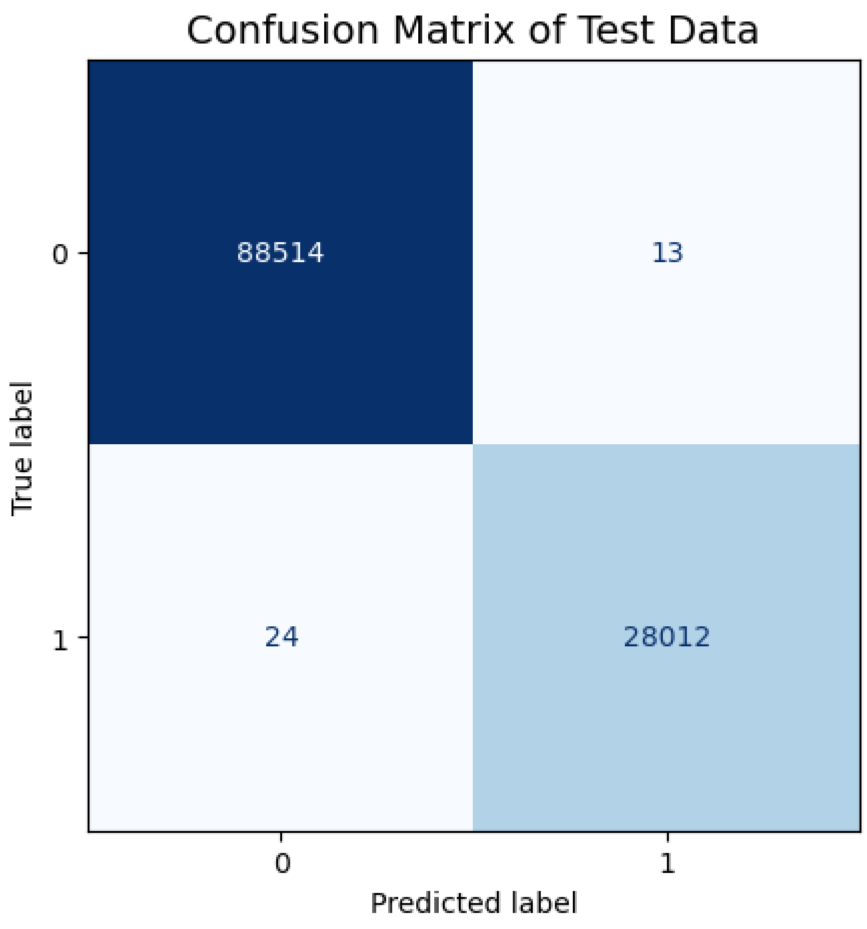
In [16]:

```
# confusion matrix

plt.figure(figsize=(5,5))
ax=plt.gca()
ConfusionMatrixDisplay.from_predictions(df3_ytest_liquid, df3_ytest_liquid_pred_xgb,
                                        ax=ax,cmap=plt.get_cmap('Blues'),colorbar=False)
plt.title("Confusion Matrix of Test Data",fontsize=14)
plt.show()
```



In [17]:

```
plt.figure(figsize=(5,5))
ax=plt.gca()
ConfusionMatrixDisplay.from_predictions(df3_ytest_solid, df3_ytest_solid_pred_xgb,
                                      ax=ax,cmap=plt.get_cmap('Blues'),colorbar=False)
plt.title("Confusion Matrix of Test Data",fontsize=14)
plt.show()
```



In [18]:

```
import os
import random

seed_value= 42
os.environ['PYTHONHASHSEED']=str(seed_value)

random.seed(seed_value)
np.random.seed(seed_value)
tf.random.set_seed(seed_value)
```

In [19]:

```
# we'd better normalize features first
num_features=['time', 'ambient_pressure', 'release_rate', 'humidity',
              'p01', 'p02', 'p03', 'p04', 'pt_01', 'pt_02',
              'pt_03', 'pt_04', 'wind_direction_high', 'wind_direction_low',
              'wind_speed_high', 'wind_speed_low', 'tt', 'x', 'y', 'z']

scaler=StandardScaler()
df3_xtrain_liquid_scale=df3_xtrain_liquid.copy()
df3_xtest_liquid_scale=df3_xtest_liquid.copy()

for feature in num_features:

    # fit scaler
    df3_xtrain_liquid_scale[feature]=scaler.fit_transform(df3_xtrain_liquid_scale[[featu

    # use the same scaler to transform test data
    df3_xtest_liquid_scale[feature]=scaler.transform(df3_xtest_liquid_scale[[feature]])

df3_xtest_liquid_scale.head(2)
```

Out[19]:

	time	ambient_pressure	release_rate	humidity	purge	obstacles	vent panel	
275253	-0.162384	0.950700	0.234349	-1.696981	0	1	0	0.600
233600	-1.502880	0.946625	0.234349	-1.737877	0	1	0	1.420

2 rows × 26 columns

In [20]:

```
df3_xtrain_liquid_scale['sealing_low-level vent']=df3_xtrain_liquid_scale['sealing_low-l
df3_xtrain_liquid_scale['sealing_low-level vent and opening to vent mast']=df3_xtrain_li
df3_xtrain_liquid_scale['sealing_none']=df3_xtrain_liquid_scale['sealing_none'].astype(i

df3_xtest_liquid_scale['sealing_low-level vent']=df3_xtest_liquid_scale['sealing_low-lev
df3_xtest_liquid_scale['sealing_low-level vent and opening to vent mast']=df3_xtest_liqu
df3_xtest_liquid_scale['sealing_none']=df3_xtest_liquid_scale['sealing_none'].astype(int
```

In [21]:

```
tf.keras.backend.clear_session()
tf_df3_liquid_bestmodel = tf.keras.models.Sequential([

    tf.keras.layers.Dense(64, activation='relu',input_shape=[df3_xtrain_liquid_scale.sha
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(1, activation="sigmoid")

])
```

In [22]:

```
tf_df3_liquid_bestmodel.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 64)                1728      
                                                                 
 dense_1 (Dense)             (None, 32)                2080      
                                                                 
 dense_2 (Dense)             (None, 8)                 264       
                                                                 
 dense_3 (Dense)             (None, 1)                 9         
                                                                 
=================================================================
Total params: 4,081
Trainable params: 4,081
Non-trainable params: 0
_________________________________________________________________
```

