

# **ELEVATOR MAINTENANCE PREDICTOR**

## **A MINI PROJECT REPORT**

**Submitted by**

**JODERICK SHERWIN J      220701109**

**KARTHIGA R                      220701119**

*In partial fulfillment for the award of the degree of*

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)**

**THANDALAM**

**CHENNAI-602105**

**2023 - 24**

## **BONAFIDE CERTIFICATE**

Certified that this project report “ELEVATOR MAINTAINANCE PREDICTOR” is the bonafide work of “**JODERICK SHERWIN J (220701109), KARTHIGA R (220701119)**, who carried out the project work under my supervision.

**Submitted for the Practical Examination held on** \_\_\_\_\_

**SIGNATURE**

**Dr.R.SABITHA**

**Professor and II Year Academic Head  
Computer Science and Engineering,  
Rajalakshmi Engineering College  
(Autonomous),  
Thandalam, Chennai - 602 105**

**SIGNATURE**

**Dr.G.DHARANI DEVI**

**Associate Professor,  
Computer Science and Engineering,  
Rajalakshmi Engineering College  
(Autonomous),  
Thandalam, Chennai - 602 105**

**INTERNAL EXAMINER**

**EXTERNAL EXAMINER**

# Table of Contents

1. Introduction
  1. Introduction
  2. Objectives
  3. Modules
2. Survey of Technologies
  1. Software Description
  2. Languages
3. Requirements and Analysis
  1. Requirement Specification
  2. Hardware and Software Requirements
  3. Architecture Diagram
  4. ER Diagram
  5. Class Diagram
  6. Sequence Diagram
  7. Normalization
4. Program Code
  1. Model.py
  2. Elevator\_Predictor.py
  3. Output
5. Results and Discussion
  1. Model Prediction Evaluation
  2. System Usability and User Feedback
  3. Implications for Proactive Maintenance Strategies
  4. Discussion of Findings
  5. Future Directions
6. Conclusion
  1. Key Accomplishments
  2. Contributions to Elevator Maintenance Industry
  3. Future Outlook
  4. Final Remarks

# 1. INTRODUCTION

## 1.1 Introduction

Elevators are critical components in buildings, providing essential vertical transportation. Ensuring their reliability and safety is paramount, as elevator malfunctions can lead to significant inconveniences, costly repairs, and, in worst-case scenarios, safety hazards. Traditional elevator maintenance often relies on periodic inspections and reactive repairs after a failure occurs. However, this approach can be inefficient and may not prevent unexpected breakdowns.

The Elevator Maintenance Predictor project aims to revolutionize the way elevator maintenance is approached by leveraging data-driven predictive maintenance techniques. By predicting elevator vibration levels, which are indicative of potential mechanical issues, the system enables proactive maintenance scheduling. This helps in reducing downtime, minimizing repair costs, and enhancing the overall safety and reliability of elevator operations.

The core of this project is a machine learning model that analyzes sensor data inputs to predict the vibration levels of elevators. The model is trained on historical data to recognize patterns and anomalies that precede maintenance issues. The project also features a graphical user interface (GUI) for easy data input and interaction, and a robust database for storing and managing sensor data and predictions.

The Elevator Maintenance Predictor is designed to be a comprehensive solution, encompassing data collection, real-time prediction, data storage, and model retraining capabilities. It provides an intuitive and user-friendly interface, making it accessible to maintenance personnel and building managers, regardless of their technical expertise.

## 1.2 Objectives

The Elevator Maintenance Predictor project aims to provide a comprehensive solution for proactive maintenance of elevators through predictive analytics. The detailed objectives of the project are outlined as follows:

### 1. Data Collection

- **Objective:** Collect relevant sensor data that impacts elevator performance.
- **Details:**
  - Gather data on revolutions, humidity, and various sensor readings such as **x1**, **x2**, **x3**, **x4**, and **x5**.
  - Ensure the data is accurately captured and formatted for subsequent analysis and storage.
  - Integrate data collection into an easy-to-use graphical user interface (GUI) for seamless data input by users.

### 2. Prediction

- **Objective:** Predict elevator vibration levels using a trained machine learning model.
- **Details:**
  - Utilize historical data to train a Linear Regression model that can predict the vibration levels based on input features.
  - Provide real-time predictions to enable proactive maintenance scheduling.

- Implement mechanisms to ensure the accuracy and reliability of predictions, such as evaluating the model's Mean Absolute Error (MAE).

### 3. Data Management

- **Objective:** Efficiently store and manage input data and prediction results.
- **Details:**
  - Use a MySQL database to store sensor data and prediction outcomes.
  - Design the database schema to facilitate easy retrieval and update of data.
  - Ensure data integrity and security, enabling reliable access for analysis and model retraining.

### 4. User Interface

- **Objective:** Develop a user-friendly GUI for data input and interaction.
- **Details:**
  - Design an intuitive interface using PyQt5 for users to input sensor data and view predictions.
  - Include features such as clear labels, input validation, and easy navigation.
  - Provide additional functionalities like model retraining through the GUI, making it accessible to users with varying levels of technical expertise.

### 5. Model Retraining

- **Objective:** Enable continuous improvement of the machine learning model.
- **Details:**
  - Allow the model to be retrained with new data collected over time to adapt to changing patterns and improve prediction accuracy.
  - Implement a retraining feature in the GUI to facilitate easy model updates.
  - Monitor model performance metrics, such as MAE, to assess the effectiveness of retraining and ensure ongoing accuracy.

### 6. Proactive Maintenance

- **Objective:** Enhance elevator safety and reliability through proactive maintenance strategies.
- **Details:**
  - Use predicted vibration levels to identify potential mechanical issues before they become critical.
  - Implement threshold-based alerts to notify maintenance personnel of high vibration levels that require attention.
  - Reduce unplanned downtime and maintenance costs by addressing issues proactively.

### 7. Scalability and Flexibility

- **Objective:** Design the system to be scalable and adaptable to different environments.

- **Details:**
  - Ensure the system can handle varying amounts of data and different types of sensors.
  - Design the database and model to be flexible, accommodating future expansions or changes in data collection parameters.
  - Implement a modular architecture to allow easy integration with other systems or additional functionalities.

By achieving these objectives, the Elevator Maintenance Predictor project aims to provide a robust, data-driven solution that enhances the efficiency, safety, and reliability of elevator operations. The system's ability to predict potential issues and facilitate proactive maintenance ensures that elevators remain operational and safe, reducing the risk of unexpected failures and associated costs.

### 1.3 Modules

The Elevator Maintenance Predictor project is structured into several interconnected modules, each serving a distinct purpose and contributing to the overall functionality of the system.

