École Centrale Casablanca

Artificial Intelligence Project and Applications in Industry 4.0

Predictive Maintenance For Aircraft Engine

Work by :

NAHLI Ghita
SALEHI Abderrahmane

Supervised by :

Sabeur ELKOSANTINI

Mars 2024

Table des matières

1 Introduction

In the modern era of advanced machinery and transportation systems, ensuring optimal operational efficiency and safety is paramount. This is especially true in the aviation industry, where the reliability of aircraft engines directly impacts both performance and safety. The primary goal of predictive maintenance is to significantly reduce the costs associated with time-based maintenance strategies while preempting unexpected equipment failures. By accurately predicting when an engine requires maintenance, repair work can be scheduled effectively, minimizing downtime and ensuring the highest safety standards.

This report delves into the application of data science techniques to the domain of aircraft engine maintenance, utilizing sensor data to foresee in-service engine failures. It outlines a comprehensive approach to predictive maintenance, leveraging a rich dataset of aircraft engine sensors. These measurements serve as the foundation for our analysis, enabling the identification of potential failure points before they lead to functional breakdowns. The insights derived from this study aim to aid in decision-making processes, enhance maintenance scheduling, and ultimately, contribute to safer and more efficient aviation practices.

2 Why exactly Aircraft engine ?

Aircraft engines are a critical component of aviation safety and efficiency. The choice to focus on aircraft engines in a predictive maintenance project is strategic for several reasons :

- **Reduce flight delays**
- **Safety :** The primary concern in aviation is safety. Engine failure during flight can have catastrophic consequences. Predictive maintenance helps identify potential issues before they lead to failure, significantly enhancing safety.
- **Cost Efficiency :** Aircraft engines are incredibly expensive to repair or replace. Predictive maintenance can reduce these costs by addressing problems before they escalate into major repairs or require engine replacement.
- **Operational Reliability :** Airlines and aviation services strive for minimal downtime. Unexpected engine failures can lead to flight delays or cancellations, affecting operational efficiency and customer satisfaction. Predictive maintenance helps in scheduling repairs during non-operational periods, thus minimizing disruption.
- **Complexity and Data Availability :** Modern aircraft engines are equipped with numerous sensors that collect extensive data on various parameters. This wealth of data is ideal for data science techniques, providing a comprehensive view of engine health and functioning.
- **Longevity of Equipment :**Regular maintenance based on predictive analytics can extend the life of aircraft engines, ensuring they operate efficiently for a longer period.

3 Project overview and objectives

In this project, our overarching goal is to enhance airline operations and minimize flight delays by accurately predicting aircraft engine failures in advance. Leveraging the wealth of data gathered from engine sensors and telemetry, we aim to transform traditional maintenance approaches by implementing a predictive maintenance framework.

This framework is centered around the concept of Time-To-Failure (TTF) prediction or RUL(Remaining Useful Life), allowing maintenance scheduling to be precisely aligned with actual engine needs rather than relying solely on routine time-based checks. By analyzing sensor data over time, our ma-

FIGURE 1 – Aircraft engine's maintenance

chine learning models are designed to uncover intricate patterns and correlations between various sensor readings and historical engine failures. This project employs a suite of supervised machine learning techniques, each tailored to different aspects of predictive maintenance :

- Regression algorithms for quantifying engine TTF.
- Binary Classification models to determine the likelihood of engine failure within a specific operational period.

These predictive insights will not only boost operational efficiency but also play a pivotal role in ensuring flight safety, reducing repair costs, and adhering to stringent regulatory standards in the aviation industry.

4 Approach

In our approach to predictive maintenance for aircraft engines, we begin by sourcing detailed sensor and settings data from the **Nasa Datasets**. This rich dataset serves as the foundation of our analysis, encompassing a broad spectrum of parameters essential for understanding engine health and performance.

The initial phase of our project : cleaning and preparing this data, a crucial step that ensures the accuracy and reliability of our subsequent analyses. During this stage, we focus on filtering out noise, handling missing values, and normalizing the data to create a consistent and analyzable dataset. Once this foundational groundwork is completed, the prepared data is securely stored, forming a robust base for the training of our machine learning models.

In the next phase of our project : we employ this prepared dataset to train various sophisticated models. This training process is critical as it enables our models to learn and identify patterns indicative of engine health and potential failures.

After the models are trained : we undertake a comprehensive evaluation process to assess their performance. This involves applying a range of metrics, such as accuracy, precision, recall, and the ROC AUC, to determine the effectiveness of each model in predicting engine failures. The evaluation results guide us in selecting the best model that not only accurately predicts the Time-To-Failure (TTF) but also reliably classifies the likelihood and the potential time frame of engine failures.

Our systematic approach, from data preparation to model evaluation, ensures we choose the most

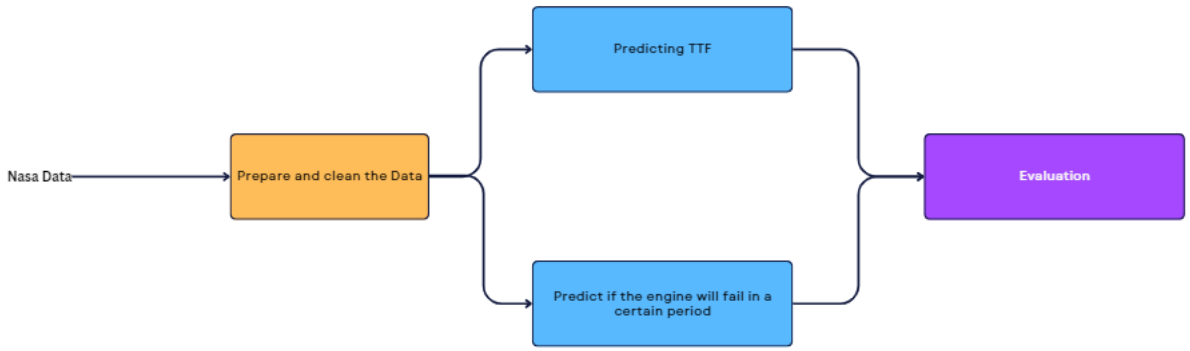

FIGURE 2 – The methodology

robust and accurate model for predictive maintenance, contributing significantly to enhancing airline safety and operational efficiency.

5 Development environment

We developed our models and codes within theses specifications :

- Google Colab running on Python 3.10
- Libraries Pandas and Numpy for Data preparation, cleaning and storage.
- Libraries MatplotLib and Seaborn for visualisation.
- Scikit-learn for developing machine learning algorithms and evaluation.

6 Data Preparation and Analysis

6.1 Data Preparation

The dataset provided for this AI project comprised three .txt files, each containing essential information for simulating aircraft engine run-to-failure events, operational settings, and sensor measurements. These files were made available by Nasa Datasets.

1. **Training Data File** : This file encompassed run-to-failure data for aircraft engines, comprising over 20,000 cycle records for 100 individual engines.
2. **Test Data File** : Unlike the training data, this file contained operational data for aircraft engines without recorded failure events. The remaining cycles for each engine were provided separately in a distinct Ground Truth Data File.
3. **Ground Truth Data File** : This file contained the true remaining cycle counts for each engine in the test dataset.