In [23]:

```
# train model

optimizer=tf.keras.optimizers.SGD(learning_rate=1e-2)
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', min_delta=0.0001,

tf_df3_liquid_bestmodel.compile(loss='binary_crossentropy',
                optimizer=optimizer,
                metrics=['accuracy'])

tf_df3_liquid_bestmodel.fit(df3_xtrain_liquid_scale, df3_ytrain_liquid, epochs=100,batch
          validation_split=0.1, callbacks=[early_stop])
```

```
Epoch 1/100
9835/9835 [==============================] - 17s 2ms/step - loss: 0.1163
- accuracy: 0.9585 - val_loss: 0.0448 - val_accuracy: 0.9851
Epoch 2/100
9835/9835 [==============================] - 16s 2ms/step - loss: 0.0355
- accuracy: 0.9874 - val_loss: 0.0327 - val_accuracy: 0.9893
Epoch 3/100
9835/9835 [==============================] - 16s 2ms/step - loss: 0.0270
- accuracy: 0.9902 - val_loss: 0.0262 - val_accuracy: 0.9900
Epoch 4/100
9835/9835 [==============================] - 16s 2ms/step - loss: 0.0229
- accuracy: 0.9913 - val_loss: 0.0216 - val_accuracy: 0.9921
Epoch 5/100
9835/9835 [==============================] - 16s 2ms/step - loss: 0.0203
- accuracy: 0.9923 - val_loss: 0.0203 - val_accuracy: 0.9922
Epoch 6/100
9835/9835 [==============================] - 16s 2ms/step - loss: 0.0193
- accuracy: 0.9925 - val_loss: 0.0170 - val_accuracy: 0.9934
Epoch 7/100
9835/9835 [==============================] - 17s 2ms/step - loss: 0.0180
- accuracy: 0.9931 - val_loss: 0.0265 - val_accuracy: 0.9902
Epoch 8/100
9835/9835 [==============================] - 16s 2ms/step - loss: 0.0175
- accuracy: 0.9932 - val_loss: 0.0155 - val_accuracy: 0.9941
Epoch 9/100
9835/9835 [==============================] - 16s 2ms/step - loss: 0.0167
- accuracy: 0.9934 - val_loss: 0.0139 - val_accuracy: 0.9947
Epoch 10/100
9835/9835 [==============================] - 16s 2ms/step - loss: 0.0163
- accuracy: 0.9936 - val_loss: 0.0174 - val_accuracy: 0.9929
Epoch 11/100
9835/9835 [==============================] - 16s 2ms/step - loss: 0.0160
- accuracy: 0.9938 - val_loss: 0.0141 - val_accuracy: 0.9948
Epoch 12/100
9835/9835 [==============================] - 16s 2ms/step - loss: 0.0154
- accuracy: 0.9940 - val_loss: 0.0133 - val_accuracy: 0.9950
Epoch 13/100
9835/9835 [==============================] - 16s 2ms/step - loss: 0.0153
- accuracy: 0.9938 - val_loss: 0.0139 - val_accuracy: 0.9946
Epoch 14/100
9835/9835 [==============================] - 16s 2ms/step - loss: 0.0150
- accuracy: 0.9942 - val_loss: 0.0169 - val_accuracy: 0.9935
Epoch 15/100
9835/9835 [==============================] - 16s 2ms/step - loss: 0.0146
- accuracy: 0.9942 - val_loss: 0.0144 - val_accuracy: 0.9943
Epoch 16/100
9835/9835 [==============================] - 16s 2ms/step - loss: 0.0144
- accuracy: 0.9943 - val_loss: 0.0157 - val_accuracy: 0.9939
Epoch 17/100
9835/9835 [==============================] - 16s 2ms/step - loss: 0.0140
- accuracy: 0.9944 - val_loss: 0.0135 - val_accuracy: 0.9945
```

Out[23]:

```
<keras.callbacks.History at 0x1850d1f53c0>
```

In [24]:

```
df3_liquid_history=pd.DataFrame(tf_df3_liquid_bestmodel.history.history)
df3_liquid_history
```

Out[24]:

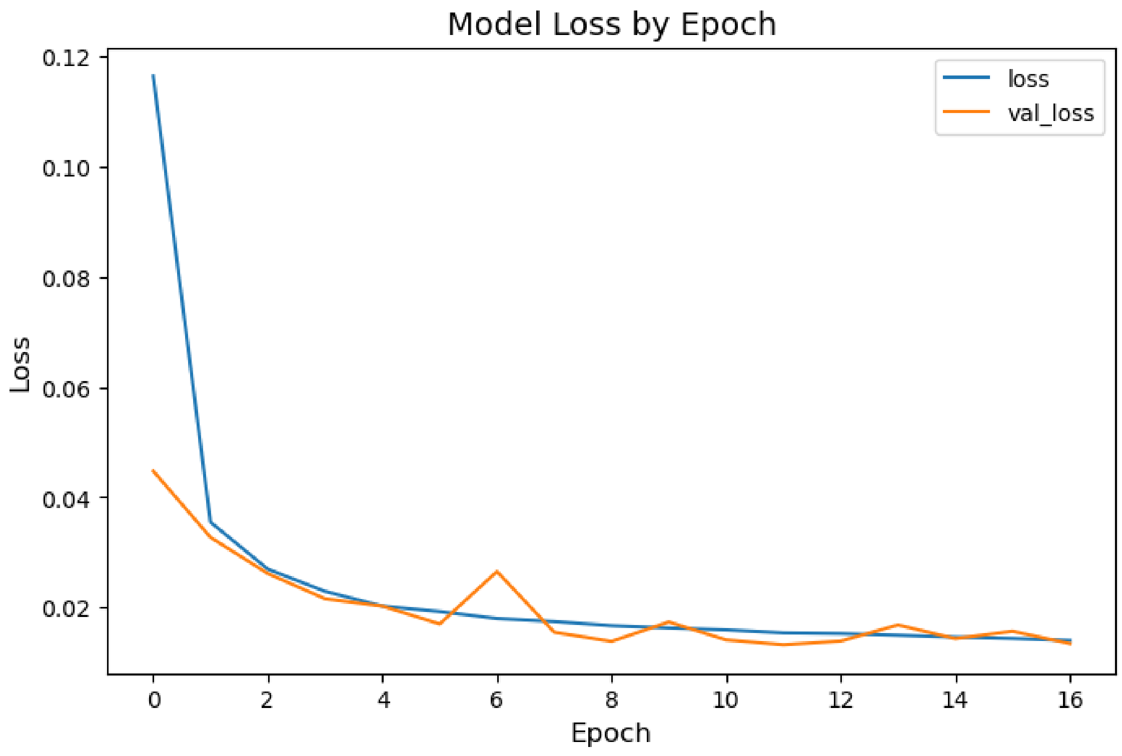
	loss	accuracy	val_loss	val_accuracy
0	0.116339	0.958493	0.044754	0.985101
1	0.035485	0.987443	0.032728	0.989276
2	0.026974	0.990223	0.026172	0.989963
3	0.022937	0.991297	0.021596	0.992079
4	0.020268	0.992333	0.020267	0.992164
5	0.019282	0.992546	0.017031	0.993394
6	0.018016	0.993067	0.026539	0.990163
7	0.017474	0.993222	0.015528	0.994052
8	0.016720	0.993420	0.013870	0.994738
9	0.016301	0.993626	0.017391	0.992908
10	0.015962	0.993779	0.014148	0.994795
11	0.015413	0.994001	0.013254	0.994967
12	0.015298	0.993810	0.013926	0.994567
13	0.014992	0.994249	0.016854	0.993451
14	0.014640	0.994182	0.014377	0.994309
15	0.014405	0.994274	0.015713	0.993937
16	0.014043	0.994408	0.013479	0.994452

In [25]:

```
plt.figure(figsize=(8,5))
ax=plt.gca()
df3_liquid_history[['loss','val_loss']].plot(kind='line',ax=ax)
plt.title('Model Loss by Epoch',fontsize=14)
plt.xlabel("Epoch",fontsize=12)
plt.ylabel("Loss",fontsize=12)
```