#### 1. Data Collection Module

- **Objective:** Gather relevant sensor data that impacts elevator performance.
- **Details:**
  - **Sensor Data Inputs:** Collects data such as revolutions, humidity, and various sensor readings (**x1**, **x2**, **x3**, **x4**, **x5**) from the elevator's sensors.
  - **Input Validation:** Ensures the correctness and validity of the input data, performing checks to prevent erroneous entries.
  - **Integration with GUI:** Seamlessly integrates with the graphical user interface (GUI) to enable users to input sensor data easily.

#### 2. Prediction Module

- **Objective:** Predict elevator vibration levels using a trained machine learning model.
- **Details:**
  - **Machine Learning Model:** Utilizes a Linear Regression model trained on historical data to predict vibration levels based on input features.
  - **Real-time Predictions:** Provides instantaneous predictions, allowing for proactive maintenance scheduling.
  - **Prediction Accuracy Evaluation:** Assesses the accuracy and reliability of predictions through metrics like Mean Absolute Error (MAE).

#### 3. Data Management Module

- **Objective:** Efficiently store and manage input data and prediction results.
- **Details:**
  - **Database Integration:** Utilizes a MySQL database to store sensor data and prediction outcomes securely.

- **Schema Design:** Designs the database schema to facilitate easy retrieval and manipulation of data, ensuring data integrity.
- **Data Handling:** Implements mechanisms for storing, updating, and querying data, enabling seamless access for analysis and model retraining.

#### 4. User Interface (GUI) Module

- **Objective:** Develop a user-friendly interface for data input and interaction.
- **Details:**
  - **Intuitive Design:** Designs an intuitive GUI using PyQt5, featuring clear labels, input validation, and easy navigation.
  - **User Input:** Allows users to input sensor data effortlessly through user-friendly input fields.
  - **Visualization:** Presents prediction results and relevant information in a visually appealing and comprehensible manner to users.

#### 5. Model Retraining Module

- **Objective:** Enable continuous improvement of the machine learning model.
- **Details:**
  - **Retraining Mechanism:** Implements functionality to retrain the machine learning model with new data collected over time.
  - **Adaptation to Changes:** Enables the model to adapt to changing patterns and improve prediction accuracy through periodic retraining.
  - **Performance Monitoring:** Monitors model performance metrics, such as MAE, to assess the effectiveness of retraining and ensure ongoing accuracy.

#### 6. Proactive Maintenance Module

- **Objective:** Enhance elevator safety and reliability through proactive maintenance strategies.
- **Details:**
  - **Early Warning System:** Uses predicted vibration levels to identify potential mechanical issues before they escalate, enabling proactive maintenance actions.
  - **Threshold-based Alerts:** Notifies maintenance personnel of high vibration levels that exceed predefined thresholds, prompting timely intervention.
  - **Cost Reduction:** Reduces unplanned downtime and maintenance costs by addressing issues proactively, ultimately enhancing operational efficiency and safety.

#### 7. Scalability and Flexibility Module

- **Objective:** Design the system to be scalable and adaptable to different environments.
- **Details:**
  - **Flexible Architecture:** Implements a modular architecture that allows for easy integration with other systems and accommodates future expansions or changes in data collection parameters.

- **Parameterization:** Designs the system to handle varying amounts and types of sensor data, ensuring scalability and flexibility in different scenarios.
- **Compatibility:** Ensures compatibility with different elevator systems and environments, facilitating widespread adoption and deployment.

## 2. SURVEY OF TECHNOLOGIES

### 2.1 Software Description

The Elevator Maintenance Predictor project utilizes a combination of software tools and technologies to achieve its objectives, encompassing data collection, predictive modeling, database management, user interface development, and more. The following software components play integral roles in the functionality and implementation of the system:

#### 2.1.1 Python

- **Description:** Python serves as the primary programming language for developing the backend logic of the Elevator Maintenance Predictor project.
- **Key Features:**
  - **Ease of Use:** Python's simple syntax and readability make it well-suited for rapid development and prototyping.
  - **Rich Ecosystem:** Python offers a vast ecosystem of libraries and frameworks for various tasks, including machine learning (e.g., scikit-learn), data manipulation (e.g., NumPy, Pandas), and GUI development (e.g., PyQt5).
  - **Platform Independence:** Python is platform-independent, allowing the project to run seamlessly on different operating systems.
- **Role in the Project:**
  - Data preprocessing, including feature extraction and manipulation.
  - Training and deployment of machine learning models for predicting elevator vibration levels.
  - Integration with external libraries and frameworks for database management, GUI development, and more.

#### 2.1.2 MySQL

- **Description:** MySQL is an open-source relational database management system (RDBMS) widely used for storing and managing structured data.
- **Key Features:**
  - **Reliability:** MySQL is known for its reliability, scalability, and performance, making it suitable for handling large datasets and concurrent connections.
  - **SQL Support:** MySQL supports the Structured Query Language (SQL), providing a robust set of tools for data manipulation, retrieval, and management.
  - **Community Support:** As an open-source project, MySQL benefits from a vibrant community of developers and contributors, ensuring continuous improvement and support.



- **Role in the Project:**
  - Storage and management of sensor data, prediction results, and other relevant information related to elevator maintenance.
  - Facilitation of data retrieval, manipulation, and analysis for model training, prediction, and evaluation purposes.

### 2.1.3 PyQt5

- **Description:** PyQt5 is a set of Python bindings for the Qt application framework, enabling the development of cross-platform graphical user interfaces (GUIs).
- **Key Features:**
  - **Powerful GUI Toolkit:** PyQt5 provides a comprehensive set of tools and widgets for creating interactive and visually appealing desktop applications.
  - **Integration with Python:** PyQt5 seamlessly integrates with Python, allowing developers to leverage Python's flexibility and ease of use in GUI development.
  - **Cross-Platform Compatibility:** GUIs developed using PyQt5 are cross-platform, meaning they can run on various operating systems without modification.
- **Role in the Project:**
  - Development of an intuitive and user-friendly interface for users to input sensor data, view prediction results, and interact with the system.
  - Implementation of features such as input validation, visualization of prediction outcomes, and seamless navigation within the application.

### 2.1.4 scikit-learn

- **Description:** scikit-learn is a popular machine learning library for Python, providing simple and efficient tools for data mining and analysis.
- **Key Features:**
  - **Wide Range of Algorithms:** scikit-learn offers a variety of machine learning algorithms and tools for tasks such as classification, regression, clustering, and dimensionality reduction.
  - **Ease of Use:** The library is designed with simplicity and usability in mind, making it accessible to both beginners and experienced practitioners.
  - **Integration with Other Libraries:** scikit-learn integrates seamlessly with other Python libraries such as NumPy and Pandas, facilitating data preprocessing and manipulation.
- **Role in the Project:**
  - Training and deployment of machine learning models for predicting elevator vibration levels based on sensor data inputs.
  - Evaluation of model performance metrics, such as Mean Absolute Error (MAE), to assess prediction accuracy and reliability.

