DATA-DRIVEN PREDICTIVE MAINTENANCE

Rahma Dawud
Dagmawi Sahleslassie Gessesse

Course:
Data Mining, Spring 2023
Prof. Dr. Adalbert F. X. Wilhelm
May 30, 2023

ABSTRACT

In today's industrial landscape, detecting and addressing potential machine failures before they occur is essential for minimizing equipment downtime and improving overall productivity. This work presents a machine learning approach for predicting machine failures due to a component within the next 48-hour time window, aiming to enable proactive maintenance in industrial settings. Leveraging the publicly available Microsoft Azure Predictive Maintenance dataset, valuable insights and patterns were uncovered through exploratory data analysis. Through feature engineering, informative features were constructed. Since machine failures are extreme rare events, the dataset is heavily imbalanced. To deal with that, higher weights were assigned to the minority failure cases during training. Subsequently, after careful data splitting, both multi-layer perceptron (MLP) and LightGBM models were employed to tackle the multi-class failure classification task. Furthermore, hyperparameter optimization was carried out specifically for the MLP to enhance its performance. While the MLP achieved a very good result, the LightGBM model outperformed it significantly, even without fine-tuning. In addition to the superior performance, the simplicity and its inherent explainability of LightGBM make it a compelling choice for the final model. In conclusion, this work showed the value of leveraging advanced machine learning techniques to help companies improve their operational efficiency.

1 Table of Contents

2 Introduction

In the era of Industry 4.0, the implementation of a robust maintenance strategy is essential for companies to ensure smooth operation and gain a competitive advantage [1]. The share of equipment maintenance is estimated to range from 15 to 60 % of the total operational cost [2] [3]. Three main strategies are commonly employed: reactive, periodic (scheduled), and predictive maintenance. Reactive maintenance is a simplistic approach where interventions occur after failures, resulting in significant downtime and high intervention costs. Periodic maintenance follows a predetermined schedule, aiming to prevent failures but often leading to unnecessary corrective actions and inefficient resource utilization. In contrast, predictive maintenance, driven by machine learning and data analysis, enables proactive interventions before failures occur, increasing productivity and allowing for optimal planning of service interventions and spare part handling.

In this work, our focus is on predictive maintenance, which has emerged as one of the most effective applications of machine learning [4]. By harnessing the power of artificial intelligence and data-driven insights, predictive maintenance moves beyond reactive and preventive strategies to accurately forecast equipment failures and optimize maintenance schedules.

To explore the potential of predictive maintenance, we will utilize the Microsoft dataset [5], a widely used benchmark dataset in the field. This dataset contains diverse sensor readings, maintenance logs, and failure events recorded over a significant period. By analyzing this dataset, we aim to develop a predictive maintenance model that effectively predicts failures, enables timely interventions, and optimizes maintenance schedules to enhance productivity and cost-effectiveness.

2.1.1 Project goal

By analyzing this dataset, we aim to develop a machine learning model for predicting machine failures due to a specific component within a pre-selected time frame. The selection of the time frame depends on the specific business requirement. In some cases, a shorter time window might be sufficient for timely intervention, while in other scenarios, a longer window spanning days or weeks might be necessary.

Here, we have chosen a time window of 48 hours. This decision is based on the understanding that it allows for an appropriate duration to assess the probability of machine failure and take necessary preventive measures. By forecasting machine failures within this 48-hour window, maintenance teams can proactively plan for repairs, coordinate resources, and minimize operational disruptions. Therefore, the objective is to build a predictive maintenance model for predicting the probability of machine failure due to a certain component in the next 48 hours.

2.1.2 Related Works

As the dataset is publicly available on Kaggle, numerous analysts have analyzed the data and examined different aspects. It ranges from data exploration to experimenting with different modeling techniques, providing valuable inspiration for our work. See for example [6], [7] , and [8]. Additionally, Microsoft has a dedicated GitHub repository [9] focusing on predictive maintenance and featuring sample notebooks and another dataset, demonstrating the development of predictive maintenance models.

2.1.3 Outline

In Section 3, we provide a comprehensive overview of the dataset. Then, in Section 4, we conduct an exploratory data analysis. Subsequently, in Section 5, we undertake the necessary steps for preparing the training data, including feature engineering and the creation of the appropriate target variable. Section 6 presents the evaluation results of MLP and LightGBM models. Finally, the report concludes with summary and key findings.

3 Dataset

We used the Microsoft Azure Predictive Maintenance dataset [9] which provides a rich and realistic data allowing researchers and practitioners to explore the opportunities and challenges of predictive maintenance. This dataset contains a collection of data from 100 machines for the year 2015 offering an hourly time series.

Our analysis is based on the Microsoft Azure Predictive Maintenance dataset [1], which offers a comprehensive and realistic collection of data, enabling researchers and practitioners to investigate the potential and complexities of predictive maintenance. This dataset comprises information from 100 machines throughout the year 2015, providing an extensive hourly time series. The dataset comprises of five data sources.

1. **Telemetry Data** (PdM_telemetry.csv): For the year 2015, the dataset provides hourly average measurements of voltage, rotation, pressure, and vibration collected from 100 machines.

2. **Error** (PdM_errors.csv): These errors occur while the machines are in operation but do not result in machine shutdown, and thus, they are not categorized as failures. The error date and times are rounded to the nearest hour to align with the telemetry data collection frequency

3. **Maintenance** (PdM_maint.csv): If a machine component is replaced, it is recorded in this table. Component replacements occur in two situations: 1) During regular scheduled visits, technicians perform proactive maintenance by replacing the

component. 2) In case of a component breakdown, technicians conduct unscheduled maintenance to replace the faulty component, which is considered a failure and captured in the Failures data. The maintenance data includes records from both 2014 and 2015 and is rounded to the nearest hour to align with the hourly telemetry data collection rate.

4. **Failures** (PdM_failures.csv): Each entry in this dataset signifies the replacement of a failed component.

5. **Machines** (PdM_Machines.csv): Contains the model type and age of the Machines.

4 Exploratory Data Analysis

To extract valuable insights from the dataset, our initial step involves conducting an exploratory data analysis (EDA). By thoroughly examining each data source, we will gain a comprehensive understanding of the variables' distributions, patterns, trends, and potential interrelationships. Utilizing a combination of visualizations, statistical analyses, and exploratory techniques, we aim to uncover hidden information and key features that will guide subsequent data modeling and analysis.

To simplify the presentation of the data, it is not feasible to include plots of every machine and variable in this report. Therefore, we have focused on showcasing a few representative examples that provide meaningful insights. However, for a more comprehensive exploratory data analysis (EDA), we recommend referring to the accompanying Jupyter notebook.

4.1 Telemetry data

The primary data source is the telemetry time-series data, comprising real-time measurements of voltage, rotation, pressure, and vibration.