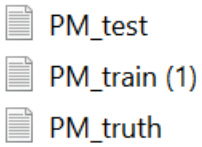

FIGURE 3 - The Data Files

To begin the data preparation process, we focused on the training data file. Initially, we imported the data into a CSV format. Upon examination, we observed that the file contained 28 columns, yet they were not labeled. Consequently, our first task was to assign appropriate names to these columns, adhering to the predetermined structure :

- **Column 1 :** *id* - Engine ID, ranging from 1 to 100.
- **Column 2 :** *cycle per engine sequence* - Denotes the cycle number for each engine's sequence, starting from 1 and extending to the cycle where a failure occurred.
- **Columns 3 to 5 :** *setting1* to *setting3* - Represent engine operational settings.
- **Columns 6 to 28 :** *s1* to *s23* - Consist of sensor measurements, totaling 23 distinct sensors.

This standardization of column names facilitated clarity and consistency throughout the subsequent data analysis and model development phases.

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s14	s15	s16	s17	s18	s19	s20	s21	s22	s23
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	8138.62	8.4195	0.03	392	2388.0	100.0	39.06	23.4190	NaN	NaN
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	8131.49	8.4318	0.03	392	2388.0	100.0	39.00	23.4236	NaN	NaN
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	8133.23	8.4178	0.03	390	2388.0	100.0	38.95	23.3442	NaN	NaN
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	8133.83	8.3682	0.03	392	2388.0	100.0	38.88	23.3739	NaN	NaN
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	8133.80	8.4294	0.03	393	2388.0	100.0	38.90	23.4044	NaN	NaN

FIGURE 4 - The training Dataset

Then we checked for NaN values in the dataset, as we noticed that the last two columns contain some NaN values. We found that our dataset is clean and does not contain NaN values, except for the last two rows, which consist solely of NaN values. We proceeded to drop these rows.

Using the *describe* method, we obtained some statistics of the dataset :

- There are 100 engines, each with between 1 and 362 cycles (with an average of 108 cycles per engine). The last cycle for each engine represents the cycle when a failure occurred.
- Some features, such as *setting3* and *s19*, are constant for all engines and cycles. We decided to drop them.

We tried to identify constant features that are irrelevant for prediction algorithms, after we confirmed that all data columns are numeric.

To identify constant columns, we checked for columns where the minimum value is equal to the maximum value, i.e., *min_values[column] == max_values[column]*. These columns were then dropped from the dataset as they do not provide valuable information for prediction.

Additionally, we observed that some columns exhibit minimal variation. However, we chose not to drop them as this slight variation might be crucial in determining engine failure.

```
Constant features where min = max:
['setting3', 's1', 's5', 's10', 's16', 's18', 's19']
```

FIGURE 5 - The dropped constant features

After removing NaN values and constant columns, we succeeded in reducing the dimensionality from **26 features to 19 features.**

For the test data file, it mirrored the structure of the training data file but with different engines, and the failure cycle was not provided, unlike in the training data. Utilizing the *describe* method, we found that there are 100 engines in the test data, with each engine having between 1 and 303 cycles (with an average of 76 cycles per engine).

Similar to the training data, we performed the same operations to reduce dimensionality in the test data. The failure events for the test data, represented by the remaining cycles before failure (TTF), were provided in a separate **truth file** .

This truth file was formatted as a CSV file, containing a column with TTF for each engine. Additionally, it included a NaN column, which we dropped during preprocessing.

	TTF
0	112
1	98
2	69
3	82
4	91

FIGURE 6 - The truth Dataset containing TTF values/engine

Upon observation, we noted that the sensor measurements exhibited low variance, which could pose challenges for predictive models relying solely on these features. To mitigate this issue, we augmented the dataset by incorporating additional features derived from the sensor readings : rolling average and rolling standard deviation.

We developed a function to facilitate the creation of these new features. This function takes two parameters :

- *df_in* (dataframe) : The input training dataframe to be processed.
- *rolling_win_size* (int) : The window size, representing the number of cycles for applying the rolling function.

The function returns a new dataframe with the rolling average and rolling standard deviation columns added for each sensor measurement (except for the constant ones that were previously dropped).

We opted for the inclusion of rolling average and rolling standard deviation columns to address the issue of low variance in sensor measurements. By applying a fixed rolling window size, we were able to smooth out fluctuations in the sensor readings, thereby enhancing the dataset's predictive capability. These additional features capture the trend and variability in sensor data over time, providing valuable insights for the predictive models.

	id	cycle	setting1	setting2	s2	s3	s4	s6	s7	s8	...	sd8	sd9	sd11	sd12	sd13	sd14	sd15	sd17	sd20	sd21
0	1	1	-0.0007	-0.0004	641.82	1589.70	1400.60	21.61	554.36	2388.06	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1	1	2	0.0019	-0.0003	642.15	1591.82	1403.14	21.61	553.75	2388.04	...	0.014142	1.499066	0.014142	0.438406	0.035355	5.041671	0.008697	0.000000	0.042426	0.003253
2	1	3	-0.0043	0.0003	642.35	1587.99	1404.20	21.61	554.26	2388.08	...	0.020000	4.632023	0.121655	0.404475	0.026458	3.717450	0.007640	1.154701	0.055076	0.044573
3	1	4	0.0007	0.0000	642.35	1582.79	1401.87	21.61	554.45	2388.11	...	0.029861	3.881555	0.171659	0.495950	0.029439	3.050906	0.028117	1.000000	0.076322	0.037977
4	1	5	-0.0019	-0.0002	642.37	1582.85	1406.22	21.61	554.00	2388.06	...	0.026458	4.587366	0.151063	0.432574	0.025884	2.651326	0.025953	1.095445	0.073621	0.033498

FIGURE 7 - The new Dataset with the new features added

Upon observation, we noticed that the target columns were not present in the dataset. To address this, we added regression and classification labels to the training data.

For regression labeling, we calculated the Time-To-Failure (TTF) for each engine cycle. This was achieved by subtracting each cycle's timestamp from the last cycle of the same engine.

For binary classification labeling, we introduced the Binary Classification Label (BNC). If the TTF is less than or equal to a specified period, it is labeled as 1 ; otherwise, it is labeled as 0.

To facilitate this labeling process, we developed a function that takes two parameters :

- *df_in* (dataframe) : The input training data.
- *period* (int) : The number of cycles for TTF segmentation.

The function returns the dataset with the target columns added.

	id	cycle	setting1	setting2	s2	s3	s4	s6	s7	s8	...	sd12	sd13	sd14	sd15	sd17	sd20	sd21	TTF	BNC	MCC
0	1	1	-0.0007	-0.0004	641.82	1589.70	1400.60	21.61	554.36	2388.06	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	191	0	0
1	1	2	0.0019	-0.0003	642.15	1591.82	1403.14	21.61	553.75	2388.04	...	0.438406	0.035355	5.041671	0.008697	0.000000	0.042426	0.003253	190	0	0
2	1	3	-0.0043	0.0003	642.35	1587.99	1404.20	21.61	554.26	2388.08	...	0.404475	0.026458	3.717450	0.007640	1.154701	0.055076	0.044573	189	0	0
3	1	4	0.0007	0.0000	642.35	1582.79	1401.87	21.61	554.45	2388.11	...	0.495950	0.029439	3.050906	0.028117	1.000000	0.076322	0.037977	188	0	0
4	1	5	-0.0019	-0.0002	642.37	1582.85	1406.22	21.61	554.00	2388.06	...	0.432574	0.025884	2.651326	0.025953	1.095445	0.073621	0.033498	187	0	0

FIGURE 8 - The new Dataset with the labeled targets

For the test data, we've done the same thing, but this time we also merged the Truth data to the test data.