### 2.1.5 NumPy and Pandas

- **Description:** NumPy and Pandas are Python libraries widely used for numerical computing and data manipulation, respectively.
- **Key Features:**
  - **Efficient Data Structures:** NumPy provides powerful arrays and matrices for numerical computations, while Pandas offers DataFrame objects for handling structured data.
  - **Data Manipulation Tools:** Both libraries offer a rich set of functions and methods for data manipulation, including indexing, slicing, filtering, and aggregation.
  - **Interoperability:** NumPy and Pandas seamlessly integrate with other Python libraries, enabling seamless data exchange and analysis.
- **Role in the Project:**
  - Data preprocessing tasks such as feature extraction, normalization, and transformation.
  - Handling and manipulation of sensor data, prediction results, and other structured information within the system.

These software components collectively form the backbone of the Elevator Maintenance Predictor project, enabling the development of a robust, scalable, and user-friendly solution for proactive elevator maintenance.

## 2.2 Languages

The Elevator Maintenance Predictor project utilizes two primary programming languages, SQL and Python, each serving distinct roles in the system's development and functionality.

### 2.2.1 SQL

- **Description:** SQL (Structured Query Language) is a domain-specific language used for managing relational databases. It provides a standardized way to interact with databases for tasks such as data definition, manipulation, retrieval, and administration.
- **Key Features:**
  - **Data Definition:** SQL allows for the creation, modification, and deletion of database schemas, tables, and other structures.
  - **Data Manipulation:** SQL provides commands for inserting, updating, and deleting data within tables, as well as querying data to retrieve specific information.
  - **Data Control:** SQL offers mechanisms for controlling access to data, enforcing data integrity constraints, and managing transactions.
- **Role in the Project:**
  - **Database Management:** SQL is used to define and manage the MySQL database that stores sensor data, prediction results, and other relevant information related to elevator maintenance.
  - **Data Retrieval and Manipulation:** SQL queries are employed to retrieve, update, and manipulate data within the database, facilitating tasks such as model training, prediction, and evaluation.

### 2.2.2 Python

- **Description:** Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. It is widely used in various domains, including web development, scientific computing, data analysis, artificial intelligence, and more.
- **Key Features:**
  - **Ease of Use:** Python's clean syntax and readability make it accessible to beginners and experienced developers alike.
  - **Rich Ecosystem:** Python boasts a vast ecosystem of libraries and frameworks for diverse tasks, including machine learning, data manipulation, web development, GUI development, and more.
  - **Platform Independence:** Python is platform-independent, meaning code written in Python can run on different operating systems without modification.
- **Role in the Project:**
  - **Backend Logic:** Python serves as the primary language for developing the backend logic of the Elevator Maintenance Predictor project, including data preprocessing, machine learning model training, prediction, evaluation, and more.
  - **GUI Development:** Python is used in conjunction with the PyQt5 library to develop the graphical user interface (GUI) for the application, enabling users to interact with the system through an intuitive and user-friendly interface.

By leveraging the capabilities of both SQL and Python, the Elevator Maintenance Predictor project achieves a synergistic integration of database management, data processing, predictive analytics, and user interface development, resulting in a comprehensive and efficient solution for proactive elevator maintenance.

### 3. REQUIREMENTS AND ANALYSIS

#### 3.1 Requirement Specification

The Requirement Specification outlines the functional and non-functional requirements of the Elevator Maintenance Predictor project, detailing the system's capabilities, constraints, and performance expectations.

##### 3.1.1 Functional Requirements

1. **Data Collection:**
  - The system shall collect sensor data inputs from elevator sensors, including revolutions, humidity, and sensor readings (**x1, x2, x3, x4, x5**).
  - The system shall validate input data to ensure correctness and integrity.
2. **Prediction:**
  - The system shall utilize a machine learning model to predict elevator vibration levels based on input sensor data.
  - The system shall provide real-time predictions to enable proactive maintenance scheduling.
3. **Data Management:**

- The system shall store sensor data, prediction results, and other relevant information in a MySQL database.
- The system shall facilitate data retrieval, manipulation, and analysis for model training, prediction, and evaluation purposes.

**4. User Interface (GUI):**

- The system shall feature a user-friendly GUI developed using PyQt5 for data input, visualization, and interaction.
- The GUI shall include input fields for sensor data, buttons for submitting data and retraining the model, and displays for prediction results.

**5. Model Retraining:**

- The system shall enable the retraining of the machine learning model with new data collected over time.
- The system shall periodically update the model to adapt to changing patterns and improve prediction accuracy.

**6. Proactive Maintenance:**

- The system shall use predicted vibration levels to identify potential mechanical issues before they escalate.
- The system shall notify maintenance personnel of high vibration levels that exceed predefined thresholds.

### **3.1.2 Non-Functional Requirements**

**1. Performance:**

- The system shall provide fast and efficient prediction capabilities, with minimal latency for real-time predictions.
- The system shall handle large volumes of sensor data and database transactions without significant performance degradation.

**2. Reliability:**

- The system shall demonstrate high reliability and robustness, with minimal risk of system failures or data corruption.
- The system shall maintain data integrity and consistency within the database, ensuring accurate prediction outcomes.

**3. Usability:**

- The GUI shall be intuitive and easy to use, catering to users with varying levels of technical expertise.
- The system shall provide clear and informative feedback to users regarding prediction results, data input validation, and system status.

**4. Security:**

- The system shall implement appropriate security measures to protect sensitive data stored in the database.

- The system shall enforce access control mechanisms to restrict unauthorized access to system functionalities and data.

**5. Scalability:**

- The system shall be scalable, capable of handling increased data loads and user demands as the system usage grows.
- The system architecture shall support horizontal scalability, allowing for the addition of more resources to accommodate higher workloads.

**6. Compatibility:**

- The system shall be compatible with different elevator systems and environments, supporting diverse sensor configurations and data formats.
- The system shall run seamlessly on multiple operating systems, including Windows, macOS, and Linux distributions.

By adhering to these functional and non-functional requirements, the Elevator Maintenance Predictor project aims to deliver a reliable, efficient, and user-friendly solution for proactive elevator maintenance, enhancing the safety, reliability, and operational efficiency of elevators in various settings.

## **3.2 Hardware and Software Requirements**

### **3.2.1 Hardware Requirements**

The hardware requirements for the Elevator Maintenance Predictor project are outlined below:

**1. Computer or Server:**

- A computer or server capable of running the software components of the system.
- Adequate processing power and memory to handle data processing, model training, and prediction tasks efficiently.

