Smart Predictive Maintenance for Industrial Equipment

Advanced Machine Learning I

Alessia SARRITZU
Alberto MARTINELLI
Marco RIVA
Michele PULVIRENTI

January 18, 2025

Contents

1 Introduction

This project was centered on developing a predictive maintenance system for industrial machinery.

We utilized the CMAPSS (Commercial Modular Aero-Propulsion System Simulation) dataset, which provided a practical and rich framework for modeling the Remaining Useful Life (RUL) of machinery components.

The project involved several stages, beginning with systematic preprocessing of the data. This step included handling missing values, scaling, and engineering features to enhance the predictive capability of the models. Following this, various machine learning models were developed and tested to classify engine conditions and estimate the RUL, with techniques such as AutoML employed to optimize model performance. Finally, a reinforcement learning-based approach was implemented to schedule maintenance tasks, effectively balancing the trade-off between minimizing downtime and avoiding unexpected failures.

The report outlines these steps in detail, from data preprocessing and model development to the evaluation of global results.

2 Data

This section illustrates the chosen dataset, its first analysis and the preprocessing steps applied to it.

2.1 Dataset Selection

The CMAPSS (Commercial Modular Aero-Propulsion System Simulation) dataset was selected for this project due to its comprehensive and realistic representation of sensor data from aero-engine systems. This dataset is widely used in the field of predictive maintenance and is particularly suited for developing models to estimate the Remaining Useful Life (RUL) of machinery components.

The dataset is provided by NASA's Prognostics Center of Excellence, and can be found at this page.

2.2 Dataset Exploration

The dataset contains sensor readings and operational settings recorded over the operational life cycles of multiple engines. Each engine runs until a failure occurs, and the dataset includes training and testing data, along with the remaining useful life for the test engines.

The dataset is composed of 4 subsets, each one containing the data of a different set of engines. As first step, the subsets have been merged into a single dataset, so to have the full range of data to explore. The `engine_id` is renumbered so not to loose the information about the original subset.

Additionally, two columns, `conditions` and `fault_mode`, are added to the dataset. These fields are described in the `txt` file contained within the dataset folder and originally not included within the values, but just attached as meta information among each different subset.

The following list summarizes the key features of the dataset:

- **Engine ID**: unique identifier for each engine.
- **Cycle**: temporal index representing the engine's operational cycle.
- **Operational Settings**: five settings capturing various controlled operating conditions.
- **Sensor Readings**: twenty-one sensor features measuring engine performance.

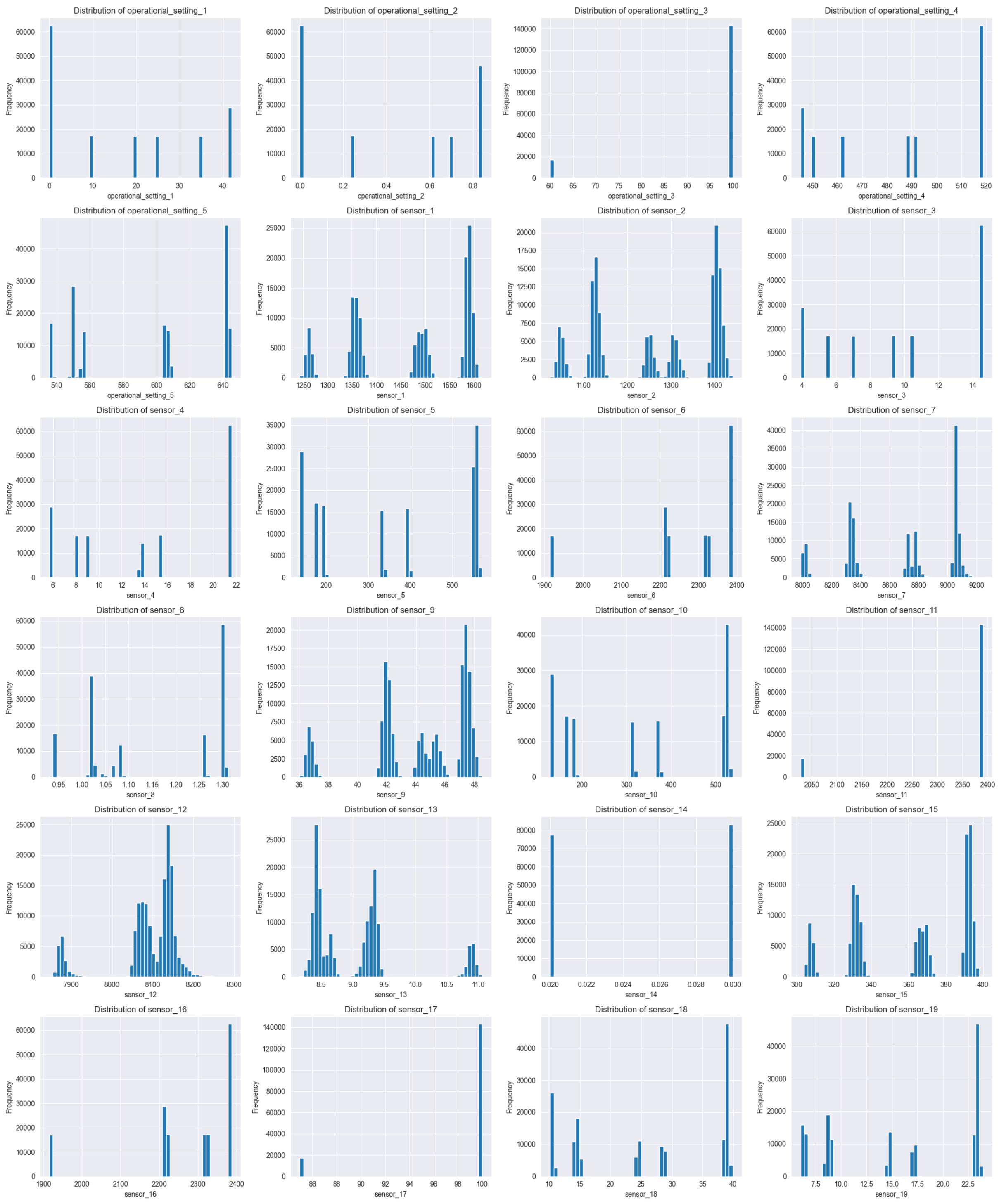
2.3 Exploratory Data Analysis

The initial exploratory data analysis (EDA) focused on understanding feature distributions and identifying potential anomalies or uninformative features. The following steps have been developed:

- **Statistics**: statistical summary that revealed meaningful variation in most sensor readings and operational settings: histograms of key features were generated to visualize their distributions, as shown in graph in the following page.
- **Missing Data**: columns `sensor_20` and `sensor_21` were completely missing and were removed from the dataset.
- **Correlation Matrix**: the correlation matrix of the full set of features was analyzed, so to understand the relationships between the different sensors and operational settings.
- **Class Imbalance**: class imbalance was visualized and assessed.

