Statistical Learning Methods 21/22

Assignment

Statistical Learning for Predictive Maintenance

Claire DELGOVE – S359263

Number of words ≈ 3000 words (2500+20%)

Department: MSc AAI

Date: November 2021

Contents

1. **CONTEXT** **3**

2. **DATA PRE-ANALYSIS** **4**
   1. DATA FILES DISCOVERY 4
   2. TRAINING DATA DESCRIPTIVE ANALYSIS 6

3. **PREDICTION OF ENGINE'S TIME-TO-FAILURE** **10**
   1. DATA PREPARATION 10
   2. MODELS AND METRICS SELECTION 10
   3. PREDICTION RESULTS 11
   4. FINAL PREDICTION 14

4. **CLASSIFICATION OF FAILING ENGINES** **15**
   1. DATA PREPARATION 15
   2. MODELS AND METRICS SELECTION 16
   3. PREDICTION RESULTS 18
   4. FINAL PREDICTION 21

5. **CONCLUSION** **22**

6. **CONTENTS OF FIGURES** **23**

1 Context

In many industries, failure prediction is a consistent topic in predictive maintenance. Airlines are particularly interested in predicting equipment failures in advance to enhance the maintenance operations and planning of the aircraft engines. Indeed, by knowing which equipment needs maintenance, repair work can be better planned. Consequently, this reduces flight delays, losses due to the time engine has spent at the ground, the cost of time-based maintenance, and prevents unexpected equipment failures. Consulting engine's health and current condition through sensors and telemetry data is assumed to promote predictive maintenance by estimating the Time-To-Failure (TTF) or the Remaining Useful Life (RUL) of in-service equipment.

The project objective is to enhance the maintenance operations and planning of time-based preventive maintenance by applying data science techniques. By exploring the aircraft engine's historical data, the machine learning algorithm can learn the relationship between the sensor values to predict future failures and make data-driven decisions on its maintenance planning. Based on this analysis, the company will be able to estimate engine's time-to-failure and optimize its maintenance operations accordingly.

This report describes the work and the results about two main objectives. First, predicting the TTF (the number of remaining cycles before engine failure) for the engine using regression modelling algorithms, secondly, classifying which engine will fail in the analysed time-period by binary classification. All the codes will be performed in Python 3, using Jupyter Notebook, and different useful environments.

2 Data Pre-Analysis

2.1 Data files discovery

Training Dataset

The training dataset (*Figure 1*) contains 4 simulated sensors measurements for 100 aircraft engines running up to a failure. It consists in 20631 pieces of data. For each aircraft, the cycle starts at one until the cycle number where failure happens. The values of the expected Time-To-Failure are available along with the classification label that is set to 1 when *TTF* is within the last 30 cycles of engine operation, 0 otherwise.

	id	cycle	s1	s2	s3	s4	ttf	label_bnc
0	1	1	1400.60	554.36	47.47	521.66	191	0
1	1	2	1403.14	553.75	47.49	522.28	190	0
2	1	3	1404.20	554.26	47.27	522.42	189	0
3	1	4	1401.87	554.45	47.13	522.86	188	0
4	1	5	1406.22	554.00	47.28	522.19	187	0

Figure 1: Training dataset extract

The engine progressing degradation pattern is reflected in its sensor measurements. Indeed, the cycle feature is essential because it contains the old-age information, informing if the engine is more likely to fail. An example of the data visualisation for the first aircraft obtained by MATLAB is shown on *Figure 2*.

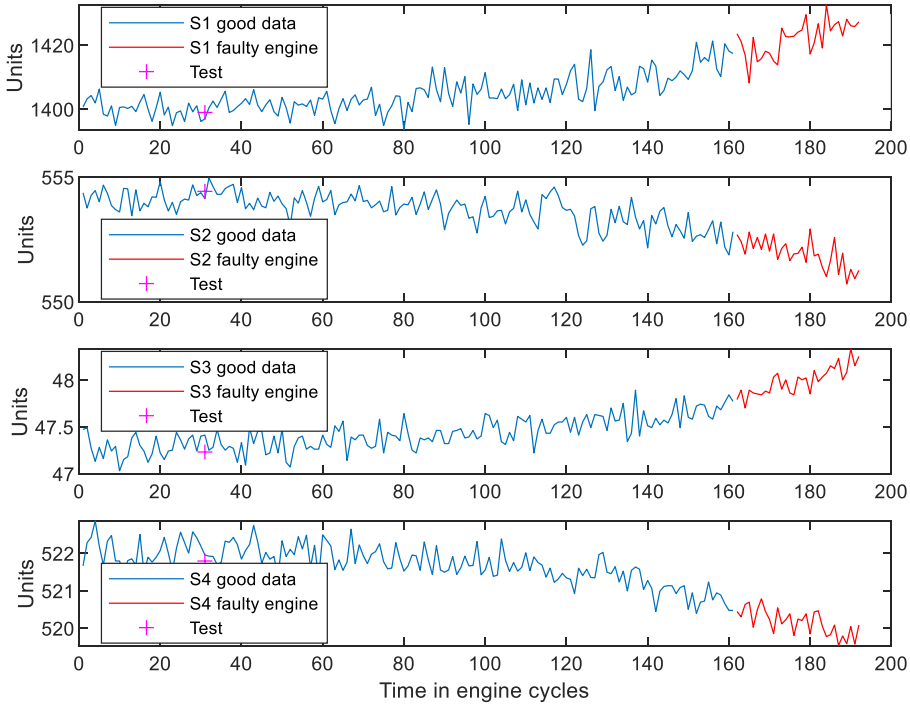


Figure 2: Training and test data for engine 1

Testing Dataset

The testing dataset (*Figure 3*) contains measurements from 4 sensors performed at a randomly selected cycle of engine operation. It consists in 100 pieces of data. True values for the *TTF* prediction are available separately for quantification of the prediction and classification results.