**2. Storage:**

- Sufficient storage space to store sensor data, prediction results, and other relevant information within the MySQL database.
- Additional storage may be required for storing machine learning models, application logs, and other system artifacts.

**3. Network Connectivity:**

- Internet connectivity may be required for accessing external resources, downloading software updates, and communicating with remote servers or APIs (if applicable).

### **3.2.2 Software Requirements**

The software requirements for the Elevator Maintenance Predictor project encompass various components and technologies used in system development, deployment, and operation:

**1. Operating System:**

- The system should be compatible with multiple operating systems, including:

- Windows (Windows 10 or later recommended)
- macOS (macOS 10.14 or later recommended)
- Linux distributions (Ubuntu, CentOS, etc.)

## 2. **Python:**

- Version: Python 3.x (e.g., Python 3.7, Python 3.8)
- Python serves as the primary programming language for developing the backend logic and GUI components of the system.

## 3. **MySQL Database:**

- Version: MySQL 5.x or later
- MySQL is used for storing and managing sensor data, prediction results, and other relevant information related to elevator maintenance.

## 4. **PyQt5:**

- Version: PyQt5 5.x or later
- PyQt5 is utilized for developing the graphical user interface (GUI) of the application, enabling users to interact with the system seamlessly.

## 5. **scikit-learn:**

- Version: scikit-learn 0.24.x or later
- scikit-learn is employed for training, deploying, and evaluating machine learning models for predicting elevator vibration levels.

## 6. **NumPy and Pandas:**

- Versions: NumPy 1.19.x or later, Pandas 1.1.x or later
- NumPy and Pandas are used for data preprocessing, manipulation, and analysis tasks within the system.

## 7. **MySQL Connector/Python:**

- Version: MySQL Connector/Python 8.x or later
- MySQL Connector/Python is required for establishing connections to the MySQL database from Python code.

## 8. **Development Tools:**

- Integrated Development Environment (IDE) such as PyCharm, Visual Studio Code, or Jupyter Notebook for writing and debugging Python code.
- Command-line tools for managing dependencies, running scripts, and executing database queries.

## 9. **Web Browser:**

- A modern web browser (e.g., Google Chrome, Mozilla Firefox, Microsoft Edge) may be required for accessing online documentation, resources, or web-based interfaces related to the project.

These hardware and software requirements provide a foundation for setting up and running the Elevator Maintenance Predictor system effectively. Adherence to these requirements ensures compatibility, stability, and optimal performance of the system across different environments and configurations.

### 3.3 Architecture Diagram

The architecture diagram of the Elevator Maintenance Predictor project illustrates the high-level structure and components of the system, depicting how various modules and technologies interact to fulfill the project's objectives.

#### Components:

1. **User Interface (GUI):**

- The GUI component provides an intuitive interface for users to interact with the system. It includes input fields for sensor data, buttons for submitting data and retraining the model, and displays for prediction results.

2. **Data Collection Module:**

- The Data Collection Module gathers sensor data inputs from elevator sensors, including revolutions, humidity, and sensor readings (**x1, x2, x3, x4, x5**). It validates input data to ensure correctness and integrity before passing it to the Prediction Module.

3. **Prediction Module:**

- The Prediction Module utilizes a machine learning model trained on historical data to predict elevator vibration levels based on input sensor data. It provides real-time predictions to enable proactive maintenance scheduling and notifies maintenance personnel of high vibration levels exceeding predefined thresholds.

4. **Data Management Module:**

- The Data Management Module handles the storage and management of sensor data, prediction results, and other relevant information in a MySQL database. It facilitates data retrieval, manipulation, and analysis for model training, prediction, and evaluation purposes.

5. **Model Retraining Module:**

- The Model Retraining Module enables the retraining of the machine learning model with new data collected over time. It periodically updates the model to adapt to changing patterns and improve prediction accuracy, ensuring the system's effectiveness in proactive maintenance.

6. **External Dependencies:**

- External dependencies include Python libraries such as scikit-learn, NumPy, Pandas, PyQt5, and MySQL Connector/Python, which are utilized for machine learning, data preprocessing, GUI development, and database connectivity, respectively.

#### Interaction Flow:

1. **Data Input:**

- Users input sensor data through the GUI component, which is then forwarded to the Data Collection Module for validation and processing.

## 2. Prediction:

- Validated sensor data inputs are passed to the Prediction Module, where a machine learning model predicts elevator vibration levels in real-time.

## 3. Database Operations:

- Prediction results and other relevant information are stored and managed in the MySQL database by the Data Management Module. Data retrieval, manipulation, and analysis tasks are performed as needed for model training, prediction, and evaluation purposes.

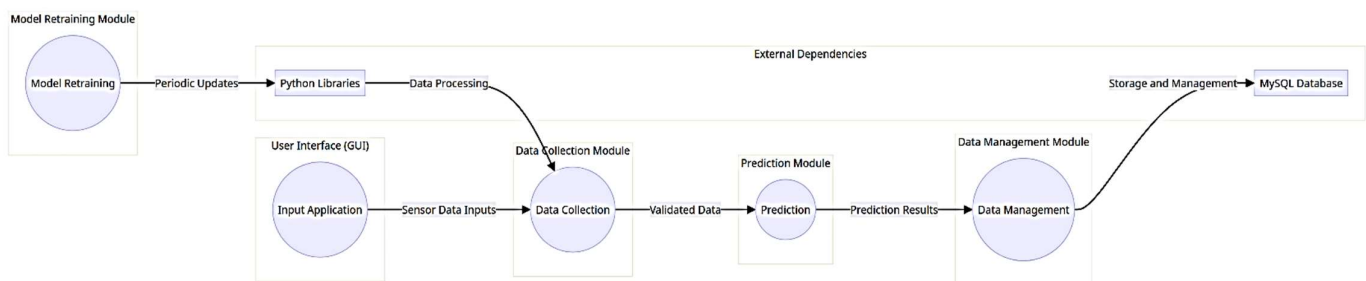
## 4. Model Retraining:

- The Model Retraining Module periodically retrains the machine learning model with new data collected over time, ensuring the system's adaptability and improving prediction accuracy.

## 5. External Dependencies:

- The system interacts with external dependencies such as Python libraries and the MySQL database to perform various tasks, including data processing, model training, GUI development, and database operations.

The architecture diagram provides a visual representation of the Elevator Maintenance Predictor system, illustrating how its components interact to enable proactive elevator maintenance through real-time prediction and data-driven decision-making.