The data contains 876100 records and contains no missing values and duplicates. This indicates that it is a synthetic dataset. However, we can see that the data is not entirely clean. There are a few spikes in the data that are likely due to sensor malfunction. In Figure 1 , the telemetry data is shown for 5 days for two machines. It shows that there are variations in the operating conditions.

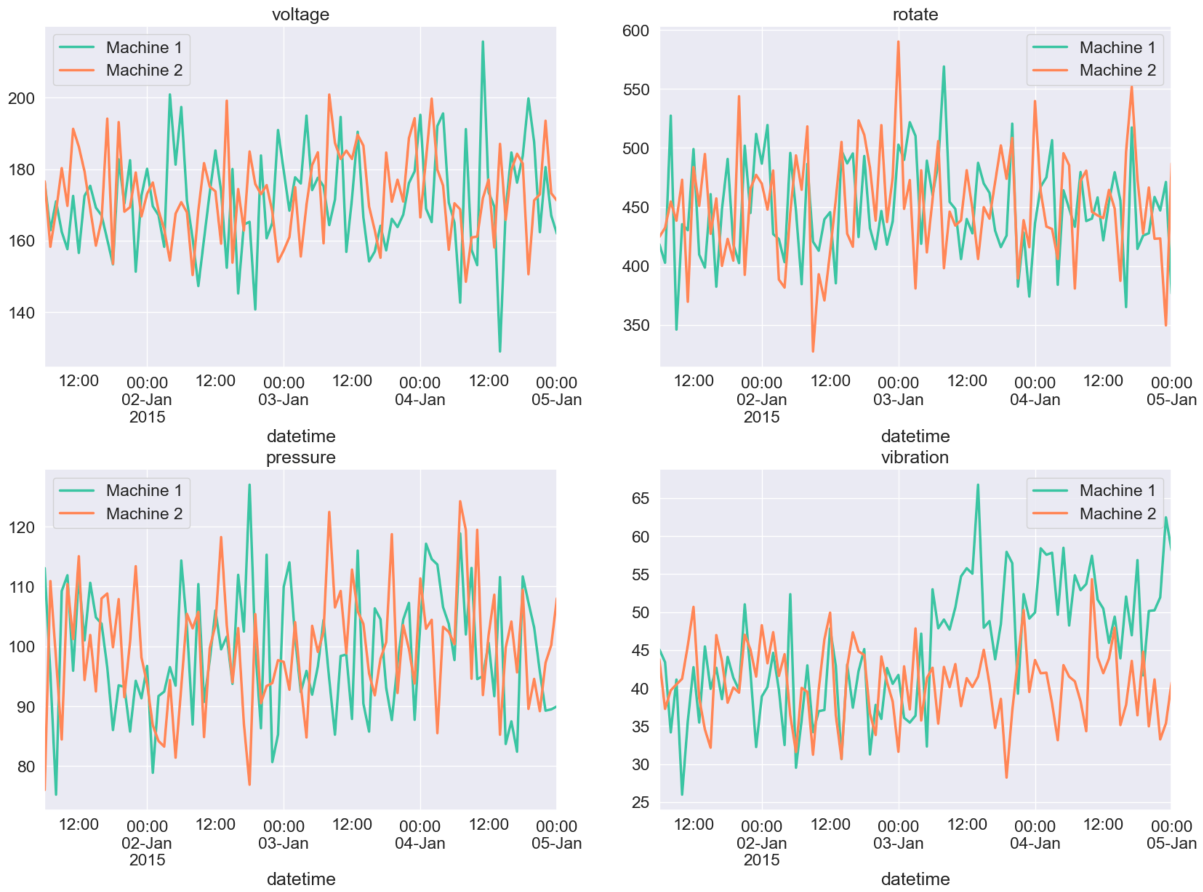

Figure 1 Telemetry Measurements for Two Machines

Next, we have had a look at the distribution of the telemetry sensor measurements. In Figure 2, the histogram and box plots are shown for machineID 1. All of them seem to have a similar distribution with different mean values. The box plots show that there are also some outliers. They might be anomalies, indicating failures.

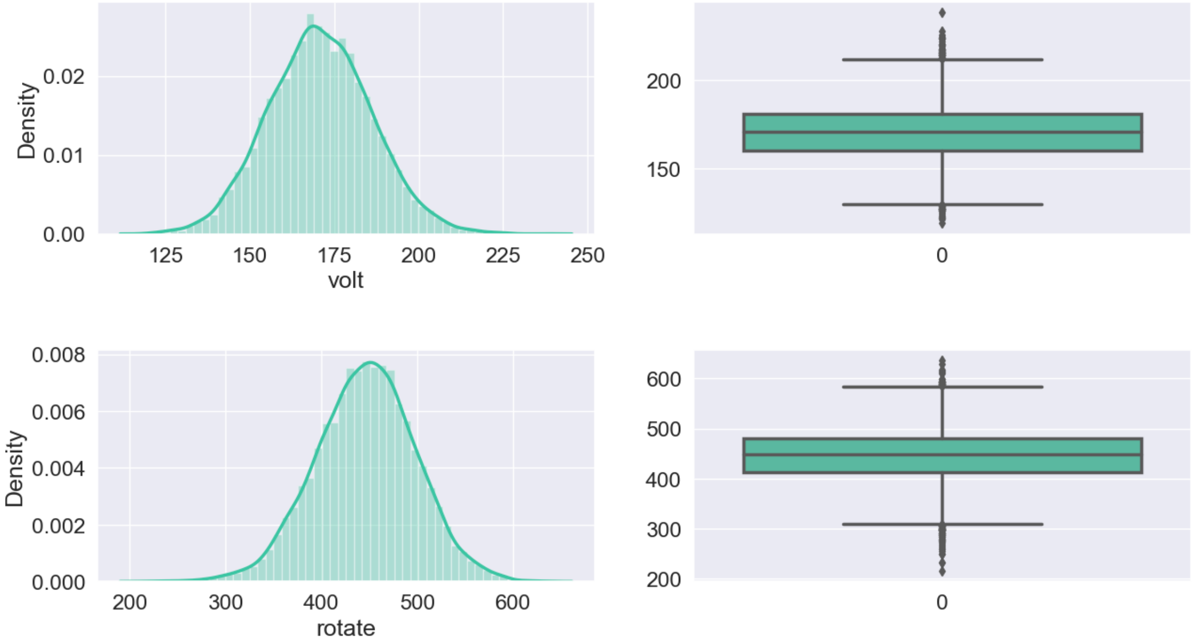

Figure 2 Distribution of voltage and rotation

We also performed a normality tests using the normaltest function from the scipy.stats module. This test is based on D'Agostino and Pearson's omnibus normality test [5], which combines skewness and kurtosis to assess the departure from normality. The test statistic and p-value were calculated for each telemetry measurement. The null hypothesis of the test is that the data follows a normal distribution. If the p-value is below the chosen significance level (typically 0.05), we reject the null hypothesis and conclude that the data does not follow a normal distribution. In this case, the test results, as shown in Table 1, indicate that the 'volt', 'rotate', 'pressure', and 'vibration' telemetry measurements for machineID 1 do not exhibit a normal distribution.