2.3.1 Data Distribution

In order to visualize the dataset features' range of values, an histogram of each of them was produced:



Distributions of Operational Settings and Sensor Readings

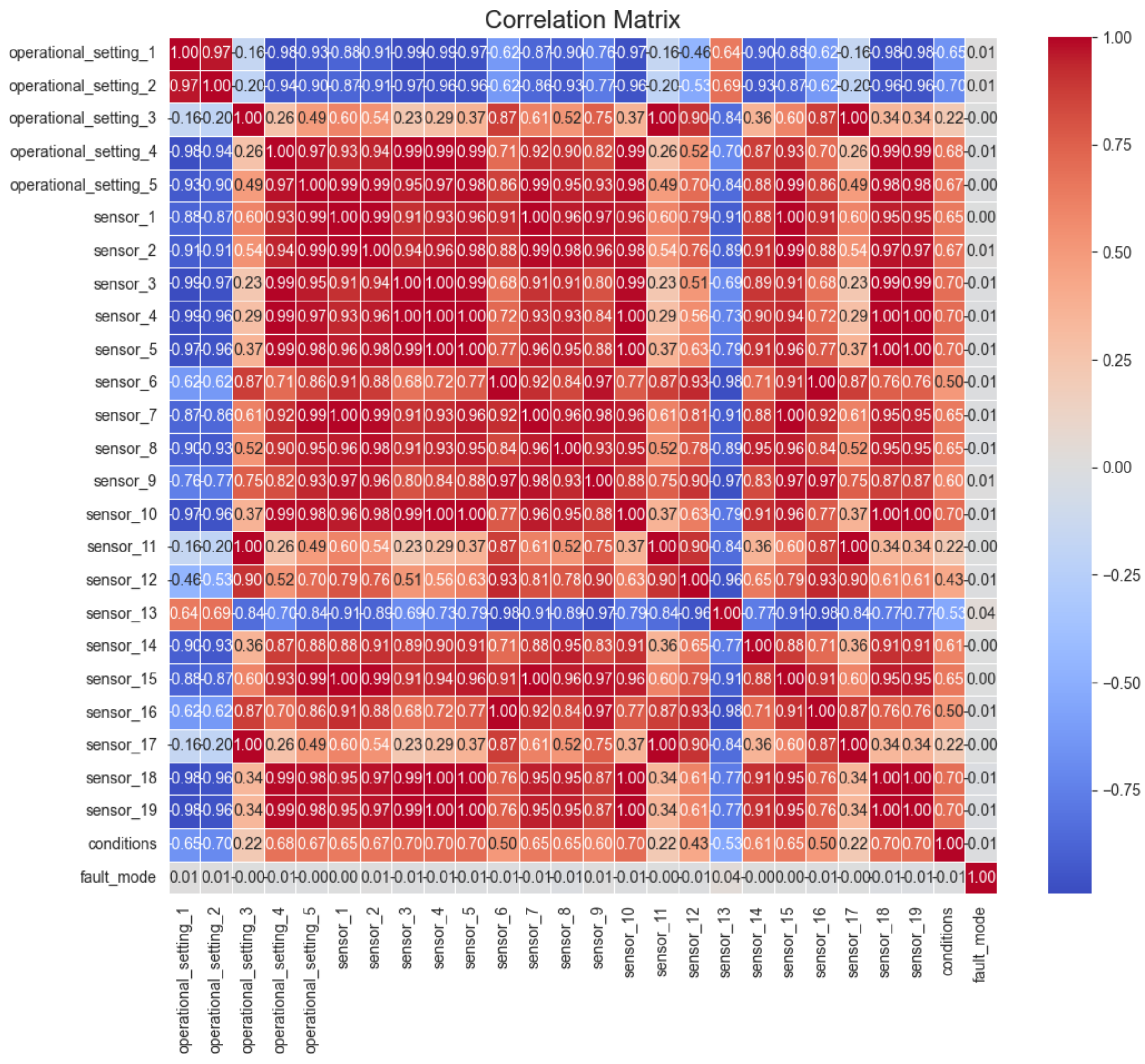
From the histograms we can evaluate that features like `operational_setting_1`, `operational_setting_2`, and `operational_setting_5` exhibit discrete values clustered into distinct groups. While this may suggest that these operational settings could potentially be treated as categorical variables, this would make them lose their ordinal nature.

Additionally some sensors have very narrow distributions, with most values concentrated in a small range or on a single mode. On the other hand, other sensors exhibit multimodal distributions, suggesting they capture distinct operating states of the engines.

Finally, we can notice that most of the operational settings have highly restricted ranges, with most values concentrated in a single cluster. This could mean these settings remain constant or change rarely, possibly having limited relevance for failure prediction.

2.3.2 Correlation Matrix

The correlation matrix is crucial to understand the relationships among features:



The correlation matrix reveals several important observations regarding feature correlations. First, some features exhibit strong positive or negative correlations, indicating potential redundancy; for example, sensors with correlations above 0.8 may carry overlapping information, which could be simplified. On the other hand, several features show near-zero correlations, suggesting independence. These independent features might capture unique aspects of the data and be valuable for predictive modeling.

Additionally, the operational settings generally display weaker correlations with sensor readings, indicating that they capture distinct aspects of engine performance and could provide complementary information.

Finally, groups of strongly correlated features form blocks in the matrix, suggesting clusters of sensors or conditions that behave similarly.