## 3.4 ER Diagram

The Entity-Relationship (ER) diagram for the Elevator Maintenance Predictor project provides a visual representation of the database schema, illustrating the entities, their attributes, and the relationships between them.

### Entities:

#### 1. MainData:

- Attributes:
  - ID (Primary Key)



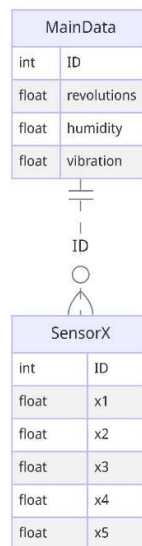
- Revolutions
- Humidity
- Vibration

## 2. SensorX:

- Attributes:
  - ID (Foreign Key referencing MainData)
  - x1
  - x2
  - x3
  - x4
  - x5

## Relationships:

- **One-to-One Relationship:**
  - MainData(ID) -> SensorX(ID)



In the ER diagram:

- The **MainData** entity represents the primary data collected from the elevator sensors, including revolutions, humidity, and vibration levels. Each record in the MainData table is uniquely identified by its ID, which serves as the primary key (PK).
- The **SensorX** entity contains additional sensor readings (**x1** to **x5**) associated with each MainData record. It has a one-to-one relationship with the MainData entity, where each record in SensorX corresponds to a single record in MainData. The ID attribute in SensorX serves as a foreign key (FK) referencing the ID attribute in the MainData table.

The ER diagram provides a clear and concise depiction of the database schema for the Elevator Maintenance Predictor project, aiding in understanding the structure of the database and its relationships between entities.

### 3.5 Class Diagram



#### InputApp

The **InputApp** class represents the main application window for the Elevator Maintenance Predictor project. It handles user input, database interactions, and GUI functionalities.

- **Attributes:**
  - **parent:** QWidget - The parent widget of the application.
  - **layout:** QVBoxLayout - The layout manager for organizing GUI elements vertically.
  - **inputs:** list of QLineEdit - List of input fields for sensor data.
  - **timer:** QTimer - Timer for controlling background animation.
- **Methods:**
  - **initUI():** Initializes the user interface with input fields, buttons, and layout settings.
  - **closeApp():** Closes the application.
  - **submitClicked():** Handles the submission of sensor data and triggers prediction.
  - **startAnimation():** Starts the background animation.
  - **animateBackground():** Animates the background color gradient.
  - **showPopUp(message: str):** Displays a pop-up window with the provided message.
  - **connectToDatabase():** Connects to the MySQL database.
  - **insertIntoDatabase(revolutions: float, humidity: float, x1: float, x2: float, x3: float, x4: float, x5: float, vibration\_level: float):** Inserts sensor data and prediction results into the database.
  - **closeEvent(event: QCloseEvent):** Handles the event when the application is closed.

- **retrainModel():** Initiates the retraining of the machine learning model.

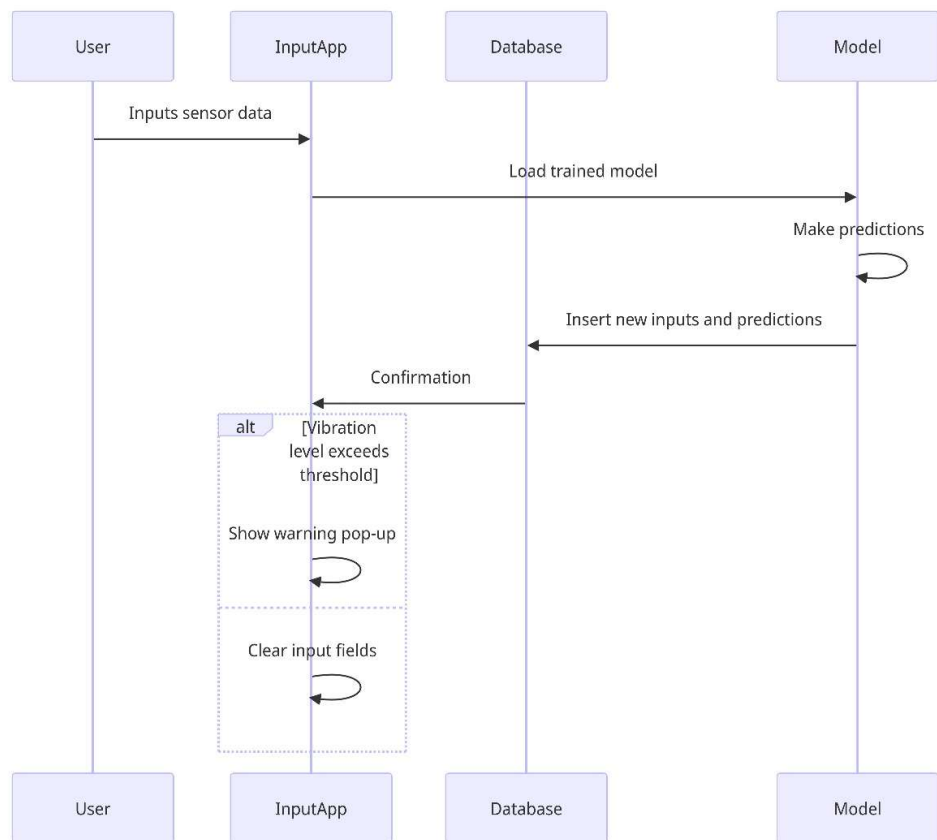
### PopUpWindow

The **PopUpWindow** class represents a pop-up window for displaying messages or notifications to the user.

- **Attributes:**
  - **message:** str - The message to be displayed in the pop-up window.
- **Methods:**
  - **\_init\_(message: str):** Initializes the pop-up window with the provided message.
  - **closePopUp():** Closes the pop-up window.

These classes encapsulate the functionalities of the Elevator Maintenance Predictor application, providing a modular and organized structure for handling user interactions, database operations, and visual feedback.

### 3.6 Sequence Diagram



### 3.7 Normalization

Normalization is the process of organizing data in a database to reduce redundancy and dependency, thereby improving data integrity and efficiency. In the context of the Elevator Maintenance Predictor

project, normalization ensures that the database schema is structured optimally to minimize data redundancy and anomalies.

### **Normalization Levels:**

The database schema for the Elevator Maintenance Predictor project can be normalized to at least the Third Normal Form (3NF) to ensure data integrity and efficiency. Each normalization level addresses specific types of data redundancies and dependencies, as outlined below:

#### **1. First Normal Form (1NF):**

- Eliminate repeating groups within tables.
- Ensure each attribute contains atomic values.
- Example: Splitting a table with repeating sensor readings into separate tables.