Table 1 Normality test for machine 1

	statistic	p-value
Voltage	12.119	2.00×10^{-3}
Pressure	238.383	1.72×10^{-52}
Vibration	256.513	1.98×10^{-56}
Rotation	35.796	1.68×10^{-08}

4.2 Errors

The error logs data, as previously mentioned, consists of non-breaking errors that occur during the operation of the machines. It is important to emphasize that these errors should not be considered as failures.

The dataset contains a total of 3,919 records, each corresponding to a specific timestamp and one of the five error types for each machine. Figure 3 illustrates the frequencies of each error type in the dataset. Error types 1 and 2 are the most frequent.

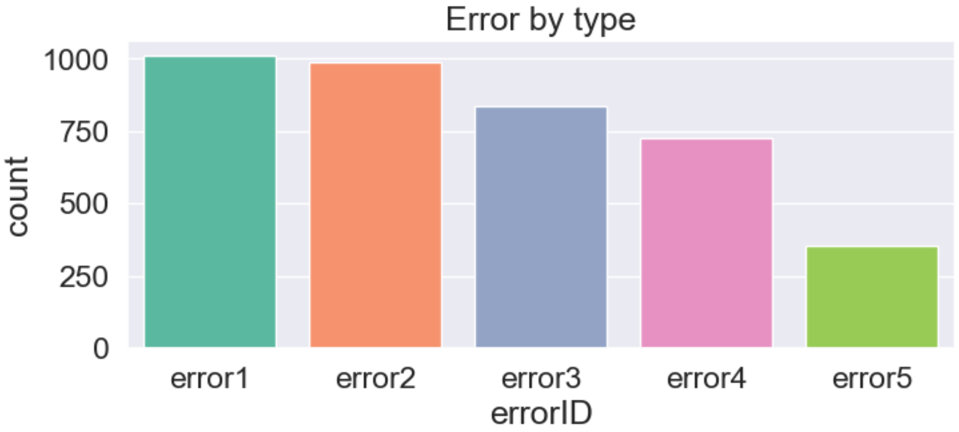

Figure 3 Distribution of error types

4.3 Maintenance

The dataset includes records of component replacements, which are captured in the maintenance table. Components are replaced in two scenarios:

- Proactive Maintenance: During the regular scheduled visits, technicians proactively replace components as part of preventive maintenance measures. These replacements are logged in the dataset, indicating planned maintenance actions.

- Reactive Maintenance: In cases where a component experiences a breakdown, technicians perform unscheduled maintenance to replace the faulty component. These instances are considered failures, and the corresponding data is recorded under the failures table.

The maintenance data includes 3,286 records from both 2014 and 2015.

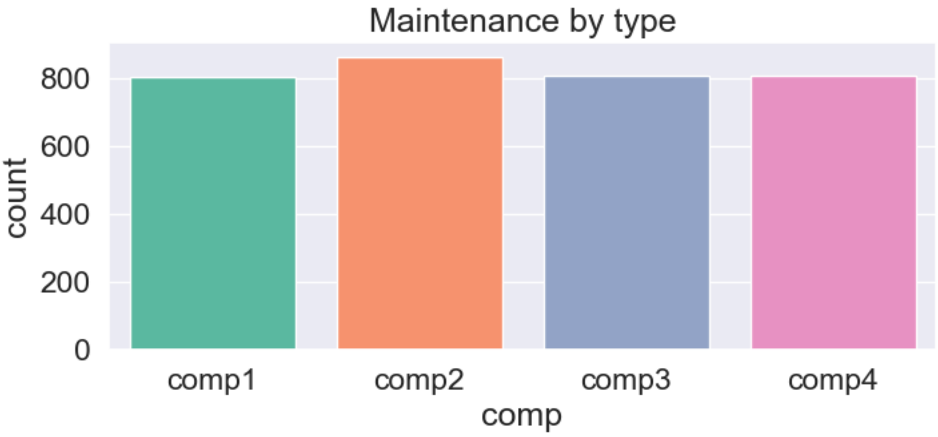

Figure 4 Distribution of component replacements due to scheduled maintenance

The number of records for each component is generally similar (**Error! Reference source not found.**), except for component 2, which has a slightly higher number of records. This observation suggests that during scheduled maintenance, it is possible that each component undergoes replacement procedures.

4.4 Machines

In this dataset, the model type and age (years in service) for each of the 100 machines.

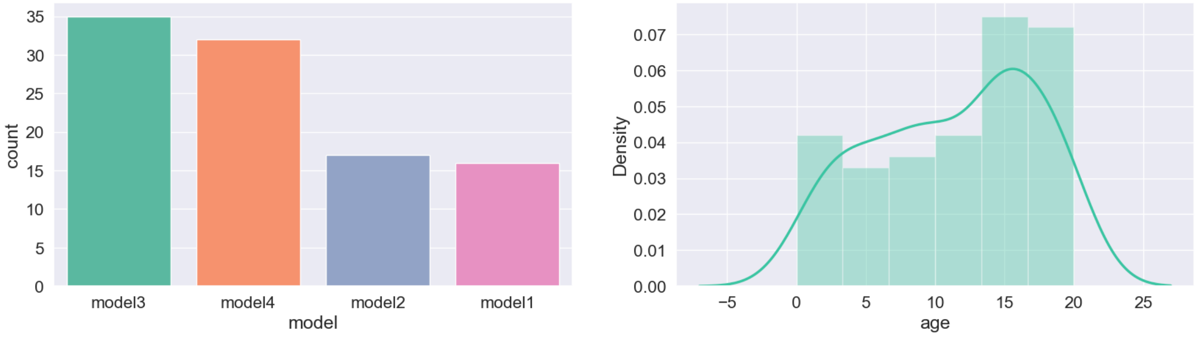

Figure 5 Model and age of the machines

As as visualized in Figure 5, model 3 and 4 machines make up a significant portion, comprising more than 60 percent, of the total. The age of the machines spans from 0 to 20 years.

The age distribution by model is shown in Figure 6. Based on the median age, the model 4 machines are newer than the other models.