2.4 Preprocessing

In order to prepare the dataset for the subsequent training set, first it was necessary to identify the kind of problem we were trying to solve. The following sections explain the analysis that lead to the final decision made and the steps involved in the dataset preparation.

2.4.1 Problem Framing

The initial approach to the project framed the problem as a regression task, where the goal was to predict the Remaining Useful Life of each engine directly as a continuous variable. This approach was intuitive given the nature of the dataset, which contains time-series data for each engine from a healthy state to failure, making it suitable for modeling RUL as a continuous outcome. Regression provides granular predictions, which are advantageous in contexts where precise estimation of RUL is critical.

However, upon evaluating the initial results and considering the tasks to be performed, we observed challenges in maintaining prediction stability and interpretability in the regression model. Specifically:

- Due to the variability in engine behavior, regression results exhibited significant variance fluctuation, especially for engines nearing failure.
- The ultimate goal of the project was to optimize maintenance schedules by identifying whether an engine was at risk of failure within a specific time frame, rather than predicting an exact RUL.
- Classification allows for a straightforward interpretation, focusing on whether an engine is operating within a "safe" range or requires maintenance.

Given these considerations, we framed the problem as a binary classification task.

2.4.2 RUL Definition

To treat the problem as a classification task, the following preparatory steps have been taken:

1. **RUL Column creation**: The RUL for each engine at every cycle was calculated, subtracting the current cycle number from the cycle at which failure occurred.
2. **Threshold Definition**: A threshold (with the value of 20) was established to define two classes:
   - **Class `healthy`**: the engine is operating normally and is not at immediate risk of failure.
   - **Class `unhealthy`**: The engine is nearing failure and requires maintenance.
3. **Dataset labeling**: Each data point was labeled as 0 or 1 based on whether its RUL was above or below the defined threshold.

2.4.3 Feature scaling

Given the varying ranges of the sensor readings and operational settings, feature scaling was a necessary step. Scaling ensured that no single feature disproportionately influenced the model due to its magnitude.

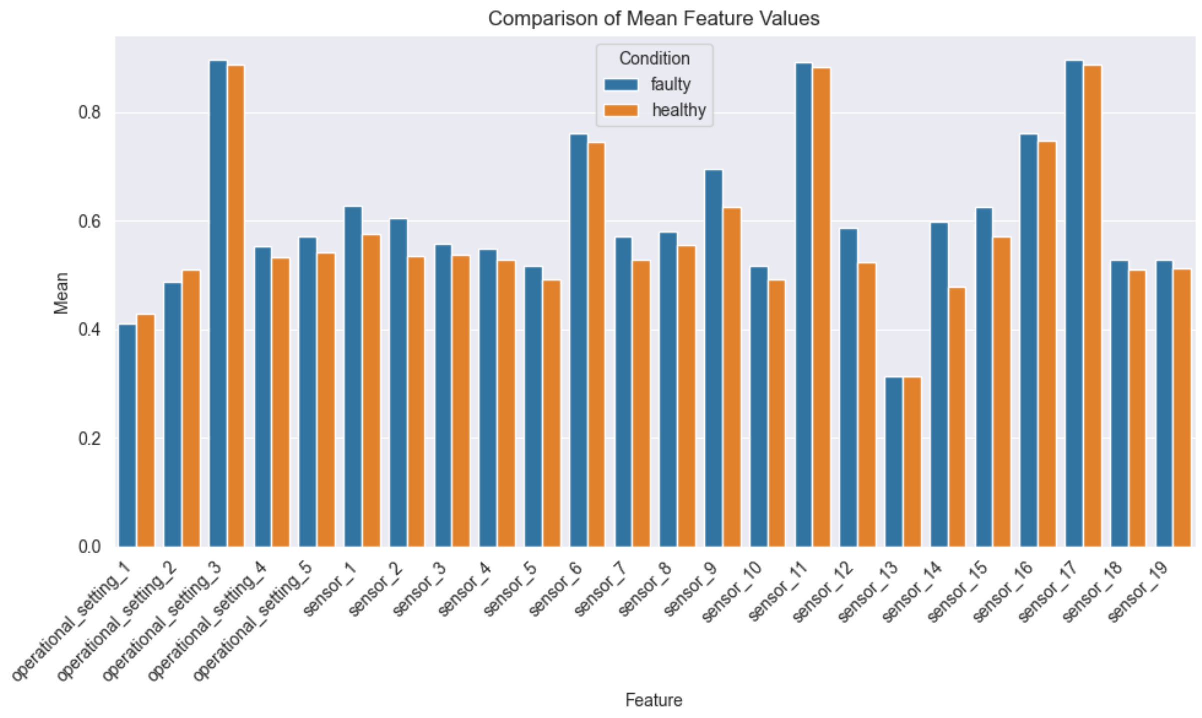
After some considerations, all features were scaled to a range between 0 and 1 using min-max normalization, thanks to the `MinMaxScaler`. This step preserved the distribution of the data while standardizing the scale of all features.

Algorithms such as logistic regression ones are sensitive to feature magnitudes: scaling improved convergence during training and ensured fair weight distribution across features.

2.4.4 Feature engineering

Feature engineering has been performed to address the high correlation observed between several variables within the dataset. The goal was to create more distinguishable data between the sensors.

As first step, the means of the features have been displayed in order to select the most relevant ones, splitting them between values among the two classes.



Comparison of Mean Features Values

After analyzing the dataset, we can notice that some features have a significant mean value difference between the two classes, suggesting that they could be useful for distinguishing between healthy and faulty engines.

These kind of features were selected for creating rolling averages: after some careful considerations, the sensors `4`, `11` and `14` were selected for the rolling window, with a size of 20.

3 Model Training

To predict engine failures based on the processed data, different models have been tested. The objective was to identify the model that provided the best balance performance metrics, ensuring reliable predictions for maintenance decision-making.

Each model was evaluated using a consistent set of metrics to provide a comprehensive assessment of its performance. These metrics included:

- **Classification report**: displays precision, recall, F1-score, and support for both classes.
- **Roc Curve**: visualizes the trade-off between true positive rate (sensitivity) and false positive rate, along with the area under the curve (AUC) as a summary statistic.
- **Confusion Matrix**: offers a granular view of model predictions, highlighting true positives, true negatives, false positives, and false negatives.