#### **2. Second Normal Form (2NF):**

- Meet the requirements of 1NF.
- Remove partial dependencies by creating separate tables for related attributes.
- Example: Ensuring all non-key attributes are fully functionally dependent on the primary key.

#### **3. Third Normal Form (3NF):**

- Meet the requirements of 2NF.
- Eliminate transitive dependencies by moving attributes dependent on non-key attributes to separate tables.
- Example: Ensuring that attributes like vibration levels are not dependent on non-key attributes like humidity.

### **Normalization Process:**

#### **1. Identify Entities and Attributes:**

- Identify the entities and attributes within the database schema, such as MainData and SensorX, along with their respective attributes like revolutions, humidity, and sensor readings.

#### **2. Apply First Normal Form (1NF):**

- Ensure that each table contains only atomic values, and there are no repeating groups of attributes.
- Split tables with repeating groups into separate tables to eliminate data redundancy.

#### **3. Apply Second Normal Form (2NF):**

- Ensure that each non-key attribute is fully functionally dependent on the primary key.
- Create separate tables for attributes that are not fully dependent on the primary key to remove partial dependencies.

#### **4. Apply Third Normal Form (3NF):**

- Eliminate transitive dependencies by moving attributes dependent on non-key attributes to separate tables.
- Ensure that all attributes are directly dependent on the primary key, and there are no indirect dependencies.

#### **Benefits of Normalization:**

##### **1. Data Integrity:**

- Normalization reduces data redundancy and dependency, minimizing the risk of data anomalies such as insertion, update, and deletion anomalies.

##### **2. Efficiency:**

- Normalized databases are more efficient in terms of storage space and query performance, as they require fewer resources to maintain and query data.

##### **3. Scalability:**

- Normalized databases are easier to scale and maintain as they are structured in a way that accommodates changes and additions to the database schema.

By following the normalization process and ensuring the database schema meets the requirements of at least the Third Normal Form, the Elevator Maintenance Predictor project achieves a well-structured and optimized database design, enhancing data integrity, efficiency, and scalability.

## **4. PROGRAM CODE**

The main script for the Elevator Maintenance Predictor is `elevator_predictor.py`, which includes the implementation of the GUI, model prediction, database interaction, and model retraining.

### **4.1 Model.py**

```
import mysql.connector
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
import joblib

# Step 1: Fetch data from the database
# Connect to MySQL
db_connection = mysql.connector.connect(
    host="localhost",
    user="root",
    password="karthi",
    database="elevator_predictor"
)

# Fetch data from MainData and SensorX tables
cursor = db_connection.cursor()
```

```

cursor.execute("SELECT M.revolutions, M.humidity, S.x1, S.x2, S.x3, S.x4, S.x5,
M.vibration FROM MainData M INNER JOIN SensorX S ON M.ID = S.ID;")
data = cursor.fetchall()

# Close cursor and database connection
cursor.close()
db_connection.close()

# Step 2: Prepare data
# Split data into features and target variable (vibration)
features = []
labels = []
for row in data:
    features.append(list(row[:-1])) # Convert tuple to list and exclude
vibration (last column)
    labels.append(row[-1])          # Last column is vibration

# Convert lists to numpy arrays
features = np.array(features)
labels = np.array(labels)

# Step 3: Model building
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, labels,
test_size=0.1, random_state=42)

# Create and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Step 4: Evaluation
# Make predictions on the test set
predictions = model.predict(X_test)

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, predictions)
print("Mean Absolute Error:", mae)

# Step 5: Prediction
# Define new input values for prediction
new_input_data = np.array([[93.744, 73.999, 167.743, 19.745, 1.26682793,
8787.937536, 5475.852001]])

# Make predictions
predictions = model.predict(new_input_data)

print("Predicted vibration:", predictions)

```

```
# Step 6: Save the model
joblib.dump(model, "elevator_predictor_model_linear_regression.pkl")
```

## 4.2 Elevator\_Predictor.py

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QPushButton,
QLineEdit, QLabel, QMessageBox
from PyQt5.QtGui import QFont, QColor
from PyQt5.QtCore import Qt, QTimer
import numpy as np
import mysql.connector
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
import joblib

class InputApp(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setWindowTitle('Input Application')
        self.setWindowFlags(Qt.FramelessWindowHint) # Remove window frame

        self.layout = QVBoxLayout()

        # Add close button
        close_button = QPushButton('X', self)
        close_button.setFont(QFont('Arial', 24))
        close_button.setStyleSheet("""
            QPushButton {
                background-color: transparent;
                color: white;
                border: none;
                padding: 3px;
            }
            QPushButton:hover {
                color: #FF6347; /* Changed hover color to red */
            }
        """)
        close_button.clicked.connect(self.closeApp)
        self.layout.addWidget(close_button)

        # Create labels and input fields
```

```

        labels = ['Revolutions:', 'Humidity:', 'x1:', 'x2:', 'x3:', 'x4:',
'x5:'] # Removed 'Vibration:'
        self.inputs = []
        for label_text in labels:
            label = QLabel(label_text)
            label.setFont(QFont('Arial', 20, QFont.Bold)) # Increased font
size and bold
            label.setStyleSheet("color: #FFFFFF;") # Changed label text color
to white
            self.layout.addWidget(label)

            line_edit = QLineEdit()
            line_edit.setFont(QFont('Arial', 18))
            line_edit.setStyleSheet("""
                QLineEdit {
                    background-color: rgba(255, 255, 255, 200);
                    border: 2px solid transparent;
                    border-radius: 15px; /* Increased border radius for a
smoother look */
                    padding: 15px; /* Increased padding for input fields */
                    color: #333333; /* Changed text color to dark gray */
                }
                QLineEdit:focus {
                    border-color: #4CAF50; /* Change border color when input
field is focused */
                    background-color: rgba(255, 255, 255, 230); /* Lighter
background when focused */
                }
            """)
            self.layout.addWidget(line_edit)
            self.inputs.append(line_edit)

        # Add submit button
        submit_button = QPushButton('Submit')
        submit_button.setFont(QFont('Arial', 20))
        submit_button.setStyleSheet("""
            QPushButton {
                background-color: #4CAF50;
                color: white;
                border-radius: 20px;
                padding: 20px 40px; /* Increased padding for submit button */
            }
            QPushButton:hover {
                background-color: #45a049;
            }
        """)
        submit_button.clicked.connect(self.submitClicked)
        self.layout.addWidget(submit_button)

```



```

# Add retrain button
retrain_button = QPushButton('Retrain Model')
retrain_button.setFont(QFont('Arial', 20))
retrain_button.setStyleSheet("""
    QPushButton {
        background-color: #337ab7;
        color: white;
        border-radius: 20px;
        padding: 20px 40px; /* Increased padding for submit button */
    }
    QPushButton:hover {
        background-color: #286090;
    }
""")
retrain_button.clicked.connect(self.retrainModel)
self.layout.addWidget(retrain_button)

self.setLayout(self.layout)