	id	cycle	s1	s2	s3	s4
0	1	31	1398.91	554.42	47.23	521.79
1	2	49	1410.83	553.52	47.67	521.74
2	3	126	1418.89	552.59	47.88	520.83
3	4	106	1406.88	552.64	47.65	521.88
4	5	98	1419.36	553.29	47.46	521.00

Figure 3: Testing dataset extract

2.2 Training data descriptive analysis

Correlation between features

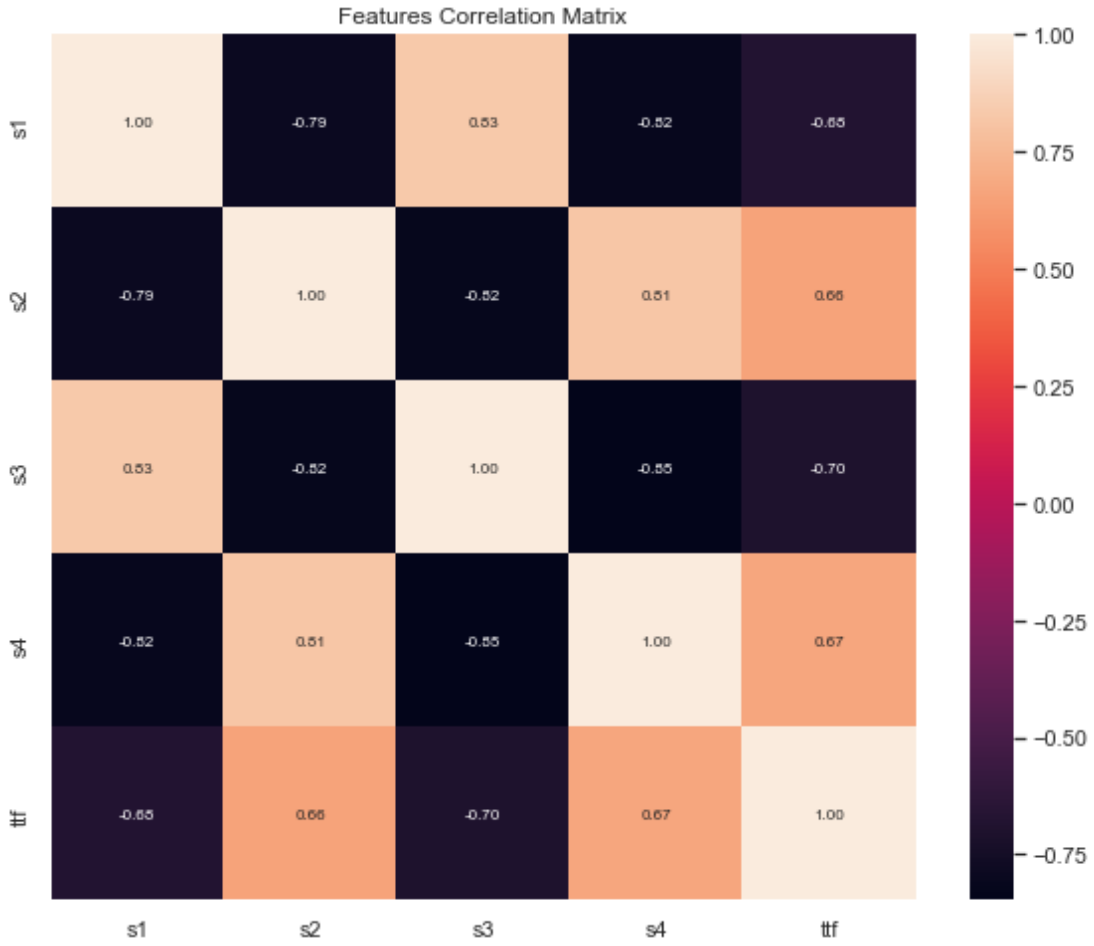


Figure 4: Correlation matrix between features

By observing the *Figure 4*, it is concluded that sensors are well correlated with the *TTF* values (correlation > 0.5 for each sensor).
It appears that sensors are also quite correlated with each other. Since the training dataset contains only four sensor features, it is wiser to keep the more information as possible with all the initial data, and not to perform features reduction. Indeed, it isn't necessary to gain in visibility in this dataset.