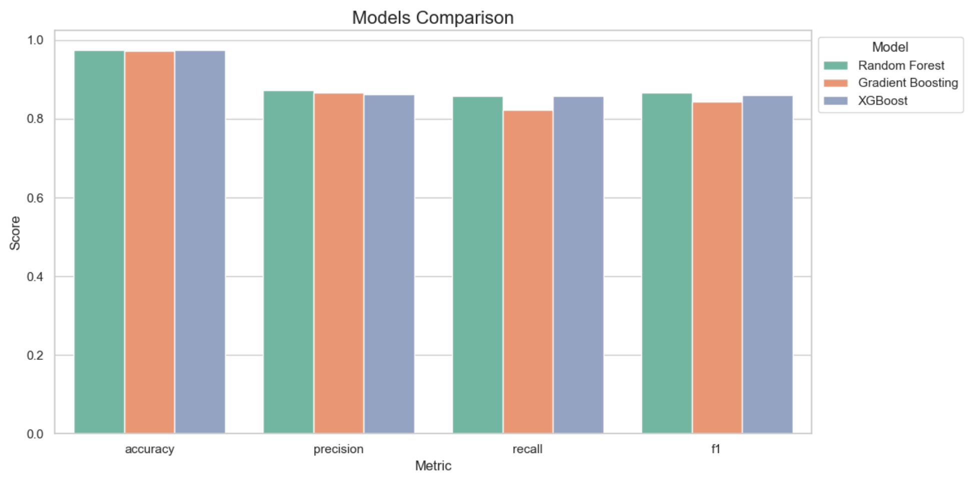
Given the nature and the complexity of the task and the dataset, as main reference metric the `F1-score` was chosen.

3.1 Ensemble Learning: first attempt

Three ensemble learning models have been chosen to be tested:

- **Random Forest**
- **Gradient boosting Boosting**
- **XGBoost**

The following bar plot summarizes and compare their performances for their first implementation attempt:



The plot shows that the three models perform similarly, with the Random Forest model slightly outperforming the other two models in terms of accuracy and F1 score. Additionally, it can be noted that the general performance of the models is already quite good, even without having handled the class imbalance in the dataset.

3.2 Handling Imbalanced data

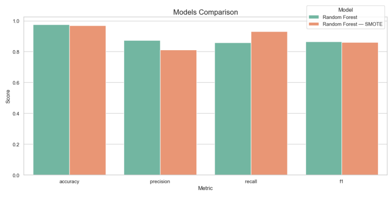
Although the initial performance of the tested models was already promising, we explored techniques to handle imbalanced data in the dataset to investigate whether further improvements could be achieved. Imbalanced datasets, where one class is significantly underrepresented, can lead to biased models that favor the majority class, potentially overlooking critical instances of the minority class.

Different methods were employed to address the class imbalance. The effects of these techniques were analyzed to understand their contribution to model performance. Each technique has been evaluated on the three models taken into account for the initial training, comparing the new performance with the original one.

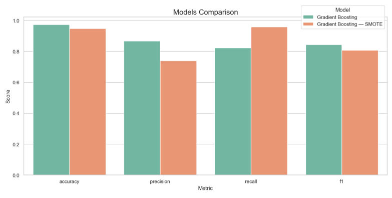
3.2.1 Oversampling

The oversampling was performed using Synthetic Minority Oversampling Technique (SMOTE), to address class imbalance in the training dataset: it balances the dataset by generating synthetic samples for the minority class, resulting in an equal distribution of classes.

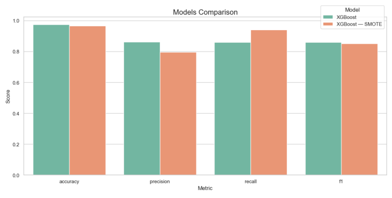
The following plots show the performance of each of the three models, compared with their original version.



Random Forest



Gradient Boosting



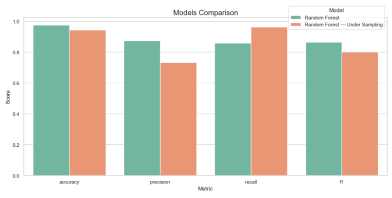
XGBoost

3.2.2 Undersampling

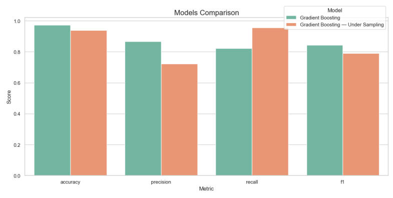
Undersampling was applied as an alternative approach to oversampling. Undersampling aims to balance the dataset by reducing the number of instances in the majority class.

The `RandomUnderSampler` method was used for this purpose. This resulted in a balanced dataset where both classes had the same number of instances.

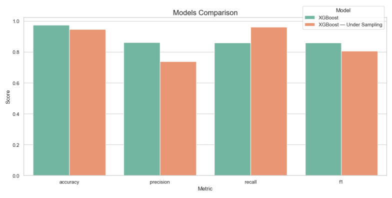
The following plots show the performance of each of the three models, compared with their original version.



Random Forest



Gradient Boosting

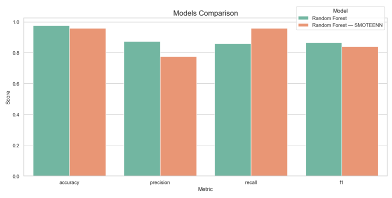


XGBoost

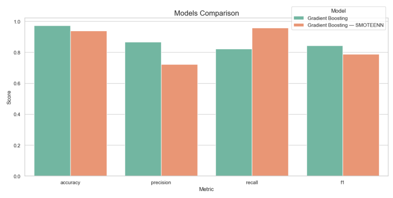
3.2.3 SMOTEENN

Another attempt was performed with `SMOTEENN`, a hybrid approach that combines oversampling using SMOTE and undersampling using Edited Nearest Neighbors. This method balances the dataset while also removing noisy samples, potentially improving model performance. This process ensures a balanced dataset with reduced noise.