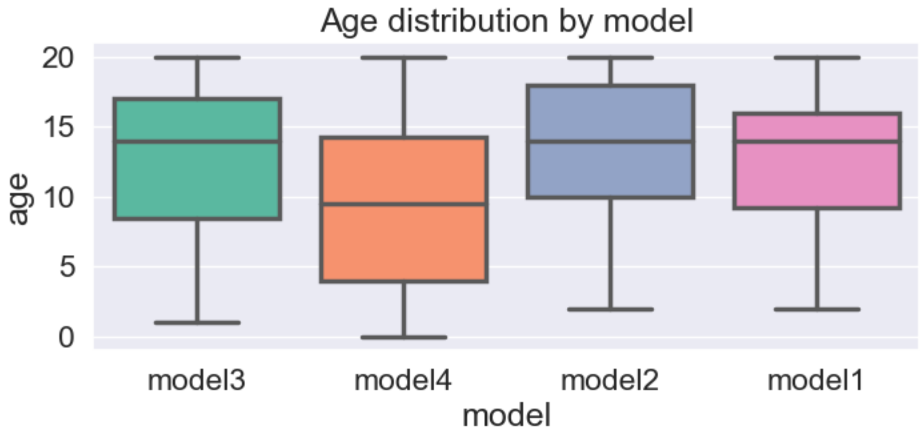

Figure 6 Distribution of the age of the machines per model type

4.5 Failures

This dataset contains component replacements due to failures. There are 761 failures recorded in 2015 and the frequencies of each component type is depicted in Figure 7.

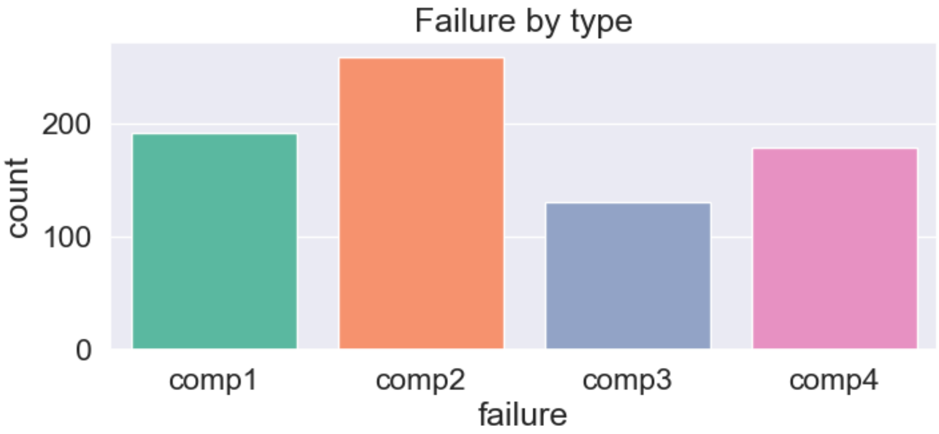

Figure 7 Distribution of component failures

Component 2 has the highest failure rate among all the components. This is consistent with the error frequencies of the components (**Error! Reference source not found.**). Component 3 has the fewest failures, indicating a relatively more reliable performance compared to the rest of the components.

Furthermore, the top 10 machines with the highest number of failures can be seen in Figure 8. Machine 99 has the most failures.

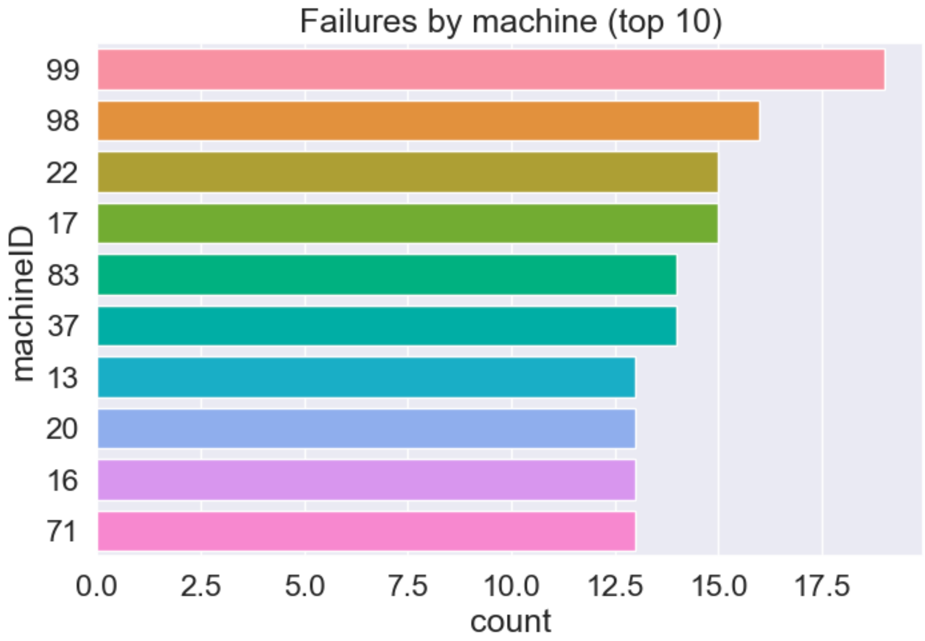

Figure 8 Top 10 machines with the most failures

5 Feature Engineering and Target Variable Construction

In this section, we present the feature engineering process and the preparation of the target variable for our intended use in modeling the failure of components within a 48-hour window.

5.1 Feature Engineering

In this section, we present the feature engineering process conducted to create meaningful features for predictive maintenance. The feature engineering aims to integrate the information from the four data sources: telemetry, error logs, component replacements, and machine features. By combining these sources, we can derive insightful features that capture the relevant patterns which provide signals for our task of component failure prediction. The features we engineered include lag features from telemetry and error data, number of days since last component replacement, and the machine features.

5.1.1 Telemetry Lag Features

Lag features are created from the telemetry data to capture the historical behavior of the sensor measurements. Specifically, we generated lag features with a short-term window of 4 hours and a long-term window of 48 hours. To derive the lag features, we used rolling mean and standard deviation as aggregations. These statistics help capture trends and fluctuations in the telemetry data, which are indicative of potential machine failures.

5.1.2 Lag Features from Error Data

The error logs contain categorical information about errors encountered during machine operation. To incorporate this data into our model, we generate lag features based on the occurrence of errors within the designated time windows. Specifically, we sum the number of errors for each error type within the given window, providing insights into the error patterns and their potential impact on machine performance. Since errors are relatively rare, we used only a 48-hour windows.

5.1.3 Number of Days Since Last Component Replacement

To capture the maintenance history of the machines, we calculate the number of days since the last component replacement. This feature captures the duration since the last replacement of a component and is expected to correlate strongly with component failures. As components are used for a longer period, the likelihood of degradation increases. This feature serves as a valuable indicator of the maintenance history of the machine.

5.1.4 Machine Features

The machine-specific features, model type and age, were used directly. These features capture the inherent differences between machines and can contribute to understanding the varying performance and failure patterns across different machine types. Since model type is a categorical feature, we need to encode it later.