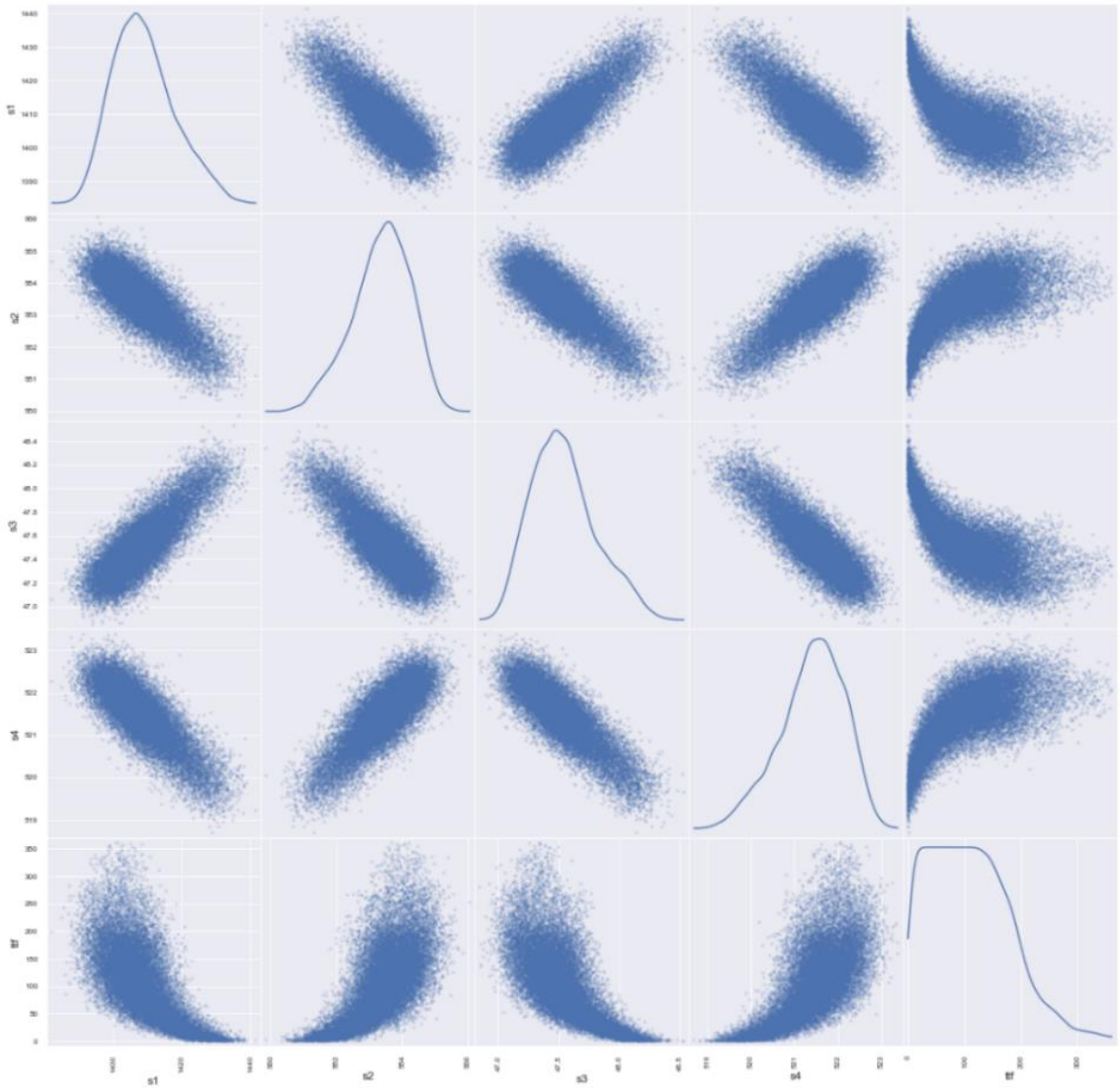


Figure 5: Scatter matrix to display relationships and distribution among features

Previewing the distribution of the sensor data according to the *TTF* (*Figure 5*), it is visible that most of the features have a non-linear relationship with the *TTF*. Consequently, it seems that using a non-linear model such as a polynomial one may lead to better results than using a standard linear regression.

Analysis for each feature

For each sensor, the histogram, the boxplot, the evolution over the cycles for ten random aircrafts, and a scatter plot drawing sensor values over *TTF* are illustrated *Figure 6 to 9* to show more insights on each feature individually.