Out[25]:

```
Text(0, 0.5, 'Loss')
```

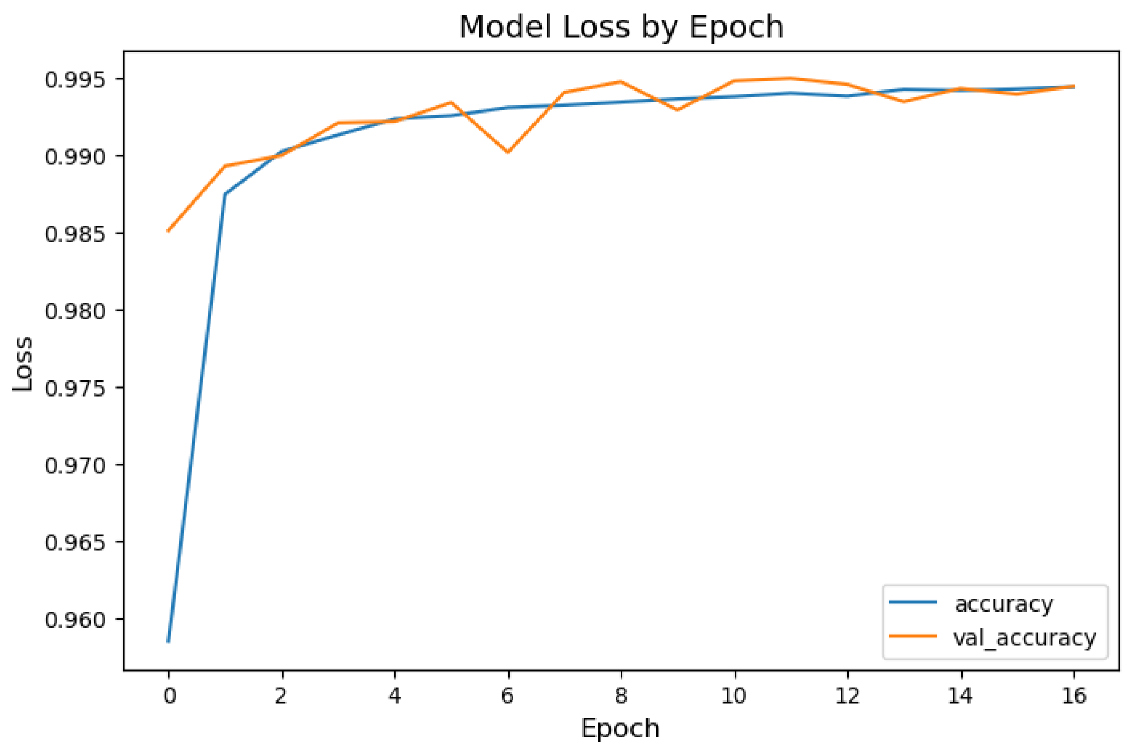


In [26]:

```
plt.figure(figsize=(8,5))
ax=plt.gca()
df3_liquid_history[['accuracy','val_accuracy']].plot(kind='line',ax=ax)
plt.title('Model Loss by Epoch',fontsize=14)
plt.xlabel("Epoch",fontsize=12)
plt.ylabel("Loss",fontsize=12)
```

Out[26]:

```
Text(0, 0.5, 'Loss')
```



In [27]:

```
df3_ytest_liquid_pred_tf = np.where(tf_df3_liquid_bestmodel.predict(df3_xtest_liquid_sca
```

```
3643/3643 [==============================] - 4s 1ms/step
```

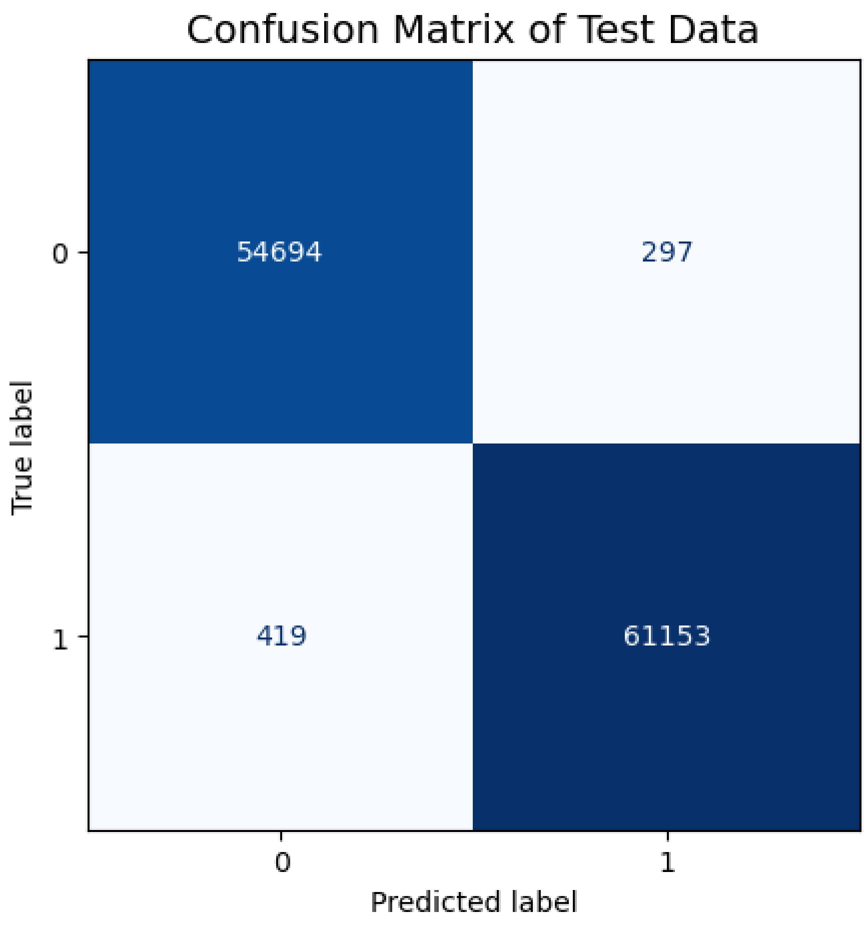
In [28]:

```
print(classification_report(df3_ytest_liquid,df3_ytest_liquid_pred_tf,digits=5))
```

```
              precision    recall  f1-score   support

           0    0.99362   0.99469   0.99416     54991
           1    0.99525   0.99430   0.99478     61572

    accuracy                        0.99448    116563
   macro avg    0.99444   0.99449   0.99447    116563
weighted avg    0.99448   0.99448   0.99448    116563
```

In [62]:

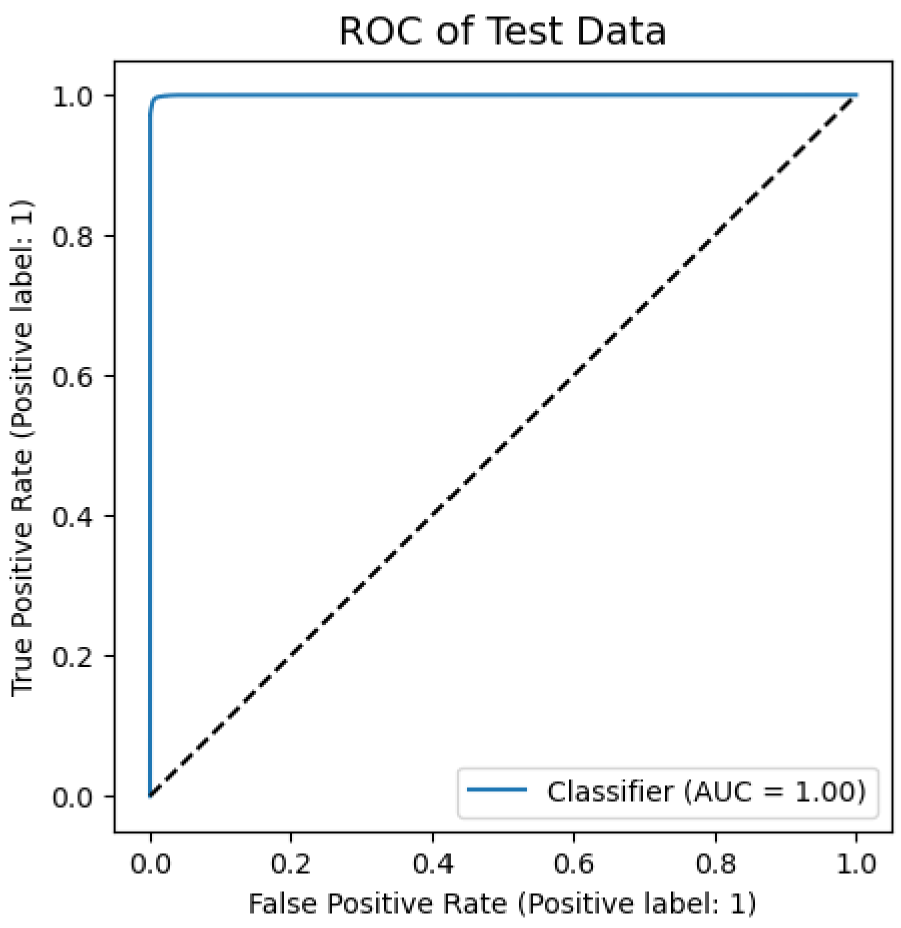
```
# confusion matrix

plt.figure(figsize=(5,5))
ax=plt.gca()
ConfusionMatrixDisplay.from_predictions(df3_ytest_liquid, df3_ytest_liquid_pred_tf,
                                       ax=ax,cmap=plt.get_cmap('Blues'),colorbar=False)
plt.title("Confusion Matrix of Test Data",fontsize=14)
plt.show()
```