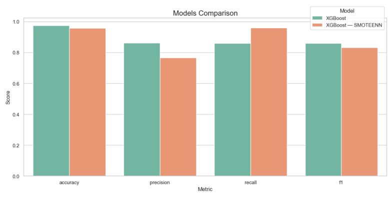
The following plots show the performance of each of the three models, compared with their original version.



Random Forest



Gradient Boosting



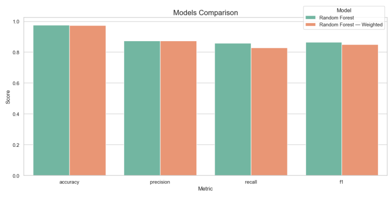
XGBoost

3.2.4 Cost Sensitive Learning

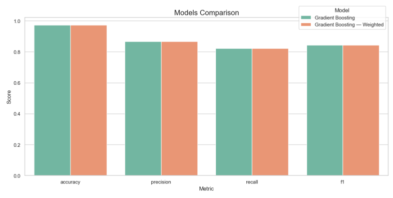
Cost-sensitive learning was the last strategy attempted to address class imbalance, by adjusting the weights of the classes in the loss function. This approach emphasizes the minority class by assigning a higher penalty to misclassifications, encouraging the model to pay more attention to instances in the minority class during training.

The `compute_class_weight` method of `sklearn` library was used to compute the weight of each class.

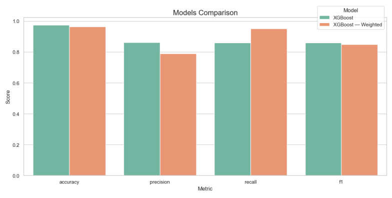
The following plots show the performance of each of the three models, compared with their original version.



Random Forest



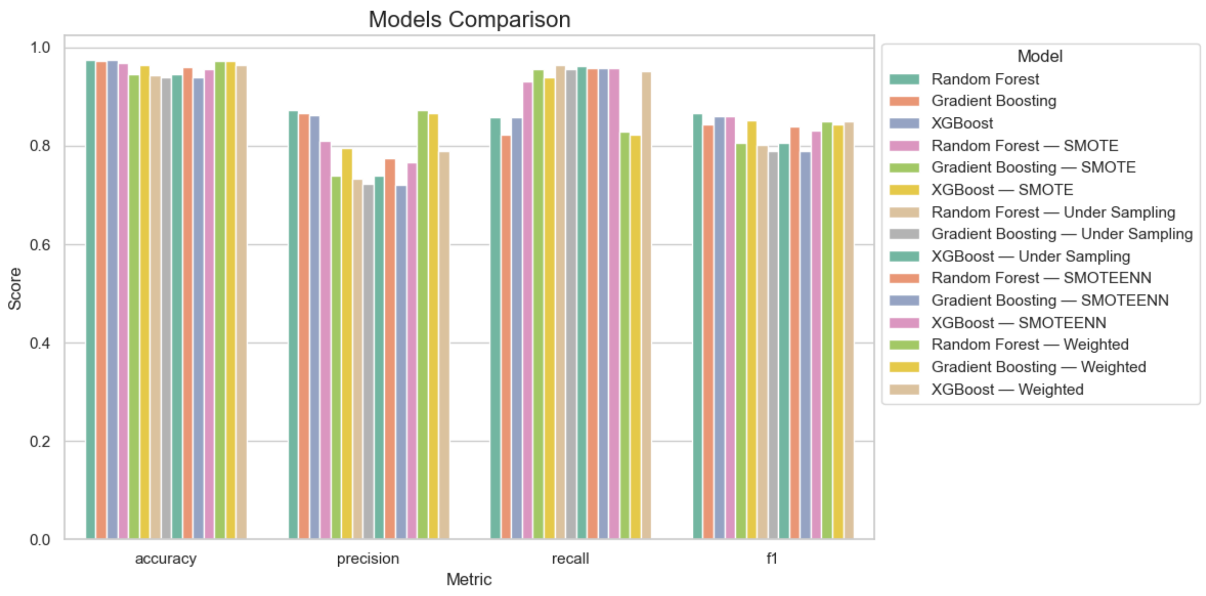
Gradient Boosting



XGBoost

3.2.5 Imbalanced Data Conclusion

All the trained models up to this point were plotted within a single graph, so to have a complete view of the achieved performance by each version.



Imbalanced Data: Model Comparison

All three models (Random Forest, Gradient Boosting, and XGBoost) exhibit similar performance trends across the different metrics, suggesting that the dataset properties have a uniform impact on these algorithms.

`Accuracy` scores are consistently high for all models and dataset-handling techniques. However, since the dataset is imbalanced, as stated before, accuracy alone might not be a reliable indicator of performance. `F1-score` might provide a more comprehensive evaluation.

It can be noticed that `recall` improves significantly when SMOTE is applied, particularly for Gradient Boosting and XGBoost. This indicates that SMOTE helps in correctly identifying the minority class (failures), reducing false negatives. However, there's a slight trade-off in `precision`, as synthetic samples might introduce noise.

On the other hand, models trained with SMOTEENN achieve a good balance between `precision` and `recall`, resulting in high `F1-scores`. This suggests that SMOTEENN effectively cleans noisy synthetic samples created by SMOTE while improving the minority class's representation.

Comparing the models transversely:

- **Random Forest**: Performs consistently well across all metrics but slightly lags behind Gradient Boosting and XGBoost in terms of **recall**. This could indicate that Random Forest struggles slightly more with the imbalanced dataset.
- **Gradient Boosting**: Performs better than Random Forest in **recall**, especially when SMOTE or SMOTEENN is applied. Gradient Boosting appears to benefit

more from sampling techniques that balance the dataset.

- **XGBoost**: XGBoost achieves the highest scores across most metrics, particularly when SMOTEENN or class weighting is applied. Its robustness and ability to handle imbalanced datasets make it the top-performing model in this comparison, making it a candidate as the best model among the three.

4 Maintenance

Reinforcement Learning was employed to optimize predictive maintenance schedules. The goal was to develop a system that learns to balance the costs associated with performing maintenance and the consequences of unexpected failures.