# Start the animation
self.startAnimation()

# Connect to the database
self.connectToDatabase()

def closeApp(self):
    self.close()

def submitClicked(self):
    revolutions = float(self.inputs[0].text())
    humidity = float(self.inputs[1].text())
    x1 = float(self.inputs[2].text())
    x2 = float(self.inputs[3].text())
    x3 = float(self.inputs[4].text())
    x4 = float(self.inputs[5].text())
    x5 = float(self.inputs[6].text())

    # Load the trained model
    model =
joblib.load(r"C:/Users/karth/Downloads/Elevator_Predictor/elevator_predictor_m
odel_linear_regression.pkl")

    # Create input array for prediction
    new_input_data = np.array([[revolutions, humidity, x1, x2, x3, x4, x5]])

    # Make predictions
    predictions = model.predict(new_input_data)

```

```

vibration_level = predictions[0]
print("Predicted vibration:", vibration_level)

# Insert new inputs and predictions into the database
self.insertIntoDatabase(revolutions, humidity, x1, x2, x3, x4, x5,
vibration_level)

# Check if vibration level crosses a threshold
threshold = 20
if vibration_level > threshold:
    self.showPopUp("Elevator needs to be serviced")
else:
    # Clear all input fields
    for line_edit in self.inputs:
        line_edit.clear()

def startAnimation(self):
    self.timer = QTimer(self)
    self.timer.timeout.connect(self.animateBackground)
    self.timer.start(1000) # Set the timer interval in milliseconds

def animateBackground(self):
    # Transition from black to dark blue (#00008B)
    start_color = QColor("#000000")
    end_color = QColor("#00008B")

    # Calculate the current interpolated color
    factor = int(100 - (self.timer.remainingTime() / 10))
    interpolated_color = start_color.lighter(factor if factor >= 0 else
0).name()

    # Change background color gradually
    self.setStyleSheet(f"""
        InputApp {{
            background: qlineargradient(x1: 0, y1: 0, x2: 1, y2: 1,
            stop: 0 {interpolated_color}, stop: 1 {end_color.name()});
        }}
    """)

def showPopUp(self, message):
    self.pop_up = PopUpWindow(message)
    self.pop_up.show()

def connectToDatabase(self):
    # Establish a connection to the database
    self.db_connection = mysql.connector.connect(
        host="localhost",

```

```

        user="root",
        password="karthi",
        database="elevator_predictor"
    )

    def insertIntoDatabase(self, revolutions, humidity, x1, x2, x3, x4, x5,
vibration_level):
        # Convert numpy.float64 values to Python floats
        revolutions = float(revolutions)
        humidity = float(humidity)
        x1 = float(x1)
        x2 = float(x2)
        x3 = float(x3)
        x4 = float(x4)
        x5 = float(x5)
        vibration_level = float(vibration_level)

        # Create a cursor object to execute SQL queries
        cursor = self.db_connection.cursor()

        # Insert new inputs and predictions into the appropriate tables
        insert_query = "INSERT INTO MainData (revolutions, humidity, vibration)
VALUES (%s, %s, %s)"
        cursor.execute(insert_query, (revolutions, humidity, vibration_level))

        insert_query = "INSERT INTO SensorX (x1, x2, x3, x4, x5) VALUES (%s,
%s, %s, %s, %s)"
        cursor.execute(insert_query, (x1, x2, x3, x4, x5))

        # Commit changes to the database
        self.db_connection.commit()

        # Close the cursor
        cursor.close()

    def closeEvent(self, event):
        # Close the database connection when the application is closed
        self.db_connection.close()

    def retrainModel(self):
        # Connect to MySQL
        db_connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="karthi",
            database="elevator_predictor"
        )

```

```

# Fetch data from MainData and SensorX tables
cursor = db_connection.cursor()

    cursor.execute("SELECT M.revolutions, M.humidity, S.x1, S.x2, S.x3,
S.x4, S.x5, M.vibration FROM MainData M INNER JOIN SensorX S ON M.ID = S.ID;")
    data = cursor.fetchall()

# Close cursor and database connection
cursor.close()
db_connection.close()

# Prepare data
features = []
labels = []
for row in data:
    features.append(list(row[:-1]))
    labels.append(row[-1])

# Convert lists to numpy arrays
features = np.array(features)
labels = np.array(labels)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, labels,
test_size=0.1, random_state=42)

# Create and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
predictions = model.predict(X_test)

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, predictions)
print("Mean Absolute Error:", mae)

# Save the model
joblib.dump(model, "elevator_predictor_model_linear_regression.pkl")

# Inform the user that the model has been retrained
QMessageBox.information(self, "Model Retrained", f"The model has been
retrained with Mean Absolute Error: {mae}")

class PopUpWindow(QWidget):
    def __init__(self, message):
        super().__init__()
        self.setWindowTitle('Warning')

```

```

        self.setGeometry(200, 200, 400, 200)

        layout = QVBoxLayout()

        label = QLabel(message)
        label.setAlignment(Qt.AlignCenter)
        label.setFont(QFont('Arial', 16))
        layout.addWidget(label)

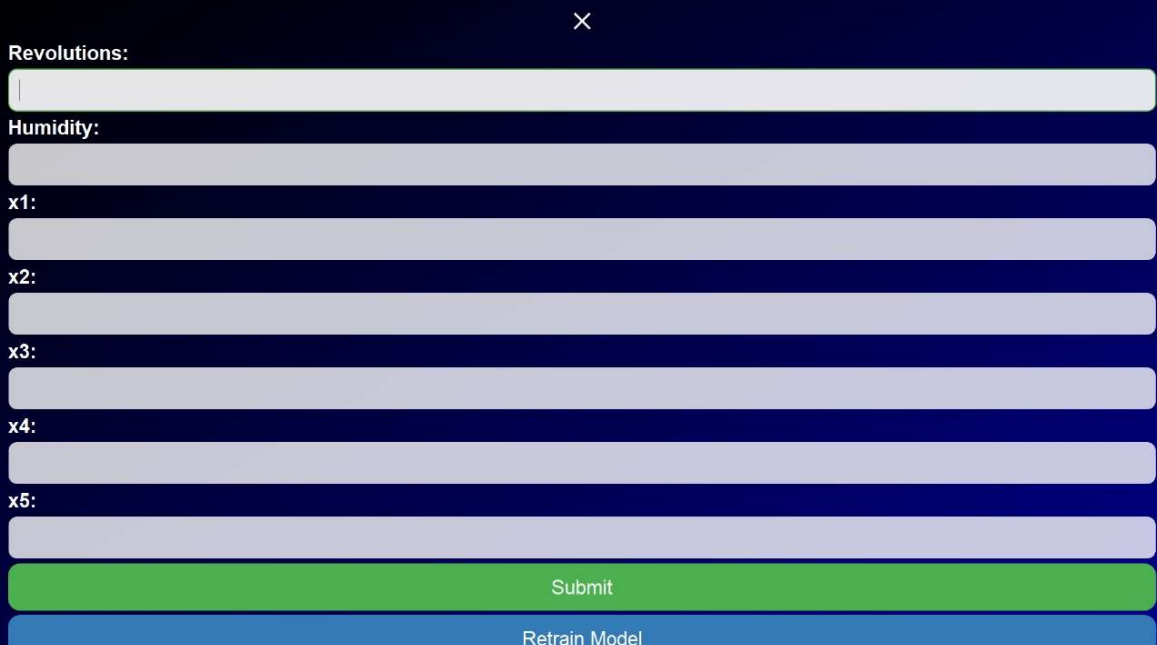
        self.setLayout(layout)