Here we decided to end the preparation Data step, and save our latest datasets as CSV files (train.csv,test.csv).

These two files were used in the exploratory Data Analysis.

6.2 Data Analysis

Moving to the data analysis step, our first objective was to analyze the correlation between the features (settings and sensor measurements + features added) and the target variable, TTF (Time-To-Failure), which we aim to predict.

We began by selecting the features for correlation analysis, sorting them to identify potential relationships with TTF. This process revealed both positive and negative correlations. However, we recognized that selecting features based solely on their correlation values may not always be optimal. To gain deeper insights, we visualized scatter plots for each feature against TTF.

Through these analyses, we identified features ['av21', 'av20', 'av12', 'av7', 's12', 's7', 's21', 's20'] exhibiting higher correlation with TTF compared to other features. Consequently, these features were prioritized for selection during the modeling phase.

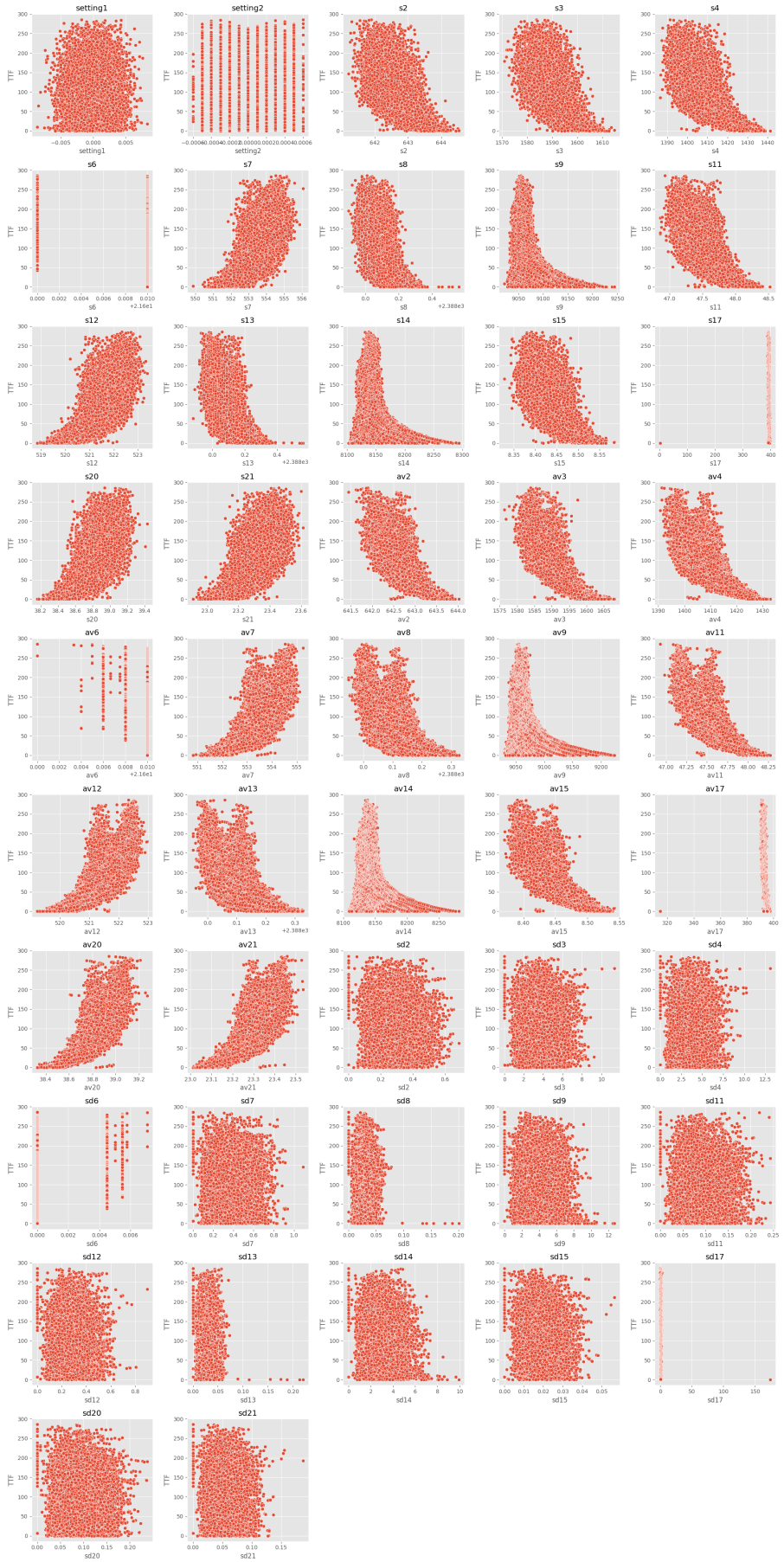

FIGURE 9 - The scatter plots of correlation with TTF

From the high correlated features with TTF, we visualized the correlation between each other using

a heatmap. The heatmap revealed a very high correlation (> 0.8) between some features.

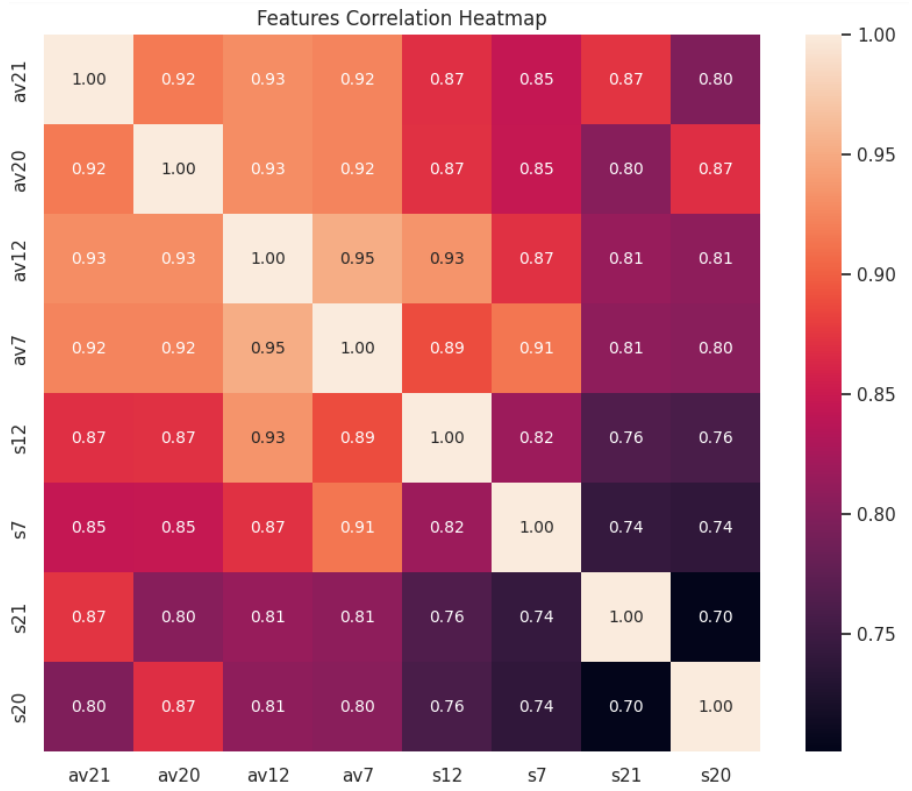

FIGURE 10 - Heatmap of correlation between the high correlated features with TTF

While high correlation can be beneficial in some cases, it may adversely affect the performance of certain machine learning algorithms by introducing multicollinearity.