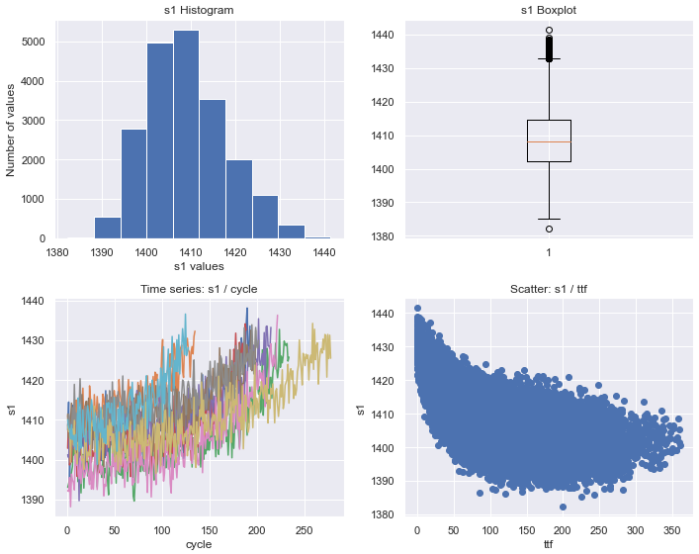


Figure 6: Charts for sensor 1

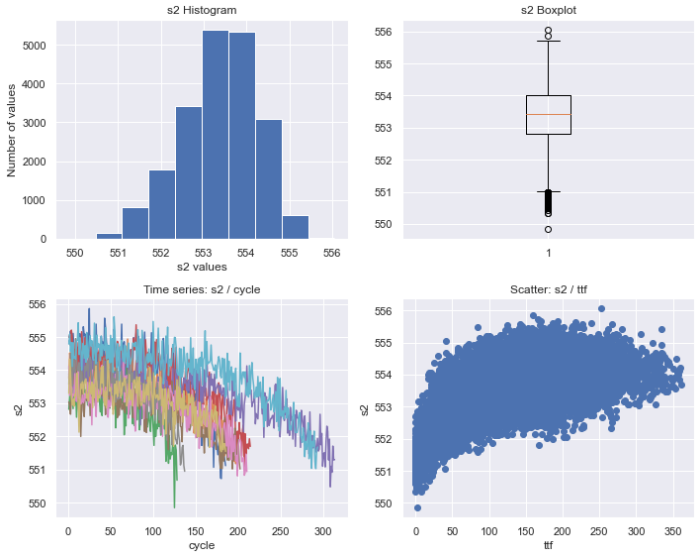


Figure 7: Charts for sensor 2

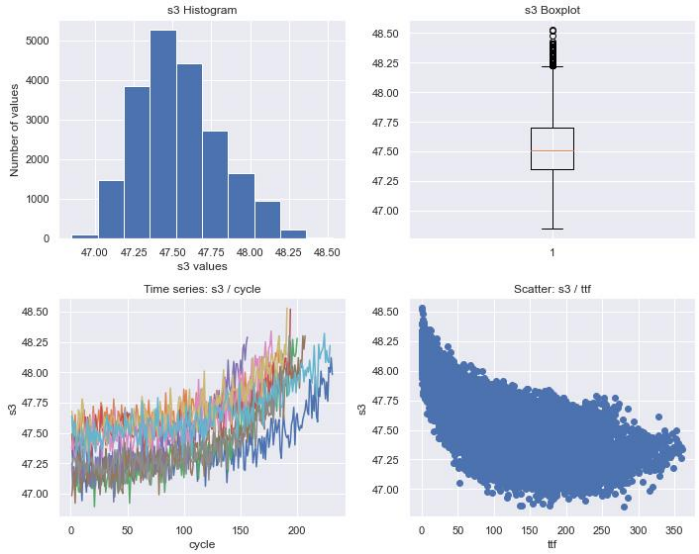


Figure 8: Charts for sensor 3

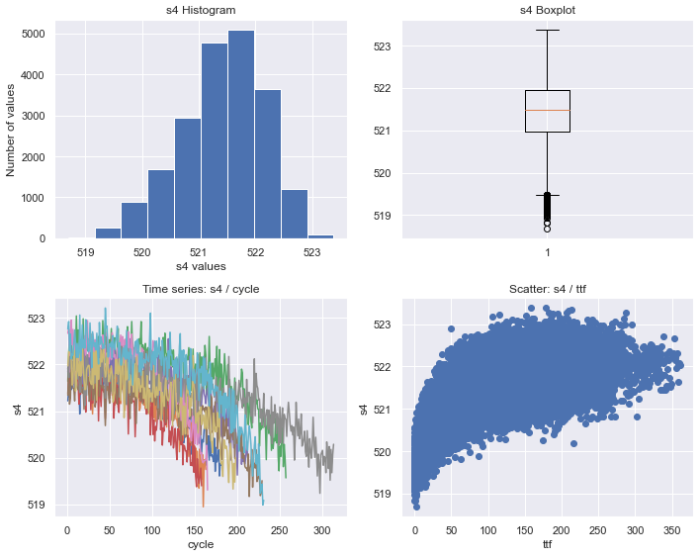


Figure 9: Charts for sensor 4

These charts point out how the different relationships between the sensors and the *TTF* have the same trend, particularly for the duo *s1 / s3*, and *s2 / s3*.
On the scatter plot, it is observed that the *TTF* of all the engines have the same decreasing or increasing trend along time, indicating the degradation of the engines. Trends seem to be polynomial non-linear, but apparently there is some noise visible by overlapping the curves due to the variability of the different engines.
Most importantly, the values of the four sensors have not the same order of magnitude (observing the histograms and the boxplots). However, the sensor values are comparable in their raw state. Consequently, in order not to lose information, a normalization won't be performed before applying the different regressions.