5.2 Target Variable Preparation

The preparation of the target variable is a crucial step in training our predictive maintenance model to estimate the probability of machine failure attributed to a specific component. Our approach involves employing a multi-class classification technique to classify failures based on component-related issues. This involves defining a specific time window prior to a failure event and categorizing the records accordingly. If a failure event occurs within the predefined time window, the corresponding record is labeled as a "failure." On the other hand, if no failure event occurs within the specified time window, the record is labeled as "Normal." This approach enables us to distinguish between instances where a failure is anticipated and instances where the machine is expected to operate normally,

For our particular prediction problem, our focus is on estimating the probability of a machine experiencing failure within the next 48 hours as a result of a specific component malfunction. To facilitate this, we construct a categorical failure feature, serving as the label. Within a 48-hour window preceding a failure event, all corresponding records are assigned the "failure" value corresponding to the component responsible. For instance, if a failure event linked to component 1 occurs within this defined window, all records falling within that temporal scope are labeled as "failure=comp1." This labeling process is applied consistently for component 2, 3, and 4 failures. Conversely, all records situated outside this 48-hour window are labeled as "failure=Normal," denoting the absence of an imminent failure occurrence.

By meticulously preparing the target variable according to these guidelines, we establish a well-defined classification problem that empowers us to train a robust predictive model. This model aims to estimate the likelihood of machine failure pertaining to a specific component within the designated 48-hour timeframe.

6 Modeling and Evaluation

Once the training data has been prepared, the focus shifts towards modeling. Our specific task involves a multi-class classification problem. This section provides a detailed description of the modeling and evaluation techniques we used.

6.1 Class imbalance

In predictive maintenance applications, the occurrence of failure events is often relatively rare, resulting in imbalanced training data. In our dataset, only 4% of the records represent failures (Figure 9), which poses a challenge for training an effective model. To address this issue, several techniques can be employed to deal with class imbalance, including resampling the dataset, adjusting class weights, or using appropriate evaluation metrics.

We have decided to use the class weighting technique. Class weighting assigns higher weights to the minority failure classes during training, i.e., more importance is given to them. The loss function is adjusted based on class weights. That way, the model learns to give more importance to the correct classification failures. Class weights were automatically calculated based on the failure distribution, ensuring that the model assigns higher importance to failure classes. This approach aimed to improve the model's ability to learn from minority classes and achieve a more balanced prediction performance across all classes.

We also considered using Synthetic Minority Over-sampling Technique (SMOTE) [10]. By generating synthetic samples for the minority class, we can increase the representation of failure cases in the training data, enabling the model to learn from a more balanced dataset. However, since we are dealing with a timeseries data with lag features and uses k nearest neighbors, SMOTE will end up using future values. Hence, it is not suitable for our case.

In our application, a higher recall is crucial. Recall, defined as TP / (TP + FN), measures the ability of our model to correctly identify actual positive instances (failures). Given the potential significant costs associated with undetected failures, we aim to minimize false negatives (FN) and maximize true positives (TP). By focusing on recall, we prioritize identifying as many failures as possible, even if it results in a higher false positive rate. However, precision, which measures the proportion of correctly predicted positive instances (TP / (TP + FP), should not be overlooked. False positives (FP) can also lead to a loss of time and resources. Therefore, achieving an acceptable level of precision is necessary. Since recall and precision are not independent, we used recall as our primary evaluation metric.

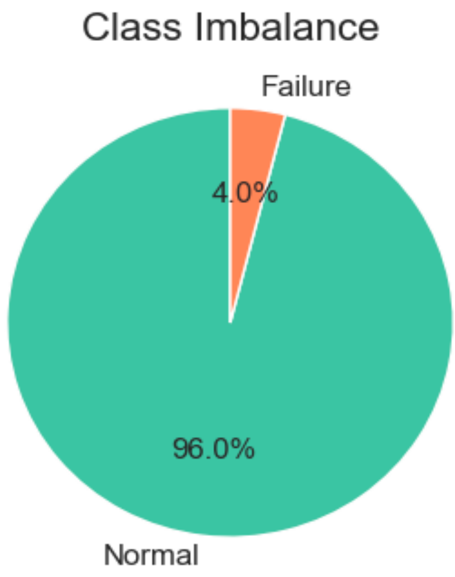

Figure 9 Class imbalance in failures

6.2 Train-validation-test split

In order to train, validate, and test our predictive model effectively, we employ a train-validation-test approach. Given that our data is time series, it is crucial to consider a time-based split to avoid data leakage and ensure reliable model evaluation.

To achieve this, we splited our dataset into three portions: the training, validation, and test sets. The training data is used to train our model, while the validation data allows us to fine-tune model parameters and assess its performance before applying it to unseen data. Finally, the test data serves as an independent evaluation set to measure the model's generalization capabilities.

It is important to note that when working with time series data, incorporating lag features can provide valuable information about past observations. However, to prevent data leakage, it is essential to introduce gaps between the training, validation, and test periods. Here, we introduce a 48-hour gap (consistent with our largest window size for the lag) between the end of the training period and the start of the validation period, as well as another 48-hour gap between the end of the validation period and the start of the test period. This ensures that our model does not utilize information from the training data during evaluation.

We employed a stratified 70/15/15 split to partition the data into training, validation, and test sets. Consequently, the training set encompassed the data up until 2015-09-15 05:00:00, while the validation set covered the period between 2015-09-17 06:00:00 and 2015-11-09 05:00:00. The actual sizes are shown in **Error! Reference source not found.**. This splitting approach ensured that we had a substantial amount of data for training our models, while still allowing for proper evaluation and validation of their performance.

Table 2 Dataset sizes for train, validation, and test

Failure	Training	Validation	Test
comp1	7,058	881	1,103
comp2	8,574	1,712	1,816
comp3	4,268	762	737
comp4	5,473	1,034	983
Normal	591,411	122,816	117,867

6.3 Data Preprocessing

To ensure fair treatment of features and facilitate convergence during training, we applied standardization to our dataset. By standardizing the data, we equalize the impact of different features, allowing them to contribute more effectively to the learning process. The features are normalized in such a way that each feature has zero mean and unit variance.

In addition, we addressed the categorical feature representing the machine model types. Since machine models are represented as discrete categories, we applied categorical encoding to transform them into a numerical representation suitable for our predictive model. Specifically, we applied one-hot encoding, which creates binary variables for each unique machine model category.

6.4 Modeling

In the modeling phase, we explored two different approaches to address our multi-class classification problem. The first approach involved implementing a Multi-Layer Perceptron (MLP) model using the Keras framework, while the second approach utilized a lightGBM model. In this section, we will present these approaches and discuss their respective outcomes. Here it should be noted that, the choice of the algorithms is partly made with our wish in learning more about how to train and fine tune neural networks.