To address this issue, we decided to target some of the highly correlated features for removal in feature selection. Specifically, we planned to remove the feature that has a low correlation with TTF from the two highly correlated features.

This approach aims to mitigate multicollinearity and improve the overall performance of our machine learning models.

During our data analysis, we focused on visualizing various aspects to gain insights into engine behavior and failure patterns.

As our goal was to predict the engine's Time-To-Failure (TTF), we visualized the distribution of engine lifespans.

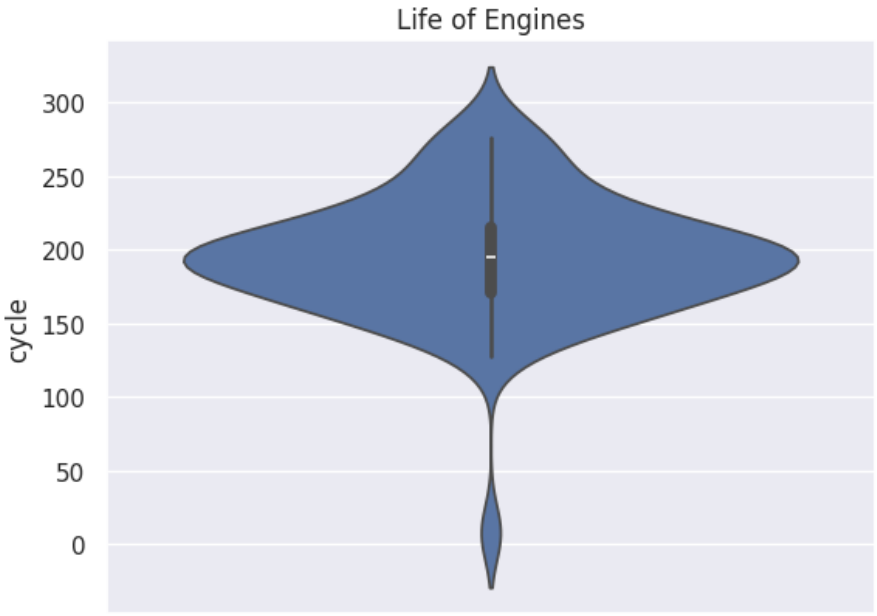

FIGURE 11 - Engines' life per cycle

The plot revealed that the average lifespan of an engine is around 200 cycles, with some engines exceeding 300 cycles.

Further analysis led us to hypothesize that there might be discernible signals in the last cycles of an engine's life indicating an imminent failure. To investigate this, we visualized the time series of sensor features for the last 50 cycles of the first 15 engines.

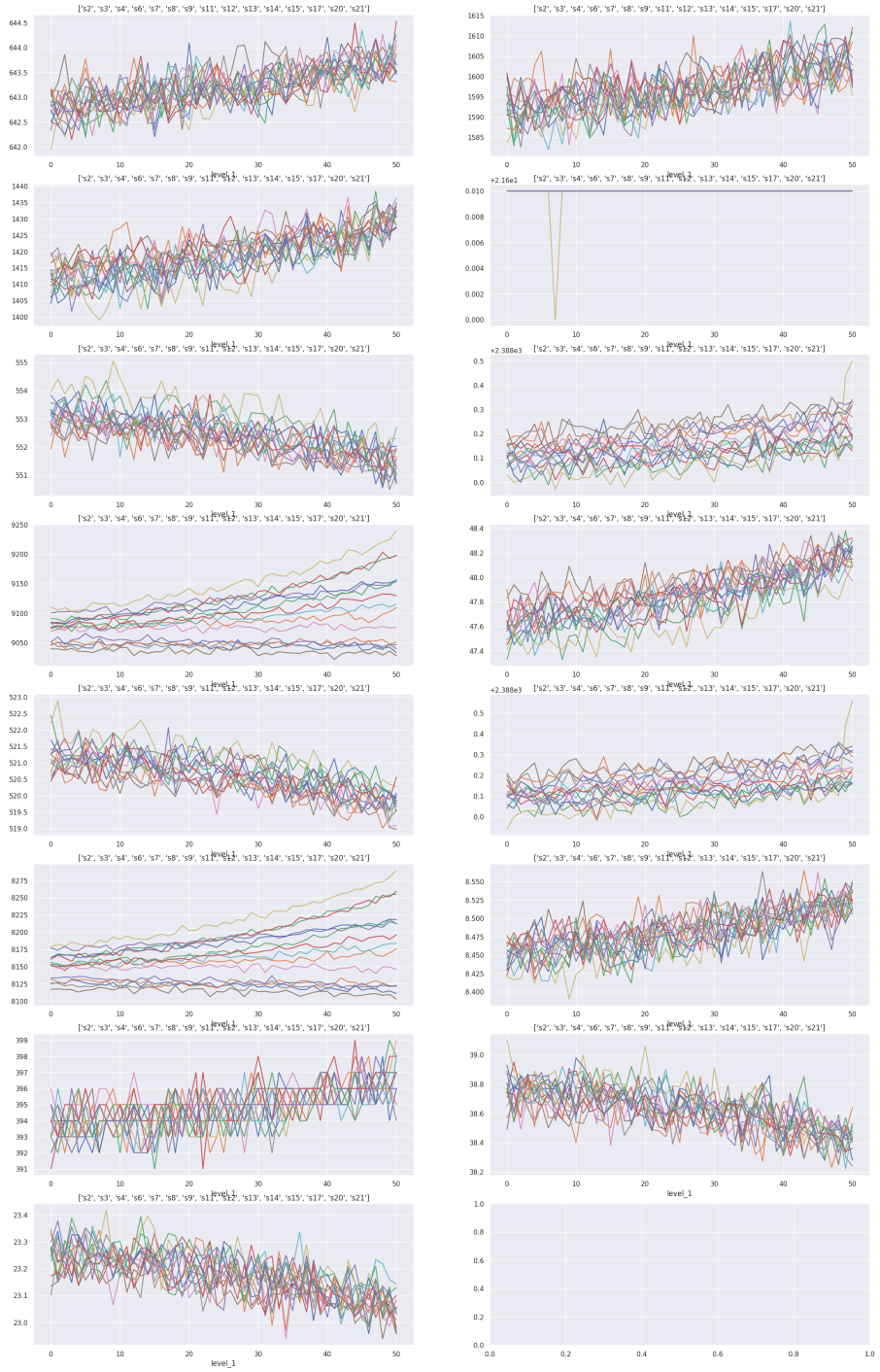

FIGURE 12 - Sensors measurements for last 50 cycles of the first 15 engines

The visualizations showed strong variations in sensor measurements for most engines, particularly in the last 50 cycles before failure. However, there were exceptions noted for the 7th and 11th engines, where sensor measurements tended to be larger. Additionally, a peculiar observation was made for the 4th engine, where all sensor measurements were consistently zero for the last 50 cycles.