Reinforcement learning provides a dynamic and adaptive approach to maintenance scheduling by allowing the model to interact with a simulated environment. Through this interaction, the model can learn optimal policies for deciding when maintenance actions should be taken, based on the current condition of the machinery and predictions of failure likelihood.

The following sections describe the key components of the Reinforcement Learning strategy.

4.1 Environment Definition

The Reinforcement Learning environment has been defined with the following characteristics:

- **State Space**
  The environment uses a discretized state space via `KMeans` clustering of sensor data. Each state represents a cluster that captures similar operational conditions. This simplifies the state representation while preserving the most relevant features for decision-making.
- **Action Space**
  The agent can choose between two actions: 0 (No Maintenance) and 1 (Perform Maintenance). This directly aligns with maintenance scheduling decisions.
- **Reward System**
  The reward structure penalizes failures heavily, incentivizes proactive maintenance in high-risk conditions (low RUL), and provides positive rewards for avoiding unnecessary maintenance in safe states.
  The scaled rewards align with minimizing costs associated with downtime (failures) and maintenance.

4.2 Learning

The Q-learning algorithm has been used.

The defined algorithm follows these steps:

- Explores different state-action pairs (via epsilon-greedy action selection).
- Updates the Q-table to reflect the cumulative reward for each state-action pair.
- Decays the exploration rate (epsilon) to transition from exploration to exploitation over time.

4.3 Reinforcement Learning Results

The learned Q-table has been evaluated to measure the average reward of the trained policy. Only a subset of the dataset (20%) is used for training due to the computational demand of the Q-learning algorithm.

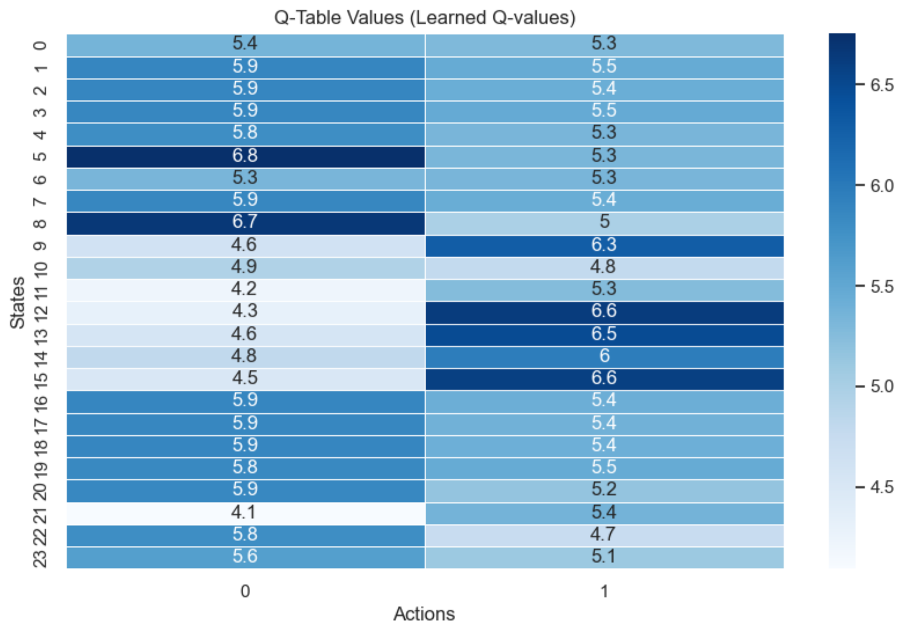
The achieved average reward scoring was equal to 14023.

This scoring suggests that the policy performs well, avoiding significant failures and managing maintenance effectively. To confirm this result, we compared it against a random policy baseline. Its scoring was of 5120.

The significantly higher score of the learned policy, compared to the random one, confirms that the agent has learned an effective predictive maintenance strategy. It successfully minimizes failures and avoids unnecessary maintenance.

4.3.1 Q-Table Visualization

To further inspect the obtained result, the policy can be visualized:

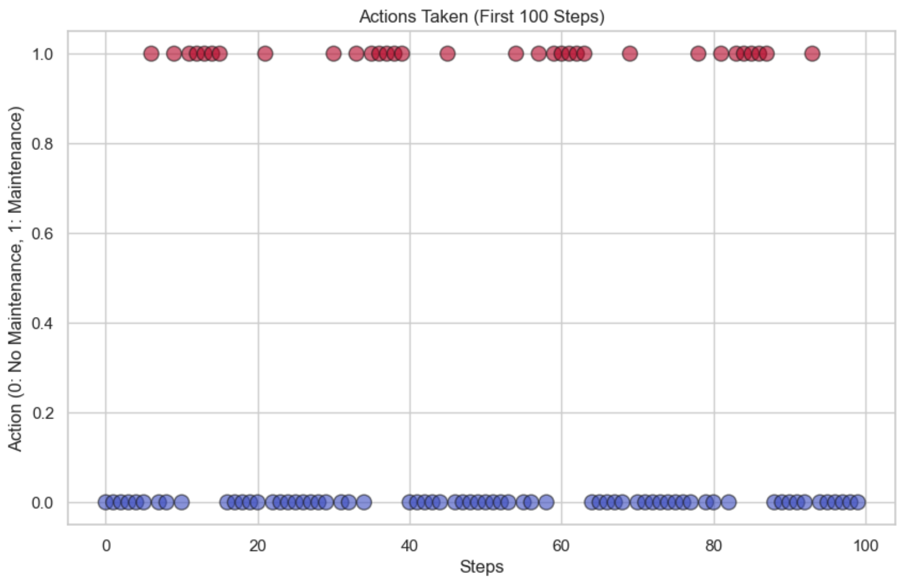


Observing the Q-table, we can notice that Q-values are higher for action 1 (maintenance) in some states, such as states 10, 14, and 15. This suggests that performing maintenance in these states is preferred, likely because they represent conditions with a low RUL. Q-values are generally balanced in several states where the values for both actions are relatively close. This indicates that the agent perceives little difference between performing maintenance or not in these states, suggesting they may represent healthy operational conditions with a higher RUL. Additionally, clusters of higher or lower Q-values can be observed in certain regions of the Q-table, indicating distinct clusters of states where maintenance or no maintenance is generally preferred.