6.4.1 Multi-Layer Perceptron

For the MLP model, we began by encoding the target variable using one-hot encoding. Subsequently, we constructed a neural network with two hidden layers, each comprising 300 units and employing the rectified linear unit (ReLU) activation function. A dropout of 0.2 is applied after each hidden layer for regularization. The output layer consisted of five units, utilizing the softmax activation function. Given the nature of our problem as a multi-class classification, we employed the weighted categorical cross-entropy as the loss function. We chose Adam optimizer with its default learning rate of 0.001.

The categorical cross-entropy loss measures the dissimilarity between the predicted class probabilities and the true class labels. It is widely used in multi-class classification problems as it effectively penalizes deviations between predicted probabilities and ground truth labels. The categorical cross-entropy loss is computed as follows:

$$\mathrm{L}(\hat{y}, y) = \sum_{i=0}^{k} \mathrm{w_i} \, . \, \mathrm{y_i} . \log \hat{y}_i$$

where $\hat{y}$ represents the predicted class probabilities, y represents the true class labels, and w are the class weights.

All choices of the hyperparameters are somehow arbitrary for now. However, we tried to take inspiration from the guide provided by Harvard and Google Brain scientists on practical tips on tuning deep learning models [11] and experimented manually.

During the training process, the MLP model underwent 20 epochs with a batch size of 1024, where each epoch represents a single pass through the training data. The progression of the loss and accuracy on the training data is illustrated in Figure 1. Notably, the training loss stabilized after approximately 15 epochs, while the accuracy on the balanced training set reached an impressive 0.95. To evaluate the model's performance, we examined the confusion matrices for the validation and test sets, which are presented in Table 3.

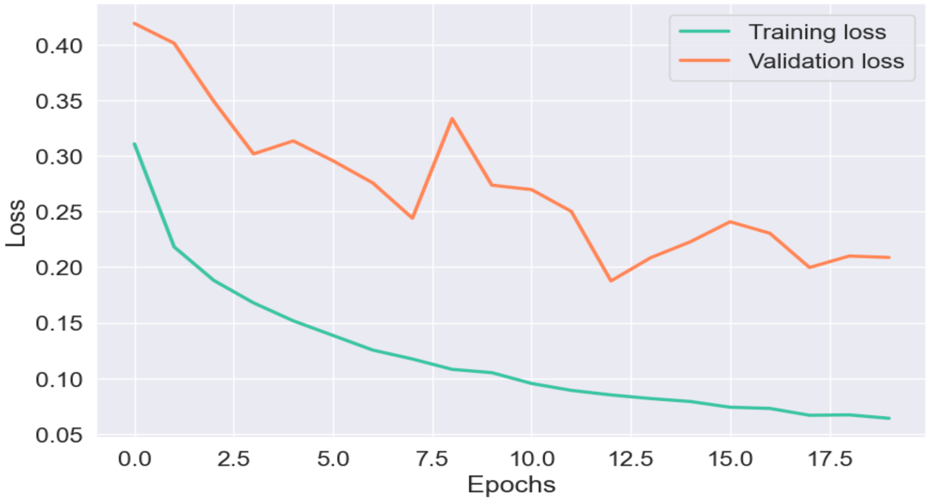

Figure 10 Loss curves for the MLP model

Table 3 Confusion matrix for the validation and test sets

Validation

	Normal	comp1	comp2	comp3	comp4
Normal	115268	2788	2689	951	1120
comp1	198	671	11	0	1
comp2	286	12	1386	25	3
comp3	60	2	6	690	4
comp4	150	7	7	4	866

Test

	Normal	comp1	comp2	comp3	comp4
Normal	110457	2759	2696	846	1109
comp1	174	864	38	1	26
comp2	282	5	1509	13	7
comp3	35	11	13	678	0
comp4	79	2	44	0	858

The Recall and Precision scores are shown in **Error! Reference source not found.**. In addition to the values per class, the macro and weighted averages are displayed. The model achieved a good recall value of 0.89 whereas the precision is all below 0.6 which is a bit too small.

In Figure 12, the Recall and Precision scores are presented, providing insights into the model's performance. The scores are shown for each individual class, along with the macro and weighted averages.

The macro average calculates the average recall and precision across all classes by giving equal weight to each class. It provides an overall assessment of the model's ability to capture positive instances, irrespective of class imbalances. In our case, the macro average recall of 0.89 indicates that the model generally performs well in identifying positive instances across all classes. However, the macro average precision, which is below 0.6, suggests that the model has a relatively higher rate of false positives across the classes, especially for component 1.

On the other hand, the weighted average considers the class distribution in the dataset. It calculates the average recall and precision, considering the number of instances in each class. The weighted average scores are close to perfect.

Overall, these results indicate that while the model shows a good overall recall, which is important for capturing positive instances, there is room for improvement in terms of precision. This highlights the need for further optimization. Next, to further enhance the performance of our model, we will apply a hyperparameter optimization (HPO) technique. HPO allows us to systematically search for the optimal set of hyperparameters that maximize the model's performance on our specific task.