These observations highlight the potential utility of sensor data in predicting engine failures and underscore the importance of further analysis and quantification of these signals using various techniques.

Upon assessing the data for both binary classification (BNC) , we observed significant class imbalance.

For the binary classification task :

— Negative samples (0) : 10385
— Positive samples (1) : 1930

This distribution indicates that negative samples comprise 84% of the dataset, while positive samples represent only 16%. Such an imbalance in the dataset warrants caution, as relying solely on classification accuracy as a performance metric may lead to misleading results. Instead, we opt to use the Area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC) curve as a more reliable performance metric.

7 Predicting Engine's Time-To-Failure (TTF)

In this section, we focus on predicting the Time-To-Failure (TTF) of engines. This is inherently a regression problem, not a classification one, as we aim to estimate a continuous variable (TTF) rather than assign discrete classes.

The models selected for this regression task are as follows :

— **Linear Regression** : A simple regression model that assumes a linear relationship between the features and the target variable.
— **LASSO (Least Absolute Shrinkage and Selection Operator)** : A regularized version of least squares regression, penalizing the L1 norm of the coefficients to encourage sparsity.
— **Ridge Regression** : Another regularized regression method that addresses multicollinearity by adding a bias term to the regression estimates.
— **Polynomial Regression** : A regression technique that models the relationship between the independent and dependent variables as an nth-degree polynomial.
— **Decision Trees** : These models learn in a hierarchical fashion by splitting the dataset into branches that maximize information gain. In regression trees, the terminal nodes represent the mean response of observations falling in that region.
— **Random Forests** : An ensemble learning method that constructs multiple decision trees and averages their predictions. It is effective for regression tasks by outputting the mean prediction of individual trees.

Which Features ?

Each model will be trained using three different sets of features :

1. Original features (after removing NaN features and constant ones).
2. Original features plus additional features (rolling mean and rolling standard deviation).
3. High-correlated features with TTF.

These approaches aim to capture different aspects of the data and provide insights into the predictive performance of the models.

Methodology ?

The methodology involves training all models on three different sets of features : original features, original features with additional features (rolling mean and rolling standard deviation), and high-correlated features with TTF. Subsequently, the performance of each model will be compared across these feature sets.

To facilitate training models on different feature sets, a variable named 'features' was created to hold the required set of features, which is then used to subset the original dataframes.

In order to streamline the analysis process, several functions were developed :

1. **Calculate Regression Metrics** : This function takes the model, actual, and predicted values as arguments and returns regression metrics such as Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), R-squared, and explained variance.
2. **Plot Feature Importance** : This function plots the feature importance for each model, providing insights into the contribution of individual features to the prediction.
3. **Plot Regression Residuals** : This function plots the regression residuals, enabling the examination of model performance in capturing the variance not explained by the features.

These functions play a crucial role in evaluating the performance of regression models and interpreting their results.

On which hyperparameters ?

Each regression model comes with its own set of hyperparameters that need to be tuned for optimal performance. Let's discuss the hyperparameters for some of the regression models used in our analysis :

- **Lasso Regression** : The key hyperparameter is alpha, which controls the strength of L1 regularization. The LassoCV class from scikit-learn library efficiently searches for the best alpha value via cross-validation.
- **Ridge Regression** : Similar to Lasso, Ridge regression also utilizes a regularization parameter alpha, but it controls the strength of L2 regularization. The RidgeCV class from scikit-learn library is used for hyperparameter tuning.
- **Polynomial Regression** : The degree of the polynomial is a hyperparameter that determines the flexibility of the model. We chose a degree of 2 for our analysis, transforming the data into polynomial features using PolynomialFeatures and then applying Linear regression.
- **Decision Trees** : The max depth parameter is a hyperparameter for decision trees, controlling the maximum depth of the tree. We fixed it to 10 for our analysis.
- **Random Forests** : Key hyperparameters include n estimators, max features, and max depth. We employed RandomizedSearchCV to efficiently search for the best hyperparameters.

Hyperparameter tuning is essential for optimizing the performance of regression models and is typically conducted via techniques such as cross-validation or randomized search across a specified range of values.

7.1 Training on Original Features - Results

The results on the test dataset are listed below :

	Linear Regression	LASSO	Ridge Regression	Decision Tree Regression	Polynomial Regression	Random Forest Regression
Root Mean Squared Error	32.041095	33.911243	31.966557	33.368578	31.461080	31.086142
Mean Absolute Error	25.591780	27.259315	25.554325	24.506921	23.999950	22.780593
R^2	0.405495	0.334071	0.408258	0.355214	0.426824	0.440405
Explained Variance	0.665297	0.637001	0.668870	0.554069	0.640990	0.651546

Table 1 – Comparison of Different Regression Models

After training and evaluating the regression models, we observed that non-linear models such as Polynomial Regression and Random Forest outperformed linear models such as Linear Regression, LASSO, and Ridge Regression. Specifically, Random Forest demonstrated superior performance, achieving an RMSE of 31.08 cycles. This indicates that the model predicts TTF within an average error range of ±31.08 cycles.

However, despite the promising performance of the models, regression residuals were not randomly spread across the average value of the residuals. This suggests that there is room for improvement, potentially by utilizing different types of features or refining the modeling approach.

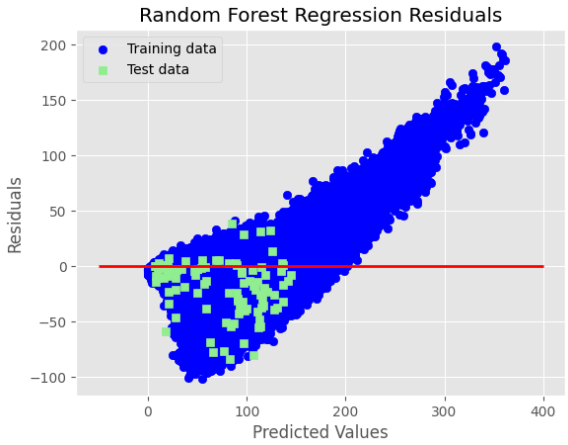

FIGURE 3 – Residuals - Random Forest

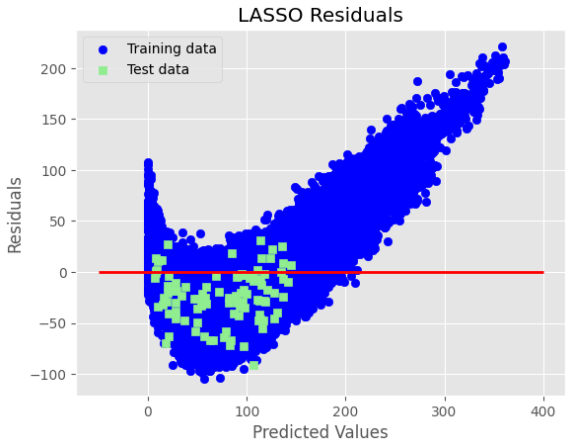

FIGURE 4 – Residuals - LASSO

Additionally, we found that for each model, there is a distinct set of important features. Nonetheless, it was consistently observed that sensor s11 and setting2 were deemed important across all models. This underscores the significance of these features in predicting engine TTF and highlights their relevance in the modeling process.