In [63]:

```
# roc
df3_ytest_liquid_pred_proba_tf = tf_df3_liquid_bestmodel.predict(df3_xtest_liquid_scale)

plt.figure(figsize=(5,5))
ax=plt.gca()
RocCurveDisplay.from_predictions(df3_ytest_liquid,df3_ytest_liquid_pred_proba_tf,ax=ax)
ax.plot([0,1],[0,1],'k--')
plt.title("ROC of Test Data",fontsize=14)
plt.show()
```

```
3643/3643 [==============================] - 10s 3ms/step
```



In [64]:

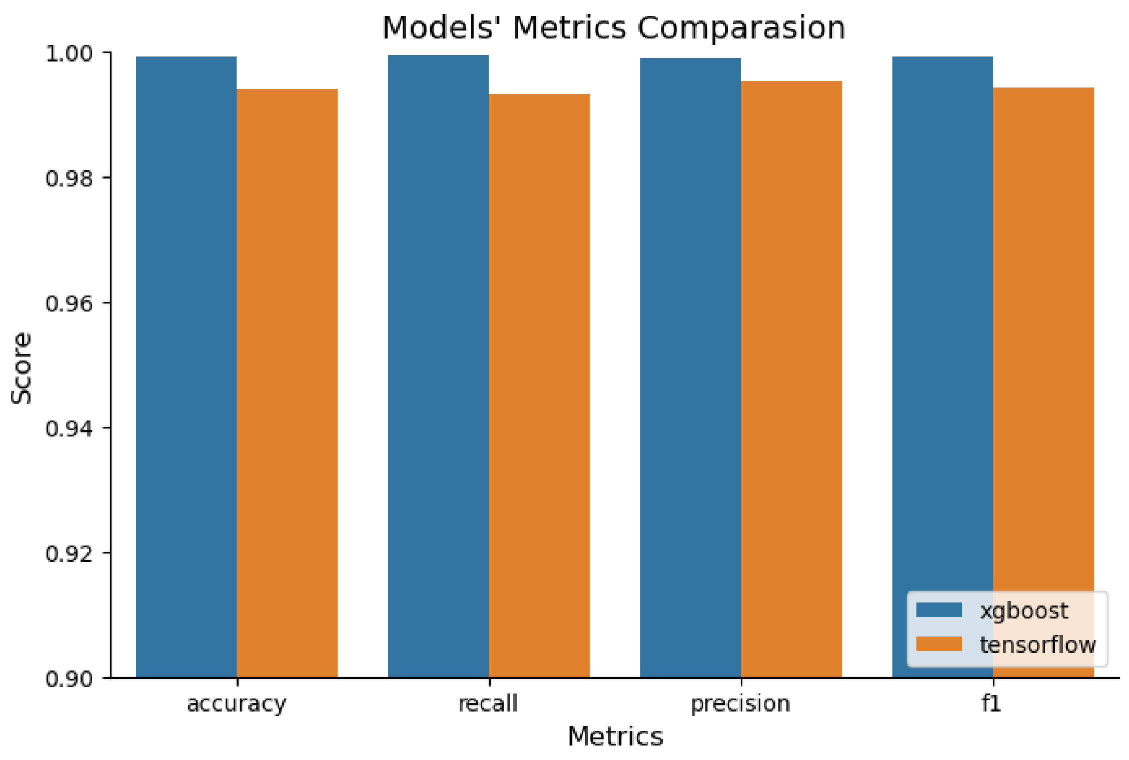
```
df3_ytest_liquid_accuracy_xgb=accuracy_score(df3_ytest_liquid,df3_ytest_liquid_pred_xgb)
df3_ytest_liquid_recall_xgb=recall_score(df3_ytest_liquid,df3_ytest_liquid_pred_xgb)
df3_ytest_liquid_precision_xgb=precision_score(df3_ytest_liquid,df3_ytest_liquid_pred_xg
df3_ytest_liquid_f1_xgb=f1_score(df3_ytest_liquid,df3_ytest_liquid_pred_xgb)

df3_ytest_liquid_accuracy_tf=accuracy_score(df3_ytest_liquid,df3_ytest_liquid_pred_tf)
df3_ytest_liquid_recall_tf=recall_score(df3_ytest_liquid,df3_ytest_liquid_pred_tf)
df3_ytest_liquid_precision_tf=precision_score(df3_ytest_liquid,df3_ytest_liquid_pred_tf)
df3_ytest_liquid_f1_tf=f1_score(df3_ytest_liquid,df3_ytest_liquid_pred_tf)

df3_ytest_liquid_metrics=pd.DataFrame([
    {'metrics':'accuracy','models':'xgboost','score':df3_ytest_liquid_accuracy_xgb},
    {'metrics':'recall','models':'xgboost','score':df3_ytest_liquid_recall_xgb},
    {'metrics':'precision','models':'xgboost','score':df3_ytest_liquid_precision_xgb},
    {'metrics':'f1','models':'xgboost','score':df3_ytest_liquid_f1_xgb},

    {'metrics':'accuracy','models':'tensorflow','score':df3_ytest_liquid_accuracy_tf},
    {'metrics':'recall','models':'tensorflow','score':df3_ytest_liquid_recall_tf},
    {'metrics':'precision','models':'tensorflow','score':df3_ytest_liquid_precision_tf},
    {'metrics':'f1','models':'tensorflow','score':df3_ytest_liquid_f1_tf},
])

df3_ytest_liquid_metrics
```

Out[64]:

	metrics	models	score
0	accuracy	xgboost	0.999048
1	recall	xgboost	0.999432
2	precision	xgboost	0.998766
3	f1	xgboost	0.999099
4	accuracy	tensorflow	0.993857
5	recall	tensorflow	0.993195
6	precision	tensorflow	0.995167
7	f1	tensorflow	0.994180

In [65]:

```
plt.figure(figsize=(8,5))
ax=plt.gca()
sns.barplot(x='metrics', y='score',hue='models',data=df3_ytest_liquid_metrics)
plt.title("Models' Metrics Comparasion",fontsize=14)
plt.xlabel("Metrics",fontsize=12)
plt.ylabel("Score",fontsize=12)
plt.legend(loc='lower right')
plt.ylim(0.9,1)
sns.despine()
```