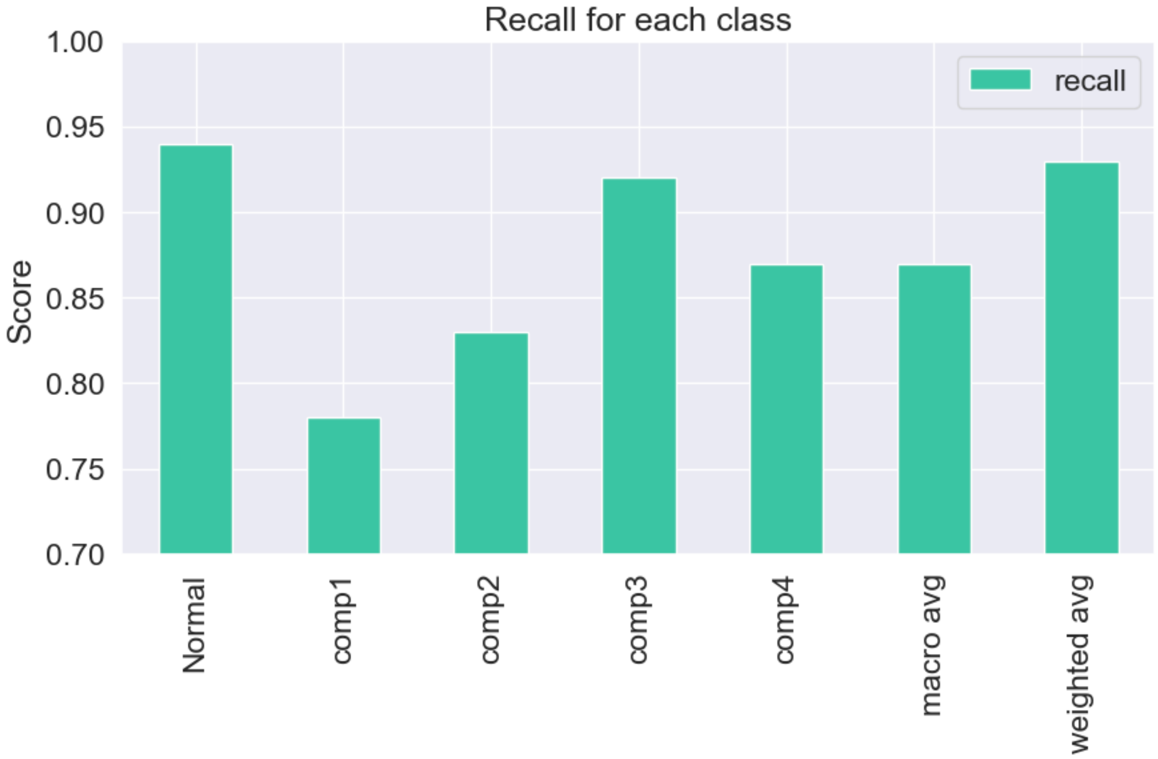

Figure 11 Recall and Precision scores of the MLP model

6.4.2 Hyperparameter Optimization

Hyperparameter optimization (HPO) was conducted on our multi-class classification MLP model using the Optuna framework [12]. The objective was to find the best hyperparameter configuration that maximizes the model's performance. We focused on the following hyperparameters during the HPO process:

Table 4 Hyperparameter search space

Hyperparameter	Search Space
Learning Rate (lr)	[1e-5, 1e-1] (log scale)
Hidden Layers	[2, 5]
Number of Units	[64, 512]
Dropout Rate	[0.0, 0.5]

To efficiently explore the hyperparameter space, we employed the TPESampler, which uses a Tree-structured Parzen Estimator algorithm [13] and employs a Bayesian optimization approach. This sampler intelligently selects hyperparameter values based on past trials, focusing the search on regions with higher potential for improvement. Additionally, we applied early stopping and pruning techniques to stop non-promising trials early, optimizing computational resources and speeding up training.

We performed 25 trials and 15 epochs with different combination of hyperparameters. The best hyperparameters were:

Table 5 Best parameters after performing hyperparameter optimization

Hyperparameter	Best values
Learning Rate (lr)	2.06e-5
Hidden Layers	2
Number of Units Layer 1	147
Number of Units Layer 2	153
Dropout Rate	0.5

The macro-average recall on the test set showed a notable improvement, increasing from 0.87 to 0.91. All the recall values for all the components were improved by the optimization.

Table 6 Performance comparison (Recall) between the MLP models with and without HPO

	MLP_NO_HPO	MLP_HPO
Normal	0.94	0.87
comp1	0.78	0.87
comp2	0.83	0.91
comp3	0.92	0.97
comp4	0.87	0.95
macro avg	0.87	0.91
weighted avg	0.93	0.87

6.4.3 LightGBM

In addition to the MLP model, we also explored the application of LightGBM, a gradient boosting framework known for its efficiency and effectiveness in handling tabular data [14]. LightGBM has gained popularity in various data science competitions, including Kaggle, where it has been used in winning solutions.

For our experiment, we trained a LightGBM model using the default parameters without performing hyperparameter optimization. Even without optimization, the results demonstrated superior performance compared to the optimized MLP model. A macro average recall of 0.96 was obtained, with all the individual recall values being above 0.93. The confusion matrix is shown in Table 7. The performance recall value comparison of the MLP model (with and without HPO) with LightGBM is presented in Figure 2. The LightGBM model significantly outperformed the MLP. In addition,

Table 7 Confusion matrix of the LightGBM model

	Normal	comp1	comp2	comp3	comp4
Normal	115779	761	843	130	354
comp1	28	1021	39	1	14
comp2	68	2	1704	15	27
comp3	10	2	1	722	2
comp4	6	7	17	0	953

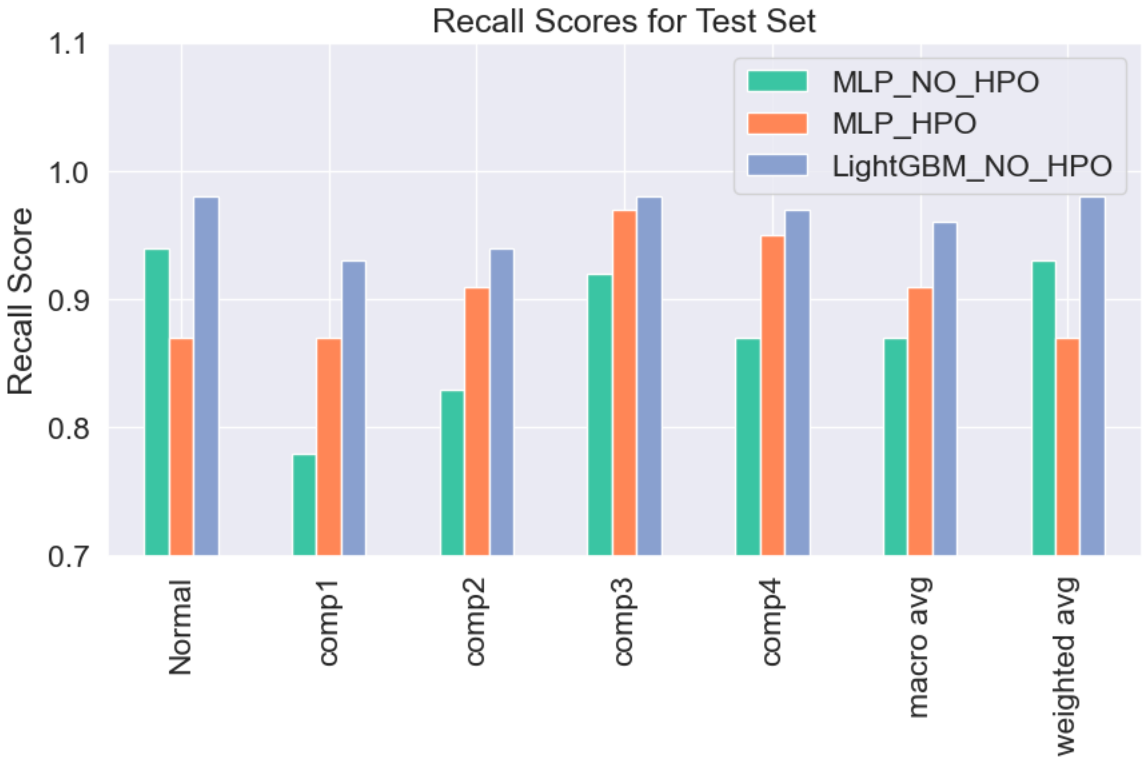

Figure 12 Performance comparison of the model

The success of LightGBM can be attributed to its ability to effectively capture complex patterns and interactions within the data through the use of gradient boosting and decision tree-based approaches. This is particularly advantageous in the context of tabular data, where features can exhibit nonlinear relationships and interactions. The performance of LightGBM in our experiment highlights its suitability for our predictive maintenance problem. It is important to note that running the MLP optimization on a more powerful machine, especially one with a GPU, has the potential to further improve the results.