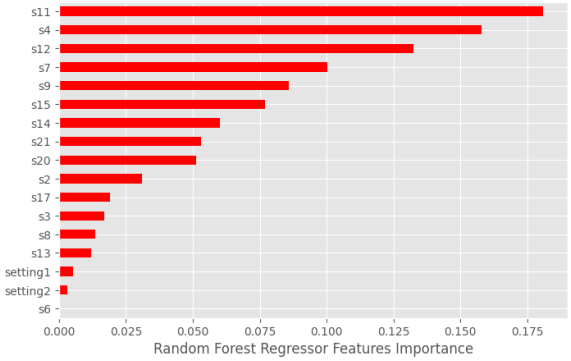

FIGURE 5 – Features importance - Random Forest

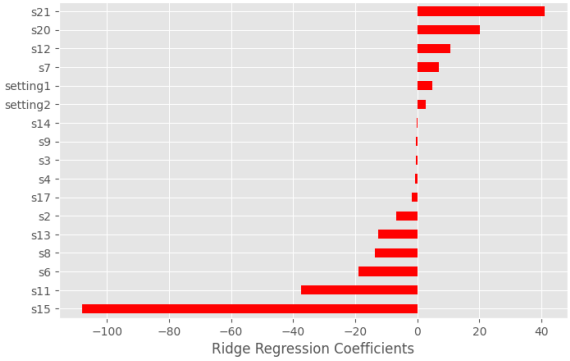

FIGURE 6 – Features importance - Ridge Regression

7.2 Training on Original Features + Added Features - Results

The results on the test dataset are listed below :

	Linear Regression	LASSO	Ridge Regression	Decision Tree Regression	Polynomial Regression	Random Forest Regression
Root Mean Squared Error	33.567598	34.093756	33.609873	40.050578	34.337596	31.428195
Mean Absolute Error	27.188280	27.796190	27.173600	26.948377	26.800907	23.089121
R^2	0.347499	0.326884	0.345855	0.071123	0.317221	0.428022
Explained Variance	0.630618	0.629381	0.631833	0.304820	0.538597	0.642285

TABLE 2 – Comparison of Different Regression Models

Upon training the models on the original features plus added features, several observations were made :

— **The inclusion of additional features did not lead to an improvement in model performance. In fact, the performance of the models was reduced compared to when trained on the original features alone.**
— Once again, non-linear models such as Random Forest outperformed linear models like Linear Regression, LASSO, and Ridge Regression. Random Forest exhibited superior performance, achieving an RMSE of 31.42 cycles. This indicates that the model predicts TTF within an average error range of ±31.42 cycles.
— Regression residuals were not randomly spread across the average value of the residuals, suggesting that there may be room for improvement in the modeling approach or feature selection.
— Interestingly, for every model, a different set of features emerged as important. However, it was consistently observed that the features added (such as rolling mean and rolling standard deviation) were among the most important features, even more so than the original features.

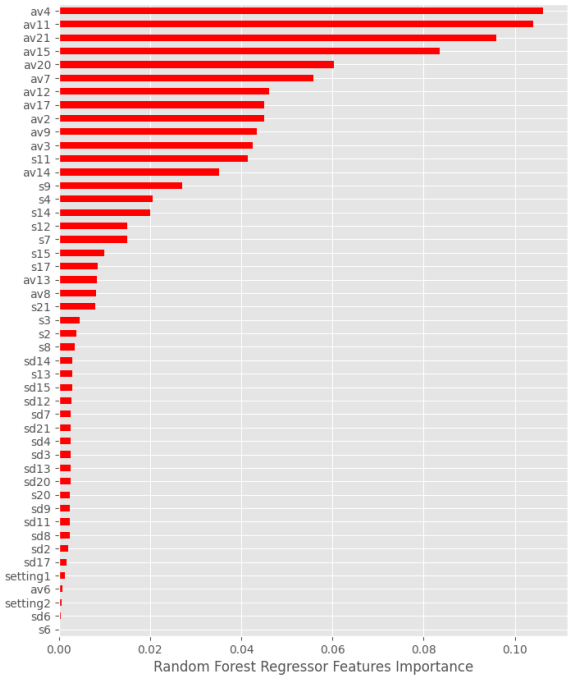

FIGURE 7 – Features importance - Random Forest

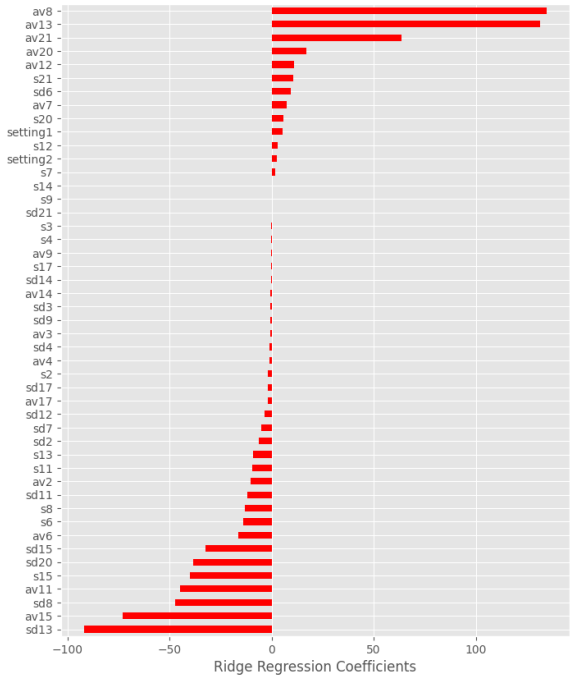

FIGURE 8 – Features importance - Ridge Regression

These findings underscore the importance of feature engineering and model selection in improving the predictive performance of regression models.

7.3 Training on High correlated features with TTF - Results

The results on the test dataset are listed below :

	Linear Regression	LASSO	Ridge Regression	Decision Tree Regression	Polynomial Regression	Random Forest Regression
Root Mean Squared Error	34.121522	34.210798	34.077596	37.356893	33.931705	34.178601
Mean Absolute Error	26.829372	26.847588	26.779848	27.379659	26.549590	25.883633
R^2	0.325787	0.322254	0.327522	0.191869	0.333267	0.323529
Explained Variance	0.546686	0.544830	0.547875	0.385668	0.547834	0.547968

TABLE 3 – Comparison of Different Regression Models

Upon training the models on high correlated features, several observations were made :

— Surprisingly, linear models performed better than Random Forest in this scenario. Ridge Regression emerged as the best model, achieving an RMSE of 34.077 cycles. This indicates that the model predicts TTF within an average error range of ±34.077 cycles.
— Once again, regression residuals were not randomly spread across the average value of the residuals, suggesting potential areas for improvement in the modeling approach or feature selection.
— For every model, features av21 and av20 consistently emerged as the most important features. This observation is not unexpected, as these features exhibit high correlation with TTF and are therefore crucial predictors.