In [29]:

```
# we'd better normalize features first
num_features=['time', 'ambient_pressure', 'release_rate', 'humidity',
                'p01', 'p02', 'p03', 'p04', 'pt_01', 'pt_02',
                'pt_03', 'pt_04', 'wind_direction_high', 'wind_direction_low',
                'wind_speed_high', 'wind_speed_low', 'tt', 'x', 'y']

scaler=StandardScaler()
df3_xtrain_solid_scale=df3_xtrain_solid.copy()
df3_xtest_solid_scale=df3_xtest_solid.copy()

for feature in num_features:

    # fit scaler
    df3_xtrain_solid_scale[feature]=scaler.fit_transform(df3_xtrain_solid_scale[[feature

    # use the same scaler to transform test data
    df3_xtest_solid_scale[feature]=scaler.transform(df3_xtest_solid_scale[[feature]])

df3_xtest_solid_scale.head(2)
```

Out[29]:

	time	ambient_pressure	release_rate	humidity	purge	obstacles	vent panel	p
460436	-0.171071	0.970818	0.994948	0.921757	0	1	1	0.9017
423601	-1.355244	0.983050	0.994948	0.909962	0	1	1	1.1211

2 rows × 26 columns

In [30]:

```
df3_xtrain_solid_scale['sealing_low-level vent']=df3_xtrain_solid_scale['sealing_low-lev
df3_xtrain_solid_scale['sealing_low-level vent and opening to vent mast']=df3_xtrain_sol
df3_xtrain_solid_scale['sealing_none']=df3_xtrain_solid_scale['sealing_none'].astype(int

df3_xtest_solid_scale['sealing_low-level vent']=df3_xtest_solid_scale['sealing_low-level
df3_xtest_solid_scale['sealing_low-level vent and opening to vent mast']=df3_xtest_solid
df3_xtest_solid_scale['sealing_none']=df3_xtest_solid_scale['sealing_none'].astype(int)
```

In [31]:

```
df3_xtrain_solid_scale_train,df3_xtrain_solid_scale_val, df3_ytrain_solid_train,df3_ytra
    df3_xtrain_solid_scale, df3_ytrain_solid,test_size=0.25,random_state=42,stratify=df3

df3_xtrain_solid_scale_train.shape, df3_ytrain_solid_train.shape,df3_xtrain_solid_scale_
```

Out[31]:

```
((262265, 26), (262265,), (87422, 26), (87422,))
```

In [32]:

```
df_tmp=df3_xtrain_solid_scale_train.copy()
df_tmp['solid_oxygen']=df3_ytrain_solid_train

df_tmp.head()
```

Out[32]:

	time	ambient_pressure	release_rate	humidity	purge	obstacles	vent panel	
201764	1.181843	-1.243264	0.671755	0.930407	0	0	0	-0.746
178229	0.425355	-1.231032	0.671755	0.932766	0	0	0	-0.665
350432	-0.967705	0.954509	0.533244	0.616667	1	1	0	1.195
155371	-0.310231	-1.239186	0.671755	0.940628	0	0	0	-0.563
213512	1.559685	-1.251418	0.671755	0.935124	0	0	0	-0.783

5 rows × 27 columns

In [33]:

```
df_tmp_0=df_tmp[df_tmp['solid_oxygen']==0].copy()
df_tmp_1=df_tmp[df_tmp['solid_oxygen']==1].copy()

df_tmp_01=pd.concat([df_tmp_0.sample(63000,random_state=42),df_tmp_1.sample(63000,random
                    ignore_index=True)

df3_xtrain_solid_scale_train=df_tmp_01.drop('solid_oxygen', axis=1)
df3_ytrain_solid_scale_train=df_tmp_01['solid_oxygen']

df3_xtrain_solid_scale_train.shape, df3_ytrain_solid_scale_train.shape,df3_xtrain_solid_
```

Out[33]:

```
((126000, 26), (126000,), (87422, 26), (87422,))
```

In [34]:

```
tf.keras.backend.clear_session()
tf_df3_solid_bestmodel = tf.keras.models.Sequential([

    tf.keras.layers.Dense(64, activation='relu',input_shape=[df3_xtrain_solid_scale.shap
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(1, activation="sigmoid")

])
```

In [35]:

```
tf_df3_solid_bestmodel.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 64)                1728      
                                                                 
 dense_1 (Dense)             (None, 32)                2080      
                                                                 
 dense_2 (Dense)             (None, 8)                 264       
                                                                 
 dense_3 (Dense)             (None, 1)                 9         
                                                                 
=================================================================
Total params: 4,081
Trainable params: 4,081
Non-trainable params: 0
_________________________________________________________________
```

In [36]:

```
plot_model(tf_df3_solid_bestmodel, to_file='tf_model_3csv_solid.png', show_shapes=True)
```

```
You must install pydot (`pip install pydot`) and install graphviz (see in
structions at https://graphviz.gitlab.io/download/) (https://graphviz.git
lab.io/download/)) for plot_model to work.
```

In [37]:

```
# train model

optimizer=tf.keras.optimizers.SGD(learning_rate=1e-3)
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0.0001, pati

tf_df3_solid_bestmodel.compile(loss='binary_crossentropy',
                optimizer=optimizer,
                metrics=['accuracy'])

tf_df3_solid_bestmodel.fit(df3_xtrain_solid_scale_train,df3_ytrain_solid_scale_train, ep
          validation_data=(df3_xtrain_solid_scale_val,df3_ytrain_solid_val), callbacks=[
```

```
Epoch 1/100
3938/3938 [==============================] - 13s 3ms/step - loss: 0.44
99 - accuracy: 0.7697 - val_loss: 0.4505 - val_accuracy: 0.7461
Epoch 2/100
3938/3938 [==============================] - 11s 3ms/step - loss: 0.33
50 - accuracy: 0.8744 - val_loss: 0.3134 - val_accuracy: 0.8673
Epoch 3/100
3938/3938 [==============================] - 10s 3ms/step - loss: 0.23
22 - accuracy: 0.9217 - val_loss: 0.2113 - val_accuracy: 0.9143
Epoch 4/100
3938/3938 [==============================] - 11s 3ms/step - loss: 0.16
58 - accuracy: 0.9465 - val_loss: 0.1569 - val_accuracy: 0.9444
Epoch 5/100
3938/3938 [==============================] - 10s 3ms/step - loss: 0.13
16 - accuracy: 0.9590 - val_loss: 0.1300 - val_accuracy: 0.9516
Epoch 6/100
3938/3938 [==============================] - 10s 3ms/step - loss: 0.11
24 - accuracy: 0.9646 - val_loss: 0.1126 - val_accuracy: 0.9581
Epoch 7/100
```

In [38]:

```
df3_solid_history=pd.DataFrame(tf_df3_solid_bestmodel.history.history)
df3_solid_history
```