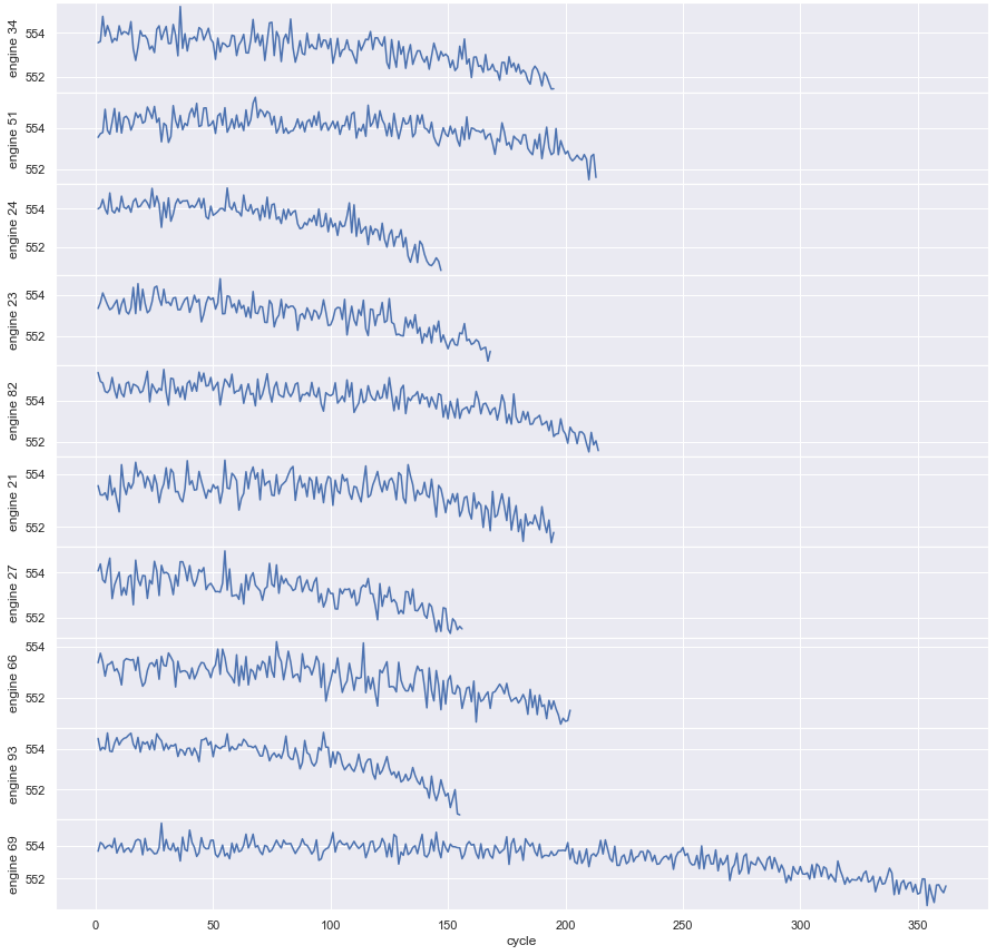


Figure 10: Time series for sensor 2 selecting random sample engines

Regarding the times series evolution for a specific sensor on different engines (*Figure 10*), it seems that the trend is found by fitting a linear model to each sensor.

3 Prediction of Engine's Time-To-Failure

The objective of this part is to predict *TTF* (the number remaining cycles before engine failure) for the engine using regression modelling algorithms.

3.1 Data preparation

Feature scaling and pre-processing

As the data don't have outliers or missing values, it isn't necessary to perform an important pre-processing. As mentioned previously, a normalization and a Principal Component Analysis aren't required.

3.2 Models and metrics selection

Background to model selection

The previous analysis gave a more accurate insight about the different features from the dataset. Even if it seems that polynomial models may lead to better results due to the non-linear relationships with the *TTF*, it makes sense to test different important algorithms to select the most suitable one for the prediction required. The previous trends didn't seem to be linear, then the Linear Regression algorithm can be directly rejected.
Therefore, the following machine learning regression algorithms were tried: *LASSO Regression*, *Ridge Regression, Polynomial Regression,* and *Random Forests Regression*.

Performance metrics selected

With the aim of selecting the best model for the maintenance predictions, the following main regression metrics are used to assess predictions. They are briefly introduced to explain their importance and suitability in the model evaluation.

- ***Root Mean Squared Error*** (RMSE):

The ***RMSE*** is the standard deviation of the residuals (prediction errors). A perfect mean squared error value is 0, which means that all predictions matched the expected values exactly. RMSE punishes larger errors more than smaller errors.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

- ***Mean Absolute Error*** (MAE):

The ***MAE*** is calculated as the average of the absolute error values. A perfect mean absolute error value is 0, which means that all predictions matched the expected values exactly. The MAE doesn't give more or less weight to different types of errors and instead the scores increase linearly with increases in error.

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n} |y_j - \hat{y}_j|$$

- ***R-squared*** (R^2: coefficient of determination):

The ***R^2*** score explains the dispersion of errors of a given dataset. It is used to measure the discrepancy between a model and actual data. The scores close to 1 are highly desired, indicating better squares of standard deviations of errors. A model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.

$$R^2 = 1 - \frac{Var(y - \hat{y})}{Var(y)}$$

3.3 Prediction results

Metrics results analysis

	LASSO Regression	Ridge Regression	Polynomial Regression	Random Forest Regression
Root Mean Squared Error	32.361810	32.360923	28.110016	26.665521
Mean Absolute Error	26.997974	26.997983	22.920712	21.350000
R^2	0.540460	0.540492	0.602923	0.662056

Figure 11: Metrics results for each algorithm

In accordance with our analysis, non-linear regression models like *Polynomial* and *Random Forest* performed better than other models like *Lasso* and *Ridge regression*. *Random Forest* clearly outperformed scoring *RMSE* of 26.89 cycles. It is a measure of how spread out these residuals (prediction errors). In other words, it tells how concentrated the data is around the line of best fit. Here, the model predicts *TTF* within an average error range of ±26.89 cycles.
In this case, the *RMSE* metric is more important than the *MAE* because it penalizes large errors more, and large errors are particularly undesirable for *TTF* predictions. Indeed, it isn't desired that a *TTF* prediction would be totally false because big troubles in the planning aren't expected.
As both *Polynomial* and *Random Forest* predict are more efficient, it is decided to select them for the future parts.