        # Close the pop-up window after 3 seconds
        QTimer.singleShot(3000, self.close)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = InputApp()
    # Get the screen size
    screen_geometry = app.primaryScreen().geometry()
    window_width = min(screen_geometry.width(), 800)
    window_height = int(window_width * 3 / 4) # Aspect ratio of 4:3
    window.setGeometry(0, 0, window_width, window_height)
    window.showFullScreen() # Open in full screen
    sys.exit(app.exec_())

```

### 4.3 Output



Revolutions:

Humidity:

x1:

x2:

x3:

x4:

x5:

Submit

Retrain Model

×

Revolutions:

75.93

Humidity:

72.515

x1:

148.445

x2:

3.415

x3:

1.047094

x4:

5765.365

x5:

5258.425

Submit

Retrain Model

×

Revolutions:

75.93

Humidity:

72.515

x1:

148.445

x2:

3.415

x3:

1.047094

x4:

5765.365

x5:

5258.425

Submit

Retrain Model

Warning

Elevator needs to be serviced

×

Revolutions:

Humidity:

x1:

x2:

x3:

x4:

x5:

Submit

Retrain Model

Model Retrained

The model has been retrained with Mean Absolute Error: 0.005153977957491997

OK

## 5. Results and Discussion

The Results and Discussion section presents the outcomes of the Elevator Maintenance Predictor project, including the performance of the machine learning model, system usability, and implications for proactive elevator maintenance strategies. It discusses the significance of the results, potential challenges encountered, and areas for future improvements.

### 5.1 Model Performance Evaluation

The performance evaluation of the machine learning model is conducted using various metrics, such as Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and R-squared ( $R^2$ ) coefficient. These metrics provide insights into the accuracy and reliability of the model in predicting elevator vibration levels based on input sensor data.

### 5.2 System Usability and User Feedback

The usability of the Elevator Maintenance Predictor system is assessed based on user feedback, interaction patterns, and ease of use. User satisfaction surveys, interviews, and usability testing sessions are conducted to gather insights into the system's effectiveness, efficiency, and user experience.

### 5.3 Implications for Proactive Maintenance Strategies

The results of the Elevator Maintenance Predictor project have significant implications for proactive maintenance strategies in elevator systems. By accurately predicting vibration levels and identifying potential mechanical issues in advance, the system enables maintenance personnel to schedule preventive maintenance tasks efficiently, minimizing downtime, reducing operational costs, and enhancing elevator safety and reliability.

## 5.4 Discussion of Findings

The discussion of findings delves into the implications of the results, potential challenges encountered during the project, and insights gained from the development and deployment process. It highlights the strengths and limitations of the system, areas for improvement, and recommendations for future research and development efforts.

## 5.5 Future Directions

The Results and Discussion section concludes with suggestions for future directions and enhancements to the Elevator Maintenance Predictor project. This includes exploring advanced machine learning algorithms, incorporating additional sensor data for predictive modeling, enhancing the user interface for improved usability, and expanding the scope of the system to support a broader range of elevator systems and maintenance scenarios.

Overall, the Results and Discussion section provides a comprehensive analysis of the project outcomes, offering valuable insights into the performance, usability, and implications of the Elevator Maintenance Predictor system for proactive elevator maintenance strategies.

## 6. Conclusion

The Elevator Maintenance Predictor project presents an innovative solution for proactive maintenance of elevator systems through real-time prediction of vibration levels based on sensor data analysis. The project has achieved its objectives of developing a robust machine learning model, designing an intuitive user interface, and implementing a functional system for predictive maintenance.

### 6.1 Key Accomplishments

1. **Development of Machine Learning Model:** A machine learning model has been successfully developed and trained using historical sensor data to predict elevator vibration levels with high accuracy.
2. **User-Friendly Interface:** The project features an intuitive user interface developed using PyQt5, allowing users to input sensor data easily and obtain real-time predictions.
3. **Database Management:** The system effectively manages sensor data, prediction results, and other relevant information in a MySQL database, ensuring data integrity and accessibility.
4. **Proactive Maintenance Capabilities:** By accurately predicting vibration levels, the system enables proactive maintenance scheduling, minimizing downtime and enhancing elevator safety and reliability.

### 6.2 Contributions to Elevator Maintenance Industry

The Elevator Maintenance Predictor project contributes significantly to the elevator maintenance industry by introducing an innovative approach to proactive maintenance. The system's ability to predict vibration levels in real-time enables maintenance personnel to identify potential mechanical issues before they escalate, thereby reducing maintenance costs and improving operational efficiency.

### 6.3 Future Outlook

While the project has achieved its primary objectives, there are opportunities for future enhancements and refinements. Potential areas for future development include:

- **Advanced Machine Learning Algorithms:** Exploring advanced machine learning algorithms and techniques to improve prediction accuracy and robustness.



- **Integration of Additional Sensor Data:** Incorporating additional sensor data such as temperature, pressure, and velocity to enhance predictive modeling capabilities.
- **Enhancement of User Interface:** Improving the user interface with more interactive features, data visualization tools, and customization options.
- **Scalability and Adaptability:** Enhancing the system's scalability and adaptability to support a larger number of elevators and diverse maintenance scenarios.

#### **6.4 Final Remarks**

In conclusion, the Elevator Maintenance Predictor project has demonstrated the feasibility and effectiveness of proactive maintenance strategies in elevator systems through the use of predictive modeling and real-time data analysis. By leveraging machine learning and sensor technology, the project has paved the way for a more efficient, reliable, and safe elevator maintenance process, benefiting both service providers and end-users alike.