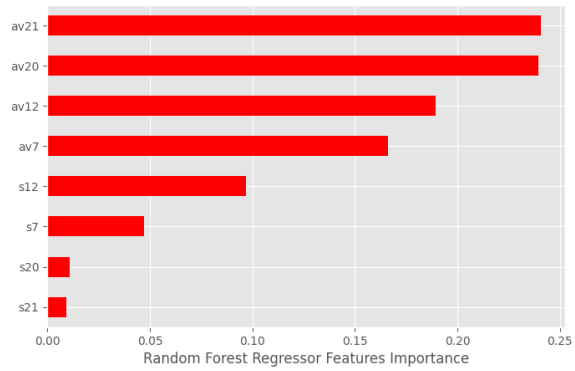

FIGURE 9 – Features importance - Random Forest

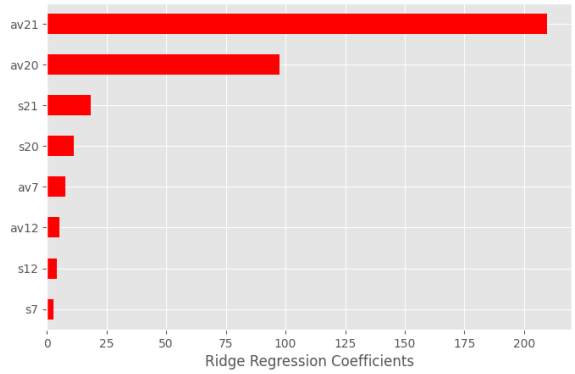

FIGURE 10 – Features importance - Ridge Regression

8 Predicting which engines will fail in the current period

Moving on to predicting which engines will fail in the current period, we employ multiple binary classification algorithms to forecast engine failures within a specific time frame, such as the remaining cycles or Time-To-Failure (TTF) in the range of 0-30 cycles. The evaluated algorithms include :

— **Logistic Regression** : A classification model that predicts probabilities using the logistic function, mapping predictions between 0 and 1.
— **Support Vector Machines (SVM)** : These algorithms utilize kernels to calculate distances between observations and find a decision boundary that maximizes the margin between classes.
— **K Nearest Neighbors (KNN)** : Instance-based algorithms that make predictions for new observations by searching for the most similar training observations and pooling their values.
— **Gaussian Naive Bayes** : An algorithm based on conditional probability and counting, where the model updates a probability table through the training data and predicts class probabilities for new observations.
— **Random Forest** : An ensemble learning method that constructs multiple decision trees and averages their predictions.

Similar to the regression models, each classification model is trained on both the original features and the original features plus added features. Evaluation metrics include Area Under the Curve (AUC), Receiver Operating Characteristic (ROC) curve, Recall, Precision, F1 Score, and Accuracy. However, more emphasis is placed on AUC and ROC due to the imbalanced nature of the data observed during Data Preparation.

On which hyperparameters ?

To optimize the performance of our classifiers, we perform hyperparameter tuning using Grid Search. This process involves systematically searching through a specified grid of hyperparameters to find the combination that maximizes the evaluation score.

For each classifier, we conduct Grid Search hyperparameter tuning followed by evaluation. This ensures that the best hyperparameters are selected for each model.

To streamline the hyperparameter tuning process across all classifiers, we created a function *tuning* that takes as arguments the model identifier, the classifier to be tuned, the Grid Search parameters, and the evaluation score. The function then returns the best parameters found through Grid Search and the corresponding predictions.

This approach allows us to efficiently tune hyperparameters and evaluate the performance of each classifier, ultimately leading to better predictive accuracy and model robustness.

How did we evaluate ?

After performing hyperparameter tuning, we evaluate the classifiers using various metrics and plots to assess their performance :

- **Confusion Matrix** : Provides a breakdown of true positive, false positive, true negative, and false negative predictions.
- **Classification Report** : Includes precision, recall, F1-score, and support for each class.
- **Metrics** :
  - **ROC AUC** : Area under the Receiver Operating Characteristic curve.
  - **Recall** : The proportion of true positive cases correctly identified.
  - **Accuracy** : The proportion of correctly classified cases.
  - **Precision** : The proportion of true positive cases among all positive predictions.
  - **F1 Score** : The harmonic mean of precision and recall.
- **Plots** :
  - **AUC ROC** : Receiver Operating Characteristic curve showing the trade-off between true positive rate and false positive rate.
  - **Precision-Recall Curve** : Illustrates the relationship between precision and recall for different thresholds.
  - **Precision-Recall Threshold Plot** : Shows precision and recall at different classification thresholds, along with the number of engines predicted for maintenance per period.
  - **TPR-FPR Threshold Plot** : Displays true positive rate (sensitivity) versus false positive rate (1 - specificity) at different classification thresholds.

These metrics and plots provide comprehensive insights into the performance of the classifiers and help in assessing their suitability for predicting engine failures.

8.1 Training on Original Features - Results

The results on the test dataset are listed below :

Based on the evaluation results :

- Gaussian Naïve Bayes, Random Forests, and Logistic Regression achieved the best AUC ROC scores.
- Although Support Vector Classifier (SVC) had the lowest AUC ROC, it demonstrated the best precision-recall performance, alongside Logistic Regression.

Model	ROC AUC	Recall	Accuracy	Precision	F1 Score
Logistic Regression	0.979200	0.68	0.91	0.944444	0.796098
Decision Tree	0.944533	0.56	0.88	0.933333	0.700000
Random Forest	0.977067	0.64	0.90	0.941176	0.761905
SVC	0.983467	0.68	0.91	0.944444	0.796098
KNN	0.935260	0.68	0.91	0.944444	0.796098
Gaussian Naive Bayes	0.987733	0.96	0.94	0.827586	0.888889

TABLE 4 – Classifier Performance Metrics

— Logistic Regression appears to be a powerful model in this scenario, as it performed well in terms of both AUC ROC and precision-recall.