Residual's analysis

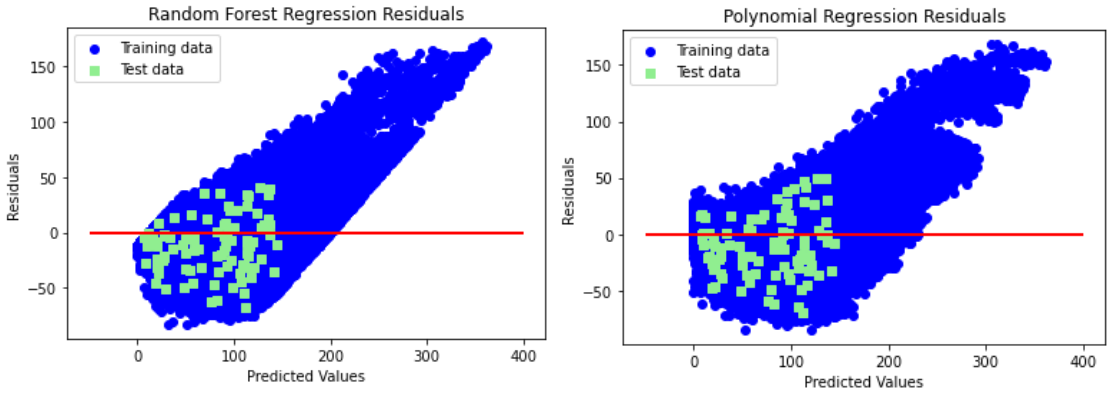


Figure 12: Residuals for the best algorithms

The previous R^2 results reflect the dataset's dispersion of errors. The better score is 0.66 for *Random Forest* but we could expect better. Another way to visualize the discrepancy is to plot the residuals (prediction errors) graphs (*Figure 12*) to check if the regression is trustful, and to measure how far from the regression the data points are. Both residuals' plots are centred to zero. This pattern indicates the models' predictions are correct on average. However, the residual's plot is a slightly better for *Polynomial Regression* rather than *Random Forest*.

Results are quite relevant, although they could be even better. Then, a hyperparameter tuning for better processing is done to select the appropriate degree for the *Polynomial Regression* and best parameters for *Random Forest.*

Polynomial regression tuning

The actual degree for *Polynomial Regression* is 2, but maybe another one would be more suitable.

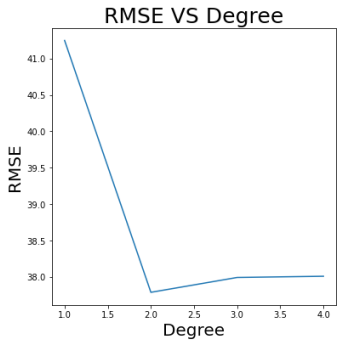


Figure 13: RMSE VS Degree

Using *Grid_Search* for tuning, the algorithm returns that the best degree for *Polynomial Regression* stays 2 (*Figure 13*). Indeed, proceeding by cross validation (*cross_validate* function) with 5 folds, the *RMSE* is lower for the degree 2.

Random forest tuning

INITIAL PARAMETERS

```
RandomForestRegressor(
                      n_estimators = 500,
                      max_features = 3,
                      max_depth = 4,
                      min_samples_split = 2,
                      n_jobs = -1,
                      random_state = 1)
```

FINAL PARAMETERS

```
RandomForestRegressor(
                      n_estimators = 200,
                      max_features = 3,
                      max_depth = 4,
                      min_samples_split = 2,
                      n_jobs = -1,
                      random_state = 1)
```

Figure 14: Random Forest's parameters

In the same way for *Random Forest*, using *Grid_Search*, the best hyperparameters are found (*Figure 14*). The first try was quite near to the optimal setting.

Model cross validation

The effectiveness of the model is checked by cross validation. The score used, to plot the learning curves *Figure 15* and *Figure 16,* is a wrap of multiple regression score such as R^2.

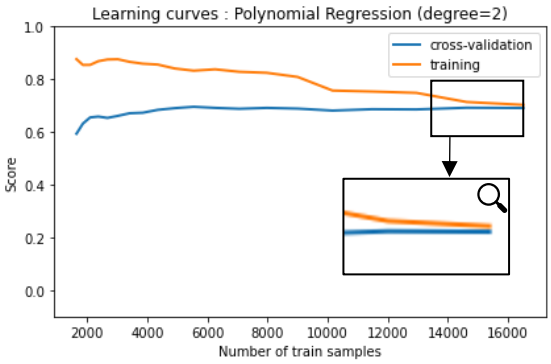


Figure 15: Learning curves for Polynomial Regression

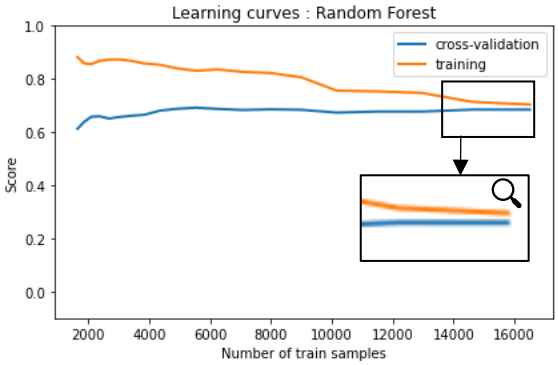


Figure 16: Learning curves for Random Forest

Obviously, the training score is higher than the validation score because the model is more efficient to fit the data it has already seen.
The convergence to a score near to 0.7 is noticeable as the number of train samples grows. Curves become more and more closer but never cross. That means models are efficient. Indeed, the graph doesn't show one of the following situations (*Figure 17*), illustrating overfitting or underfitting cases.