In conclusion, the agent has successfully learned a policy by associating specific actions with states to maximize the cumulative reward. The distribution of Q-values reflects the reward structure. For example, high penalties for failure push the agent to prefer maintenance in high-risk states, while maintenance costs incentivize avoiding unnecessary maintenance in low-risk states.

4.3.2 Policy Visualization

As last analysis, we can visualize the policy in order to evaluate its effects:



Observing the above graph, we can notice that frequent switching between actions is observed, where the agent alternates frequently between the two actions.

This suggests the policy is dynamically reacting to the current state and reward system. A predictable pattern emerges, as the agent tends to perform maintenance (action 1) after periods of no maintenance (action 0). This could correspond to when the agent anticipates a failure.

Furthermore, proactive maintenance can be observed, as maintenance is not evenly distributed but concentrated in specific intervals. This indicates that the agent is making decisions based on the state or Remaining Useful Life.

5 Model Selection

Model selection and hypermarameter tuning were also performed, in order compare the previous model with an hypertuned one.

The process consisted of an AutoML (Automated Machine Learning) process, which automates the process of model selection and hyperparameter tuning.

5.1 Auto ML

For the automated ML process, the `TPOT` library has been used.

The classifier has been optimized for the `F1-score`. A sufficient amount of enerations (`generations=10`) and population size (`population_size=50`) have been used to explore the search space effectively.

The resulting pipeline was automatically saved in a `.py` file, thanks to the `TPOT` library's `tpot.export()` function.

5.1.1 Result

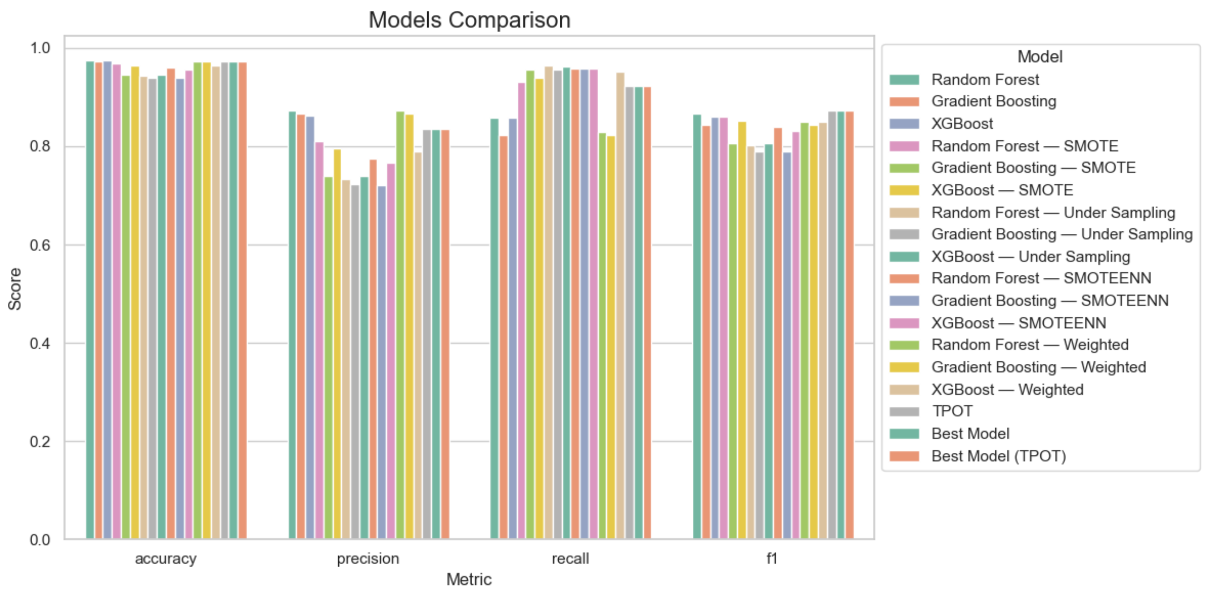
The AutoML process identified a robust pipeline that combines ensemble methods with stacking.

The final pipeline consists of a stacking estimator where an **Extra Trees Classifier** is used as the primary estimator, stacked with a **Gaussian Naive Bayes** classifier.

The scoring on the selected metric (`F1-score`) was of 0.76. This scoring seems in line with the previous models, but if we consider that the F1-score is a more reliable metric for imbalanced classification tasks, this could be a better result compared with the other models (see next section).

5.1.2 Comparison

All the previous models, with all the available metrics, have been plotted among the one resulting from this part of the project:



The evaluation shows that the AutoML model performs well, with a high F1-score and comparable performance to the previous models.

While `accuracy` and `recall` scores are similar to the previous models, and `precision` is actually generally not higher, the resulting `F1-score` is a bit higher compared to all the previous models. This metrics is probably the most important one, as it provides a balanced measure of a model's precision and recall.

This concludes that the found best model from the AutoML process is at least in line with the manually trained models, if not even slightly better.

6 Conclusion

6.1 Best Models

This section summarizes the best model found based on different criterion. As explained in the previous sections, the `F1-score` is considered as the main metric.

6.1.1 Per Metric

The best model for each metric has been identified:

- **F1 Score**: *TPOT* with F1 Score of 0.87343
- **Accuracy**: *Random Forest — base model* with Accuracy of 0.97525
- **Precision**: *Random Forest — base model* with Precision of 0.87243
- **Recall**: *Random Forest — Under Sampling* with Recall of 0.96381

The results show a general high quality among the (best) models, on all the metrics considered (F1, accuracy, precision, recall).

6.1.2 Per Type

The best model for each type has been identified based on the F1 score:

- **Random Forest**: *Random Forest — base model* with F1 Score of 0.86560
- **Gradient Boosting**: *Gradient Boosting — base model* with F1 Score of 0.84297
- **XGBoost**: *XGBoost — base model* with F1 Score of 0.85999

The results show that the base version of each of the types of models is the best performing one (based on the F1 score), suggesting that the base models are already quite effective in capturing the patterns in the dataset and that the class imbalance techniques adapted do not enough improvements to overtake the base versions of the models (again, considering the F1 score as the comparison metrics).

6.1.3 Per Technique

The best model for each technique has been identified based on the F1 score. This highlights the most effective approach for handling the class imbalance in the dataset, for each of the type of models considered.