Out[38]:

	loss	accuracy	val_loss	val_accuracy
0	0.449916	0.769690	0.450465	0.746139
1	0.334960	0.874381	0.313368	0.867287
2	0.232215	0.921714	0.211290	0.914335
3	0.165803	0.946548	0.156862	0.944350
4	0.131606	0.958960	0.130010	0.951625
5	0.112391	0.964603	0.112637	0.958100
6	0.099656	0.967302	0.104911	0.961291
7	0.090627	0.969587	0.092704	0.964540
8	0.083747	0.971341	0.087926	0.965958
9	0.078056	0.972683	0.083427	0.967125
10	0.073198	0.974325	0.073598	0.970545
11	0.068933	0.975786	0.069176	0.973393
12	0.065303	0.977063	0.067196	0.973771
13	0.061938	0.978643	0.058772	0.977843
14	0.059145	0.980040	0.062229	0.976390
15	0.056534	0.981270	0.059442	0.978095
16	0.054183	0.982397	0.054923	0.980154
17	0.052207	0.983667	0.053312	0.980611
18	0.050224	0.984460	0.052008	0.981252
19	0.048482	0.985079	0.052072	0.981275
20	0.046884	0.986056	0.049825	0.982373
21	0.045350	0.986500	0.045468	0.983917
22	0.043888	0.987056	0.043594	0.985370
23	0.042598	0.987484	0.043388	0.984889
24	0.041364	0.988016	0.040418	0.986125
25	0.040177	0.988397	0.041178	0.985759
26	0.039029	0.988794	0.039086	0.987189
27	0.038117	0.989087	0.037985	0.986754
28	0.037201	0.989167	0.037016	0.987360
29	0.036372	0.989563	0.037214	0.987566
30	0.035596	0.989897	0.038745	0.986834
31	0.034800	0.990198	0.033829	0.988458
32	0.034097	0.990254	0.038013	0.986628
33	0.033528	0.990436	0.032874	0.989259
34	0.032864	0.990611	0.033774	0.988184
35	0.032272	0.990817	0.032346	0.989190
36	0.031632	0.990913	0.032498	0.988973

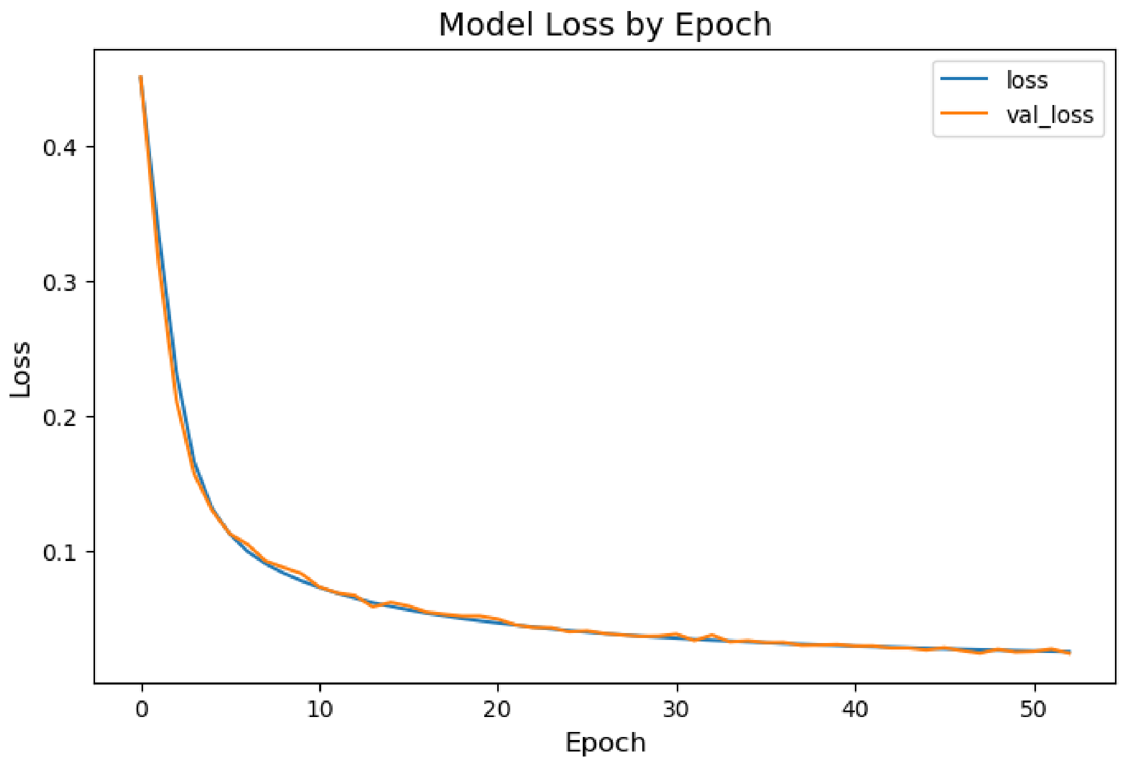
	loss	accuracy	val_loss	val_accuracy
37	0.031253	0.991016	0.030410	0.989511
38	0.030733	0.991254	0.030584	0.989373
39	0.030264	0.991270	0.030967	0.989328
40	0.029760	0.991357	0.029797	0.989453
41	0.029334	0.991460	0.029854	0.989362
42	0.029014	0.991500	0.028351	0.990163
43	0.028594	0.991754	0.028361	0.990094
44	0.028186	0.991913	0.026966	0.990815
45	0.027836	0.992063	0.028671	0.989968
46	0.027494	0.992048	0.026402	0.991352
47	0.027123	0.992238	0.024508	0.992176
48	0.026855	0.992286	0.027519	0.991226
49	0.026503	0.992556	0.025353	0.991398
50	0.026212	0.992603	0.025815	0.991341
51	0.025967	0.992603	0.027715	0.990151
52	0.025643	0.992627	0.024453	0.991718

In [39]:

```
plt.figure(figsize=(8,5))
ax=plt.gca()
df3_solid_history[['loss','val_loss']].plot(kind='line',ax=ax)
plt.title('Model Loss by Epoch',fontsize=14)
plt.xlabel("Epoch",fontsize=12)
plt.ylabel("Loss",fontsize=12)
```

Out[39]:

```
Text(0, 0.5, 'Loss')
```

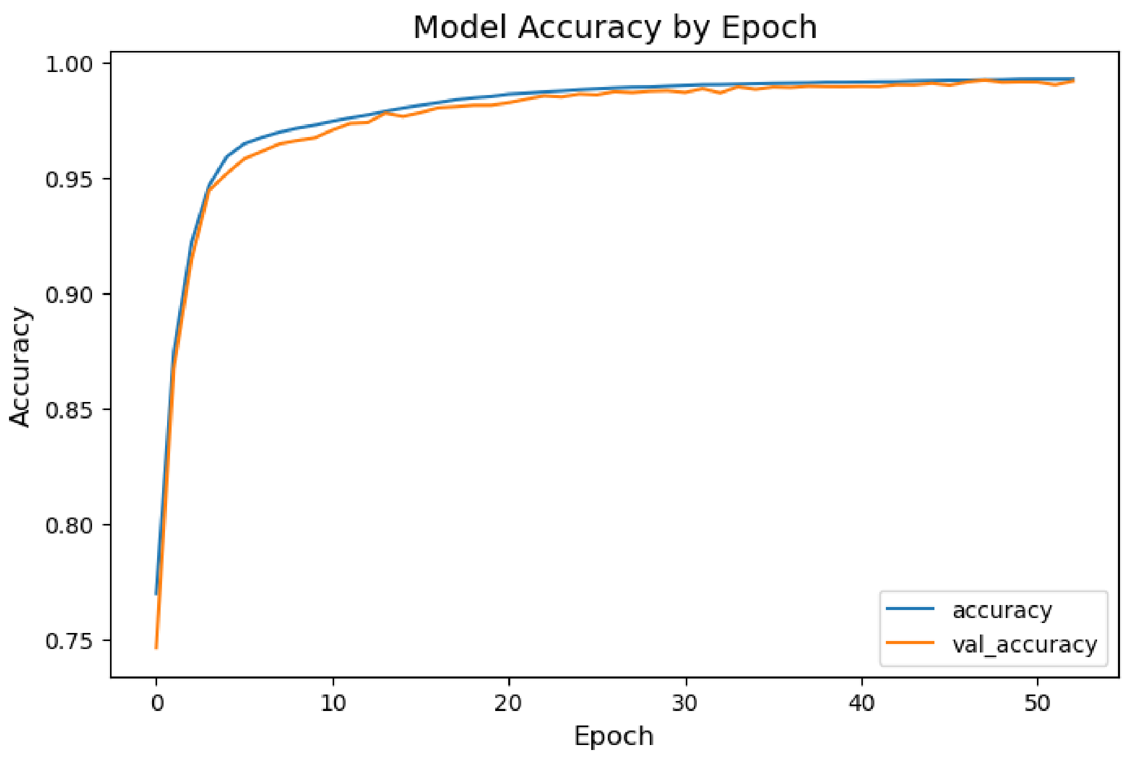


In [40]:

```
plt.figure(figsize=(8,5))
ax=plt.gca()
df3_solid_history[['accuracy','val_accuracy']].plot(kind='line',ax=ax)
plt.title('Model Accuracy by Epoch',fontsize=14)
plt.xlabel("Epoch",fontsize=12)
plt.ylabel("Accuracy",fontsize=12)
```