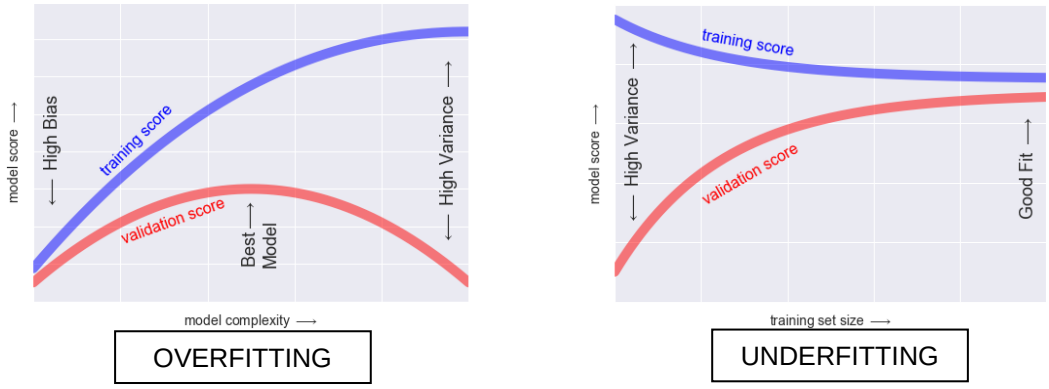


Figure 17: Overfitting and underfitting examples for explanation

Since both models are efficient and similarly performant, *Random Forest* will be selected for final predictions because of its higher metrics scores.

3.4 Final prediction

```
R^2 training: 0.699, R^2 test: 0.581
```

	Random Forest Regression
Root Mean Squared Error	26.665521
Mean Absolute Error	21.350000
R^2	0.662056

Figure 18: Random forest's scores

As the metrics reveal (*Figure 18*), the model *Random Forest* isn't perfect. Even by applying tuning, the R^2 can't reach 1, and the *RMSE* 0.
Since predicting *TTF* is critical to all kinds of modelling performed in this project, more work is required to enhance regression performance. This could be by trying and tuning stronger models (such as Neural Networks). Maybe the scores aren't perfect partly due the noise in the sensors data observed (*Figure 6 to 10)*. Then, a solution would be to smooth the data, using regression for the trends of the sensors (*Figure 10*), but this can affect the training data and doesn't describe the test data well anymore.

4 Classification of failing engines

From now on, the objective is to classify which engines will fail in the analysed time-period (*TTF* in the range of 0-30 cycles) by binary classification. It will be able to predict whether an engine is considered as faulty within its current operation cycle.

4.1 Data preparation

Data pre-visualization

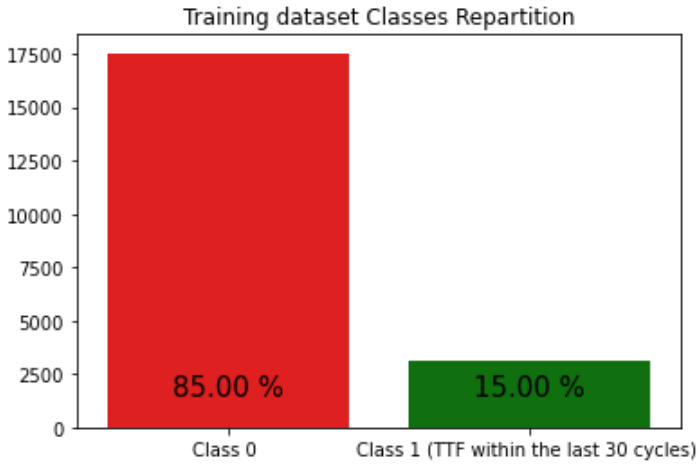


Figure 19: Training data repartition

The histogram (*Figure 19*) points out the dataset is unbalanced. Indeed, it contains more training data labelled as Class 1 (*TTF* within the last 30 cycles of engine operation), than Class 0. This repartition is due to the time-period set at 30 cycles.
This repartition could affect the future classifier which could be able to recognize Class 0 more easily than Class 0. This means it would avoid sending engines in maintenance if it isn't needed, in order to not lose money leaving an engine on the ground. The more realistic approach would have been to have more Class 1 in the training dataset to not miss engines in need of maintenance. But this choice depends on the airline's risk appetite. Even if this more realistic approach isn't reflected in the dataset repartition, it can be taken into consideration later fixing a specific threshold.

Feature scaling and pre-processing

As the data don't have outliers or missing values, it isn't necessary to perform an important pre-processing. As mentioned previously, a normalization and a Principal Component Analysis aren't required. Then, a features extraction won't be performed.

4.2 Models and metrics selection

Background to model selection

To classify the engines which will fail in the given period, the following machine learning classification algorithms were tried, because there are normally adequate for binary classification and standard datasets: ***Logistic Regression***, ***Random Forest Classifier***, ***SVC Linear, KNN,*** and ***Gaussian NB***.