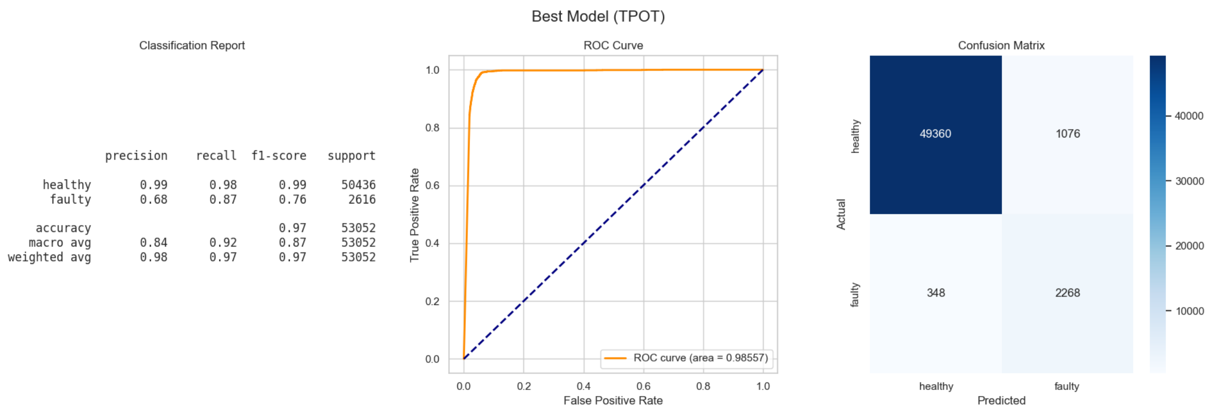
- **SMOTE**: *Random Forest* with F1 Score of 0.85966

- **Under Sampling**: *XGBoost* with F1 Score of 0.80673
- **SMOTEENN**: *Random Forest* with F1 Score of 0.83882
- **Weighted**: *Random Forest* with F1 Score of 0.84941

6.1.4 Overall

The best model overall has been identified (based on the F1 score).

The best model results to be the one obtained from the AutoML process. Its performance metrics are reported below.



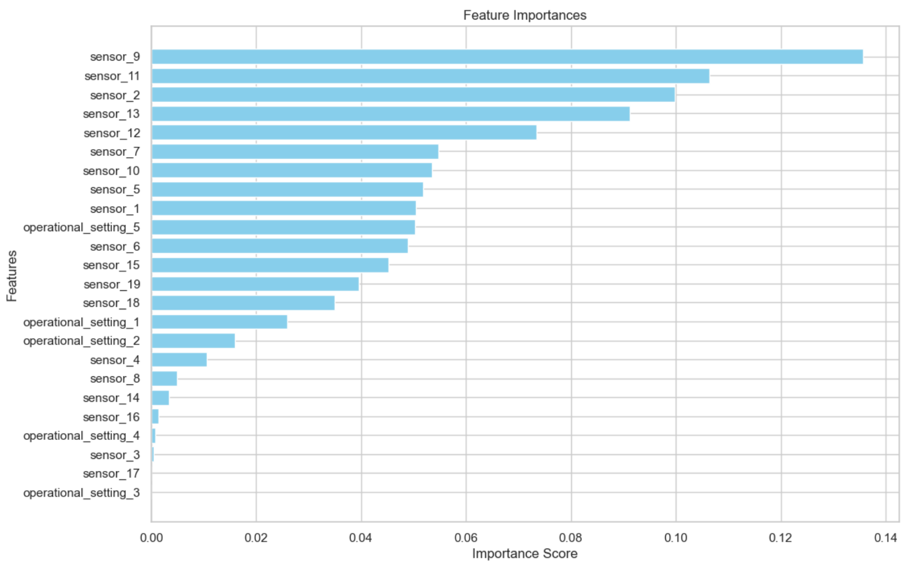
```
              precision    recall  f1-score   support

     healthy       0.99      0.98      0.99     50436
      faulty       0.68      0.87      0.76      2616

    accuracy                           0.97     53052
   macro avg       0.84      0.92      0.87     53052
weighted avg       0.98      0.97      0.97     53052
```

6.2 Feature Importance

The feature importance of the best version of the Random Forest model (which is the best performing model among the tree-based one) has been analyzed to understand which features have the most significant impact on the model's predictions.



We can notice that the most important features are `sensor_9`, `sensor_11`, `sensor_2`, `sensor_13`, and `sensor_12`. These sensors show a strong influence on the model's predictive performance, indicating their critical role in capturing engines behavior and early indicators of failure.

Features like `sensor_7`, `sensor_10`, `sensor_5`, and `sensor_1` exhibit moderate importance. While not as critical as the top features, these sensors still capture meaningful information.

Finally, `operational_setting_5` is the only operational feature with moderate importance, indicating that it captures some influence on engines failure.

6.3 Recommendations

This project demonstrated the potential of predictive maintenance systems in industrial settings by leveraging advanced machine learning techniques. Through comprehensive data preprocessing, feature engineering, and model evaluation, we developed a robust pipeline for classifying engines failures and optimizing maintenance schedules. The use of AutoML further streamlined the process, ensuring high-performance model selection and tuning. The reinforcement learning-based approach to maintenance scheduling effectively balanced operational downtime with the risk of equipment failure.

Some recommendations for future works can be made:

- Real-Time Systems: incorporate the developed predictive maintenance pipeline into real-time monitoring systems would enhance its practical applicability. This would allow continuous data collection and on-the-fly predictions for real-world scenarios.
- Alternative algorithms: investigate additional machine learning algorithms, such as deep learning models, which may capture complex patterns in sensor data more effectively.
- Domain-specific feature engineering: incorporate domain-specific features, that could improve prediction accuracy and interpretability.
- Cost optimization in scheduling: enhance the Reinforcement Learning framework to incorporate more complex cost models, including penalties for over-maintenance and resource constraints.
- Scalability and generalization: Explore the scalability of the approach to larger datasets and its adaptability to other industrial domains.

By addressing these areas, future works can build upon the foundation laid by this project, driving further advancements in predictive maintenance.