Overall, the LightGBM model outperformed the MLP model in terms of both computational efficiency and prediction accuracy, showcasing its potential as a robust modeling approach for our multi-class classification task.

Another added benefit of using the LightGBM model is its ability to provide some inherent model explainability. LightGBM offers a feature importance, which allows us to see the importance of different features in the prediction process. Figure 13 demonstrates that the short-term (4-hour) telemetry lag features play a crucial role in predicting failures. Voltage is the most informative.

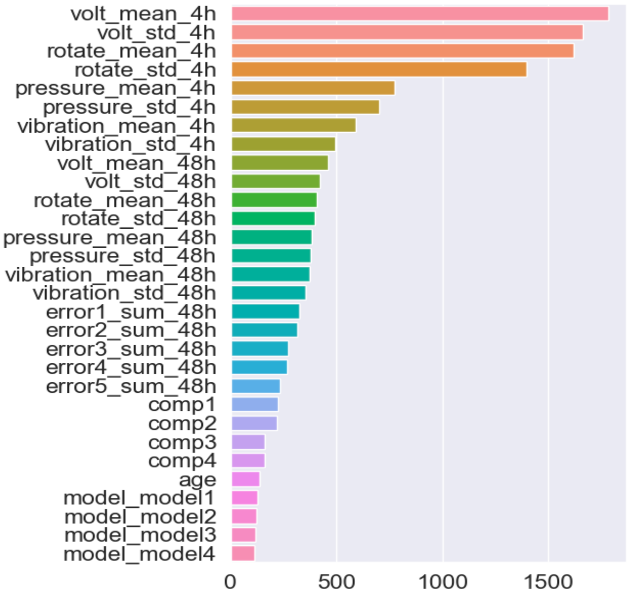

Figure 13 Feature importance

7 Conclusion

This project focuses on developing a machine learning model for predicting machine failures within a 48-hour time window. Leveraging the Microsoft Azure Predictive Maintenance dataset and conducting exploratory data analysis, valuable insights and patterns have been uncovered. The feature engineering process incorporated lag features, maintenance history, and machine-specific features capturing relevant information for failure prediction.

Due to the rare occurrence of failures, there was a significant class imbalance in the dataset. We addressed this challenge by using a class weighting technique, assigning higher weights for the minority failure classes during training.

Following a careful data splitting into train, validation, and test sets, we employed an MLP model and LightGBM model for modeling the multi-class failure classification task. Since the cost of missing a failure is too high, we used macro-average recall as our main evaluation metric. We started with some manually selected hyperparameters for the MLP model and obtained a relatively good recall for all components and an average recall of 0.87. We then performed a hyperparameter optimization with the Optuna framework and were able to obtain an improvement in the performance. The macro average recall increased to 0.91. To improve the performance by exhausting the full search space, one should perform a relatively big experiment, preferably with a GPU. Finally, we trained a LightGBM model with the default parameters and obtained an impressive performance of macro average recall 0.96. As a result, given its superior performance, simplicity, and explainability, we highly recommend using LightGBM as the final model for our problem.

In conclusion, in this project we took the challenge of predicting the likelihood of a failure happening due to a component in the next 48 hours and showed that extremely good results could be obtained using advanced machine learning algorithms.

8 Appendix

8.1 Source code

The source code for the project is available at:-

Disclaimer about ChatGPT Usage

I used ChatGPT for two primary purposes during my project. Firstly, I employed it as a tool for rephrasing and refining my draft, which helped enhance the clarity and coherence of my content. Secondly, I leveraged ChatGPT to seek assistance with coding-related inquiries. Whenever I encountered challenges or had questions about specific coding concepts, I relied on ChatGPT to provide insights and explanations.

9 Bibliography

[1] L. Swanson, "Linking maintenance strategies to performance," *International Journal of Production Economics,* vol. 70, no. 3, pp. 237-244, 2001.

[2] T. Zonta, C. A. d. Costa, R. d. R. Righi, M. J. d. Lima, E. S. d. Trindade and G. P. Li, "Predictive maintenance in the Industry 4.0: A systematic literature review," *Computers & Industrial Engineering,* vol. 150, 2020.

[3] J. Zhao, C. Gao and T. Tang, "A Review of Sustainable Maintenance Strategies for Single Component and Multicomponent Equipment," *Sustainability,* vol. 14, no. 5, 2022.

[4] W. Zhang, D. Yang and H. Wang, "Data-Driven Methods for Predictive Maintenance of Industrial Equipment: A Survey," *IEEE Systems Journal,* vol. 13, no. 3, pp. 2213-2227, 2019.

[5] R. B. &. P. E. S. D'Agostino, "Tests for departure from normality.," *Biometrika,* vol. 60, no. 3, pp. 613-622, 1973.

[6] Arnab, "Kaggle," [Online]. Available: https://www.kaggle.com/code/arnabbiswas1/predictive-maintenance-exploratory-data-analysis. [Accessed 20 04 2023].

[7] D. Cardoso and L. Ferreira, "Application of Predictive Maintenance Concepts Using Artificial Intelligence Tools," *Applied Sciences,* vol. 11, no. 1, 2021.

[8] M. K. Bora, "medium," 27 06 2022. [Online]. Available: https://medium.com/@Medini_2020/predictive-maintenance-using-machine-learning-3d8b62d5df8e. [Accessed 02 05 2023].

[9] [Online]. Available: https://www.kaggle.com/datasets/arnabbiswas1/microsoft-azure-predictive-maintenance.

[10] N. V. Chawla, K. W. Bowyer and L. O. H. W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of artificial intelligence research,* vol. 16, pp. 321-357, 2002.

[11] V. G. Nado, G. E. Dahl, J. Gilmer, C. J. Shallue and Zachary, "Deep Learning Tuning Playbook," 2023. [Online]. Available: https://github.com/google-research/tuning_playbook.

[12] T. Akiba, S. Sano, T. Yanase, T. Ohta and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," in *Proceedings of the 25th {ACM} {SIGKDD} International Conference on Knowledge Discovery and Data Mining*, 2019.

[13] J. Bergstra, R. Bardenet, Y. Bengio and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems,* vol. 24, 2011.

[14] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye and T.-Y. Liu, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," *Advances in Neural Information Processing Systems,* pp. 3146-3154, 2017.