Out[40]:

```
Text(0, 0.5, 'Accuracy')
```



In [41]:

```
df3_ytest_solid_pred_tf = np.where(tf_df3_solid_bestmodel.predict(df3_xtest_solid_scale)
```

```
3643/3643 [==============================] - 4s 1ms/step
```

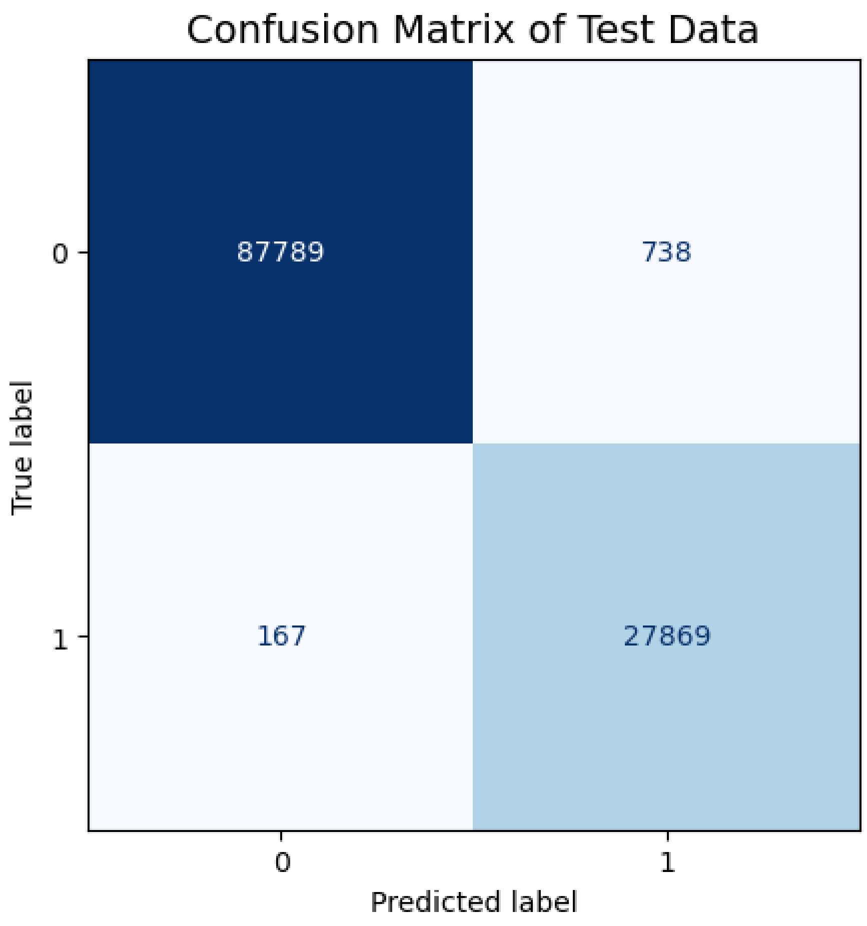
In [42]:

```
print(classification_report(df3_ytest_solid,df3_ytest_solid_pred_tf,digits=5))
```

```
              precision    recall  f1-score   support

           0    0.99810   0.99166   0.99487     88527
           1    0.97420   0.99404   0.98402     28036

    accuracy                        0.99224    116563
   macro avg    0.98615   0.99285   0.98945    116563
weighted avg    0.99235   0.99224   0.99226    116563
```

In [43]:

```
# confusion matrix

plt.figure(figsize=(5,5))
ax=plt.gca()
ConfusionMatrixDisplay.from_predictions(df3_ytest_solid, df3_ytest_solid_pred_tf,
                                      ax=ax,cmap=plt.get_cmap('Blues'),colorbar=False)
plt.title("Confusion Matrix of Test Data",fontsize=14)
plt.show()
```



In [44]:

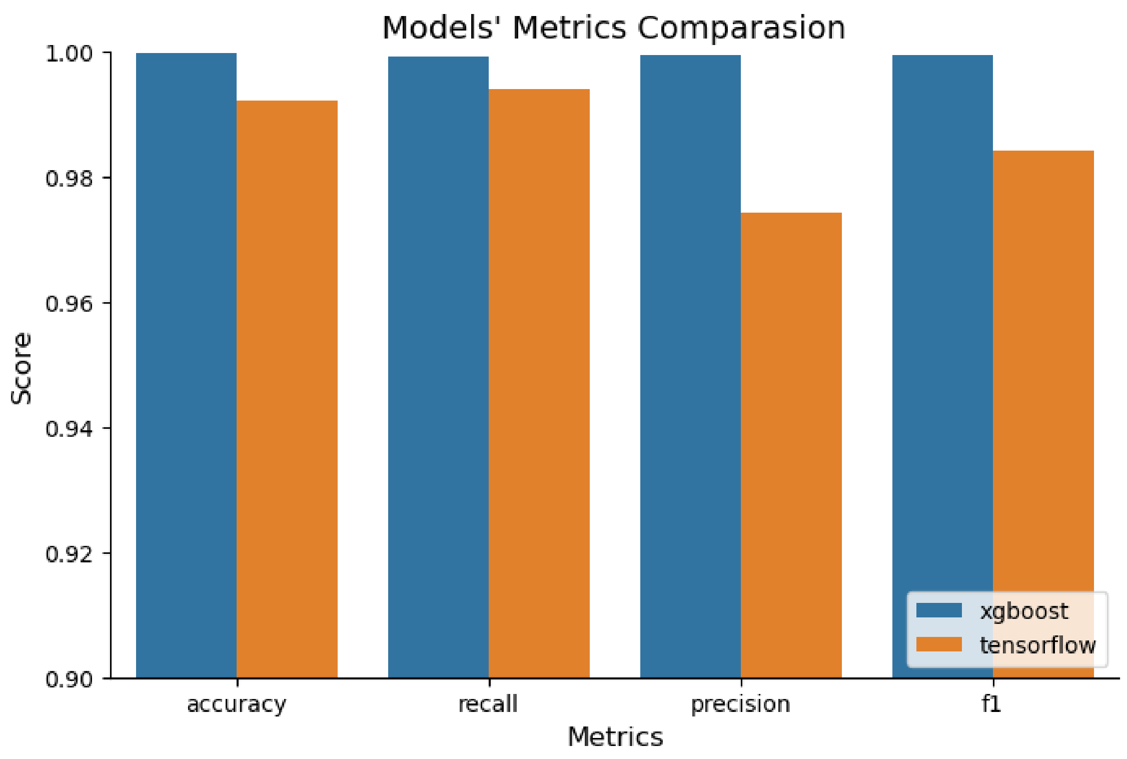
```
df3_ytest_solid_accuracy_xgb=accuracy_score(df3_ytest_solid,df3_ytest_solid_pred_xgb)
df3_ytest_solid_recall_xgb=recall_score(df3_ytest_solid,df3_ytest_solid_pred_xgb)
df3_ytest_solid_precision_xgb=precision_score(df3_ytest_solid,df3_ytest_solid_pred_xgb)
df3_ytest_solid_f1_xgb=f1_score(df3_ytest_solid,df3_ytest_solid_pred_xgb)

df3_ytest_solid_accuracy_tf=accuracy_score(df3_ytest_solid,df3_ytest_solid_pred_tf)
df3_ytest_solid_recall_tf=recall_score(df3_ytest_solid,df3_ytest_solid_pred_tf)
df3_ytest_solid_precision_tf=precision_score(df3_ytest_solid,df3_ytest_solid_pred_tf)
df3_ytest_solid_f1_tf=f1_score(df3_ytest_solid,df3_ytest_solid_pred_tf)

df3_ytest_solid_metrics=pd.DataFrame([
    {'metrics':'accuracy','models':'xgboost','score':df3_ytest_solid_accuracy_xgb},
    {'metrics':'recall','models':'xgboost','score':df3_ytest_solid_recall_xgb},
    {'metrics':'precision','models':'xgboost','score':df3_ytest_solid_precision_xgb},
    {'metrics':'f1','models':'xgboost','score':df3_ytest_solid_f1_xgb},

    {'metrics':'accuracy','models':'tensorflow','score':df3_ytest_solid_accuracy_tf},
    {'metrics':'recall','models':'tensorflow','score':df3_ytest_solid_recall_tf},
    {'metrics':'precision','models':'tensorflow','score':df3_ytest_solid_precision_tf},
    {'metrics':'f1','models':'tensorflow','score':df3_ytest_solid_f1_tf},
])

df3_ytest_solid_metrics
```