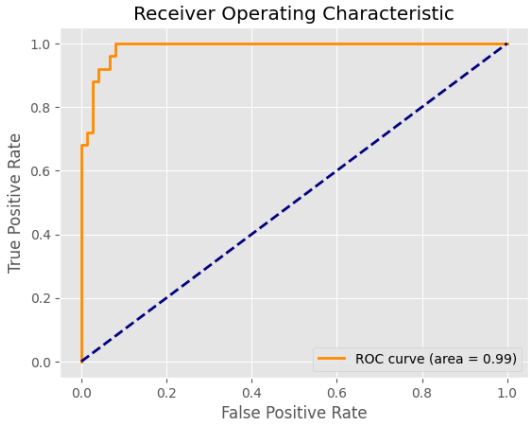

FIGURE 11 – ROC Curve - Gaussian Naive Bayes

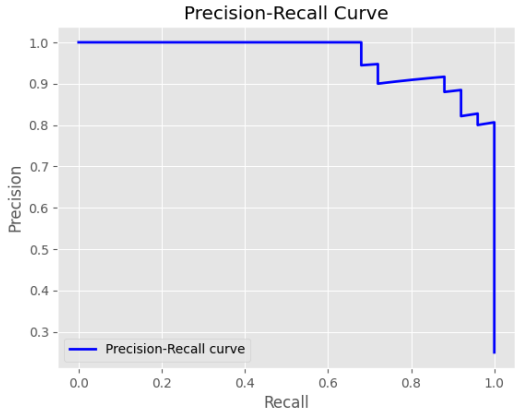

FIGURE 12 – Precision-Recall Curve - Gaussian Naive Bayes

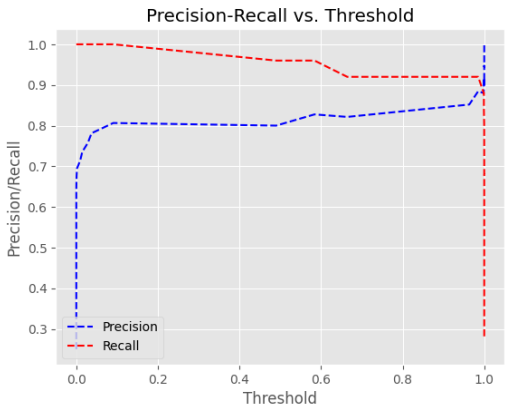

FIGURE 13 – Precision-Recall vs. Threshold - Gaussian Naive Bayes

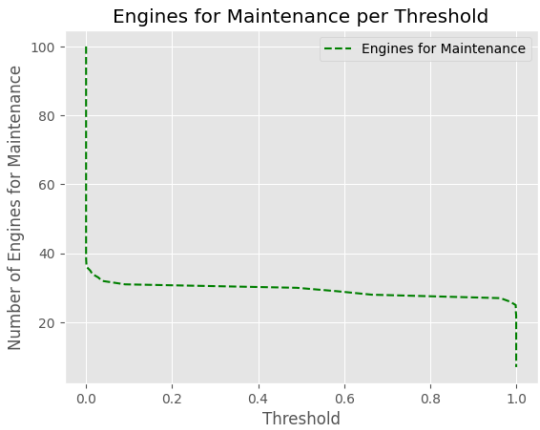

FIGURE 14 – Engines for Maintenance - Gaussian Naive Bayes

For the best model, Gaussian Naive Bayes, based on the True Positive Rate (TPR), False Positive Rate (FPR), and Threshold :

— **True Positive Rate (TPR)** : Also known as recall or sensitivity, measures the proportion of actual positives that are correctly identified. Initially, as the threshold increases from 0, the TPR rapidly reaches its peak, indicating a high sensitivity of the model for a range of thresholds.

— **False Positive Rate (FPR)** : Measures the proportion of actual negatives that are incorrectly identified as positives. The FPR starts high when the threshold is low and decreases as the threshold increases. The steep drop signifies that initially, many negatives are classified as positives, which then quickly corrects as the threshold increases.
— **Threshold** : The x-axis indicates the threshold level. As the threshold for classifying a positive increases, both TPR and FPR show significant changes up to a certain point. Beyond a threshold of around 0.5, changes become less pronounced for TPR and more for FPR.

Interpretation :

— For thresholds close to 0, the model predicts most instances as positive, resulting in a high TPR (many true positives) but also a high FPR (many false positives).
— As the threshold increases to around 0.25, TPR remains high, but FPR decreases, suggesting this may be a good operating point for maximizing true positives while reducing false positives.
— Beyond a threshold of around 0.5, TPR remains stable, indicating a consistently high recall rate.
— The continued decrease in FPR as the threshold increases beyond 0.5 is advantageous for reducing false positives but may also increase false negatives (not shown in the graph).
— A threshold setting above 1.0 leads to both TPR and FPR approaching zero, indicating that the model predicts most instances as negative, which would not be useful for a classifier aimed at detecting positives.
— Engines in the above charts represent the number of engines to be maintain per period, i.e. maintenance capacity.

8.2 Training on Original Features + Added Features - Results

The results on the test dataset are listed below :

Model	ROC AUC	Recall	Accuracy	Precision	F1 Score
Logistic Regression	0.981333	0.72	0.92	0.947368	0.818182
Decision Tree	0.962933	0.72	0.92	0.947368	0.818182
Random Forest	0.983467	0.64	0.90	0.941176	0.761905
SVC	0.926400	0.72	0.92	0.947368	0.818182
KNN	0.936467	0.72	0.92	0.947368	0.818182
Gaussian Naive Bayes	0.985533	0.96	0.94	0.827586	0.888889

Table 5 – Classifier Performance Metrics

After adding new features, the comparison table reveals the following observations :

— Most of the binary classifiers demonstrated improved performance metrics with the addition of new features.
— Random Forest, Logistic Regression, and Naive Bayes exhibited similar performance before and after feature engineering, yet still remained among the best-performing classifiers, with slight improvements observed.
— Linear SVC with new features achieved better scores compared to SVC without new features, indicating the effectiveness of feature engineering in enhancing performance.
— Linear SVC exhibited significant changes in scoring metrics before and after adding features, suggesting a notable impact of feature engineering on its performance.

— Although Gaussian Naive Bayes (GNB) achieved the best ROC AUC, it exhibited the lowest precision among the classifiers.
— Logistic Regression emerged as the top-performing classifier, boasting high ROC AUC and precision, making it the preferred choice for this classification task.

These findings highlight the importance of feature engineering in enhancing classifier performance and underscore the effectiveness of Logistic Regression for this particular classification problem.

9 Summary

In this AI project, we embarked on a journey to predict aircraft engine failures using machine learning techniques. Our dataset comprised simulated aircraft engine run-to-failure events, operational settings, and sensor measurements provided by Nasa Datasets. The dataset was divided into training, test, and ground truth files, containing information about engine cycles, operational settings, sensor measurements, and remaining cycles until failure.

We began by preparing the training data, which involved reading the data from a CSV file and assigning appropriate column names based on the provided information. We then cleaned the dataset by handling missing values and removing constant columns to reduce dimensionality. Additional features, such as rolling average and standard deviation of sensor readings, were introduced to smooth sensor measurements and potentially improve model performance.

After preparing the data, we explored various regression models to predict the remaining cycles until engine failure. Through hyperparameter tuning and evaluation, we observed that non-linear models, such as Polynomial Regression and Random Forest, outperformed linear models, with Random Forest exhibiting the best performance.

Transitioning to binary classification, we employed multiple algorithms to predict engine failures within a specific time frame. Hyperparameter tuning was performed for each classifier to optimize model performance. While Gaussian Naive Bayes achieved the best ROC AUC, Logistic Regression emerged as the preferred classifier due to its high precision and ROC AUC.

Furthermore, the addition of new features enhanced the performance of most binary classifiers, with Logistic Regression maintaining its status as the top-performing model. Linear SVC exhibited notable improvements in scoring metrics after feature engineering, underscoring the importance of feature selection and engineering in improving classifier performance.

In conclusion, this project demonstrated the efficacy of machine learning techniques in predicting aircraft engine failures. By leveraging regression and classification algorithms and optimizing model parameters through hyperparameter tuning, we achieved promising results in identifying potential engine failures and guiding maintenance efforts. Moving forward, continued refinement of feature engineering and model selection processes will be essential for further improving predictive accuracy and real-world applicability.