Out[44]:

	metrics	models	score
0	accuracy	xgboost	0.999683
1	recall	xgboost	0.999144
2	precision	xgboost	0.999536
3	f1	xgboost	0.999340
4	accuracy	tensorflow	0.992236
5	recall	tensorflow	0.994043
6	precision	tensorflow	0.974202
7	f1	tensorflow	0.984023

In [45]:

```
plt.figure(figsize=(8,5))
ax=plt.gca()
sns.barplot(x='metrics', y='score',hue='models',data=df3_ytest_solid_metrics)
plt.title("Models' Metrics Comparasion",fontsize=14)
plt.xlabel("Metrics",fontsize=12)
plt.ylabel("Score",fontsize=12)
plt.legend(loc='lower right')
plt.ylim(0.9,1)
sns.despine()
```



In [ ]:

Bibliography

Aaneby, J., Gjesdal, T., and Voie, y. A. (2021). Large scale leakage of liquid hydrogen (lh2)–tests related to bunkering and maritime use of liquid hydrogen.

Aaneby, J. and Hafskjold, M. (2021). Safety and environmental risks related to hydrogen as an energy carrier. *Journal of Loss Prevention in the Process Industries*, 70:104318.

Bengio, Y., Goodfellow, I., and Courville, A. (2016). *Deep learning*. MIT Press.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Chi, H., Wang, H., and Hu, J. (2018). Numerical simulation of lng spills over water based on shallow water equations. *Marine Pollution Bulletin*, 127:563–572.

Debe, M. K. (2012). Electrocatalyst approaches and challenges for automotive fuel cells. *Nature*, 486:43–51.

Doc, I. (2007). 75/07/e/rev. *Determination of safety distances. European industrial gases association.*

Ferrari, F. (2022). Data analytics for hydrogen safety: Prediction of liquid hydrogen release characteristics. Master's thesis, NTNU.

Gavelli, F., Agostini, F., and D'Alessandro, F. (2016). Modeling and simulation of a liquefied natural gas spill: a case study. *Journal of Loss Prevention in the Process Industries*, 43:92–98.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial networks. In *Advances in neural information processing systems*, pages 2672–2680.

Hasanzadeh, E. and Abbassi, R. (2017). Numerical investigation of liquid hydrogen pool spreading over an inclined flat plate with focused attention on the hot spots. *International Journal of Hydrogen Energy*, 42(28):17973–17987.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science Business Media.

Hastie, T., Tibshirani, R., and Wainwright, M. (2015). *Statistical learning with sparsity: the lasso and generalizations*. CRC Press.

Hinton, G. E., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., and Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97.

Hong, W., Liu, L., and Wen, Z. (2021). Study on the safety of hydrogen fuel cell vehicles in tunnel fires based on numerical simulation. *International Journal of Hydrogen Energy*, 46:5104–5114.

Hu, J.-W. et al. (2015). Progress and future directions in solid oxide fuel cells: An overview. *International Journal of Energy Research*, 39:897–918.

Huang, S., Zhang, Y., and Wang, L. (2019). Safety evaluation of hydrogen fuel cell vehicle under tunnel fires: A review. *International Journal of Hydrogen Energy*, 44:4986–4995.

Jia, W., Li, X., Zhao, M., and Jiang, Y. (2019). Numerical simulation of the release and dispersion of hydrogen from an accidental spill in an industrial park. *International Journal of Hydrogen Energy*, 44(13):6867–6879.

Karypis, G. (2004). Scalable parallel algorithms for clustering and association mining. *IEEE Transactions on Knowledge and Data Engineering*, 16(8):880–892.

King, R. D. (2010). Apache mahout: A scalable machine learning and data mining library. In *Proceedings of the 2010 IEEE International Conference on Data Mining Workshops*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Li, X., Li, F., and Li, H. (2020). Safety of hydrogen storage and transportation: Recent advances and future challenges. *International Journal of Hydrogen Energy*, 45:14203–14214.

Li, Z., Pan, X., Meng, X., and Ma, J. (2012). Study on the harm effects of releases from liquid hydrogen tank by consequence modeling. *International journal of hydrogen energy*, 37(22):17624–17629.

Mazzotti, M. and Bidini, G. (2020). Hydrogen: A possible solution for the future of energy. *Chemical Engineering Transactions*, 80:73–78.

Mitchell, T. (1997). *Machine learning*. McGraw Hill.

Ng, A. (2018). *Machine learning yearning*.

O'Connor, M. (2019). Hydrogen storage for energy applications: Past, present and future. *International Journal of Hydrogen Energy*, 44:25731–25748.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press.

Tian, T., Gao, B., Guo, X., Chen, C., and He, T. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Wang, H. and Yuan, Z. (2014). Recent advances in the electrocatalysis of oxygen reduction reaction for fuel cell applications. *Science China Chemistry*, 57:704–715.

Zeng, X., Yang, G., Song, X., and Yin, J. (2018). Numerical simulation of jet flame characteristics of hydrogen release from liquefied hydrogen tank. *International Journal of Hydrogen Energy*, 43(12):6211–6222.

Zhang, Y., Wu, H., He, Y., Zhang, X., and Gao, Z. (2019). Numerical simulation of a liquid hydrogen spill in a confined space. *International Journal of Hydrogen Energy*, 44(15):7743–7753.

Zheng, J., Liu, X., Xu, P., Liu, P., Zhao, Y., and Yang, J. (2012a). Development of high pressure gaseous hydrogen storage technologies. *International Journal of Hydrogen Energy*, 37(1):1048–1057.

Zheng, Y., Zhang, Q., and Zhang, J. (2012b). Hydrogen as an energy carrier: Prospects and challenges. *Renewable and Sustainable Energy Reviews*, 16:3024–3033.

Zhu, S., Hu, B., Chen, W., Xu, C., Liu, X., and Chen, X. (2017). Analysis of the hazards and risk mitigation measures for hydrogen refueling stations. *International Journal of Hydrogen Energy*, 42(29):18608–18616.