Performance metrics selected

Evaluation metrics must be selected carefully. Indeed, it would be irrelevant to choose the *Accuracy* metrics. Because the dataset is unbalanced, *Accuracy* could be high (for instance, high capacity to predict Class 0), but not representative of the real capacity of the classifier, which could have more difficulty to predict Class 1. Instead, the *Precision* metrics will be prevailed to know the proportion of Class 1 well predicted by the classifier.

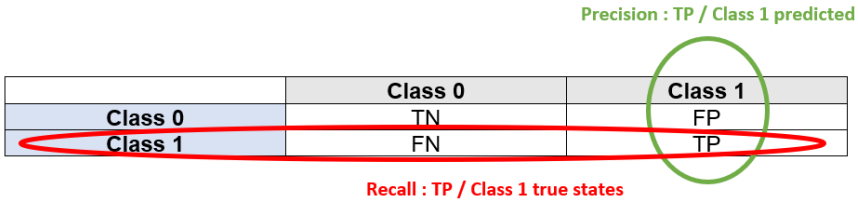
Before explaining the metrics, let's remind the signification of different abbreviations:

TP: *True Positive* – Faulty engine considered as faulty
TN: *True Negative* – Proper engine considered as proper
FP: *False Positive* – Proper engine considered as faulty
FN: *False Negative* – Faulty engine considered as proper

With the aim of selecting the best classification model, the following main classification metrics are used to assess predictions. They are briefly introduced to explain their importance and suitability in the model evaluation.

- ***Precision***:

Precision metric tells us how many predicted samples are relevant in the data predicted as positive. It is a measure of a classifier exactness.



$$Precision = \frac{TP}{TP + FP}$$

Figure 20: Precision / Recall comparison

- ***Recall***:

Recall metric shows how many relevant samples are selected. It is a measure of the classifier completeness.

$$Recall = \frac{TP}{TP + FN}$$

- ***F1 Score***:

F1 metric is the weighted average of the precision and recall.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

- ***ROC / AUC***:

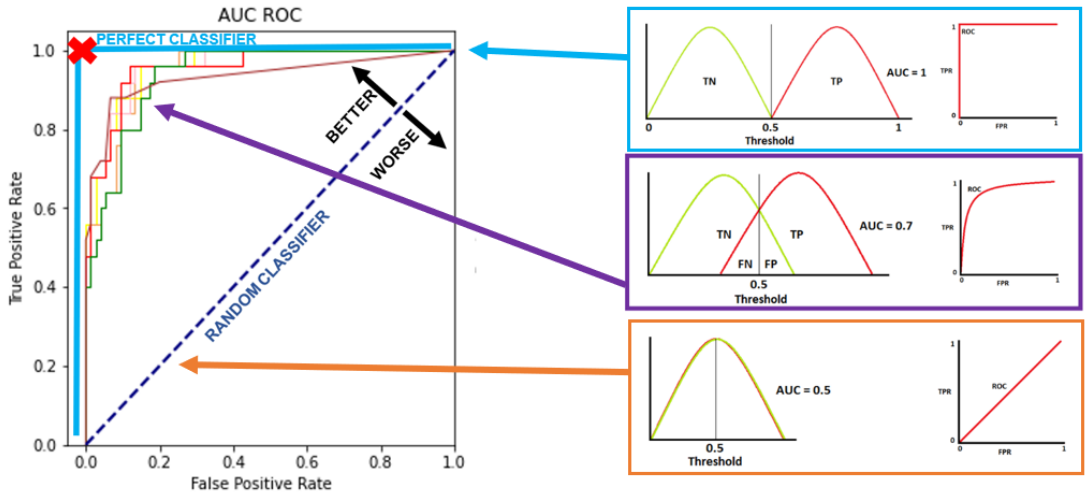


Figure 21: AUC / ROC curve explanation

The ***Area Under the Curve*** (***AUC***) is the area which measures the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between both classes**.**

4.3 Prediction results

Classifier tuning

Before using a specific classification algorithm, tuning the classifier is the best way to improve performance, choosing the best hyper-parameters. The score used for the *GridSearch* hyper-parameters tuning is *AUC/ROC*.

Metrics results analysis

	Logistic Regression	Random Forest	SVC Linear	KNN	Gaussian NB
Precision	0.894737	0.894737	0.714286	0.857143	0.750000
Recall	0.680000	0.680000	0.800000	0.480000	0.720000
F1 Score	0.772727	0.772727	0.754717	0.615385	0.734694
ROC / AUC	0.960000	0.964267	0.952000	0.929067	0.958933

Figure 22: Metrics results for each algorithm

Regarding the metrics' results, the best algorithms are *Logistic Regression* and *Random Forest*.
Models are around 90% precise predicting quite correctly the labels. The score doesn't reach 100% due to a few engines considered as faulty whereas they aren't (*False Positive*). This makes losing money to the Airline leaving aircrafts at the ground while they could work.
However, it isn't the worse error made by both models. Indeed, they have more difficulty to select the relevant samples for each class. Their *Recall* is around 0.68 which means some engines are considered as proper while they are faulty (*False Negative*). Sensitively, it is the main error to minimize because sending a faulty engine in a flight could be a disaster for passengers and the crew.
Since the F1 Score provides an average between the *Recall* and the *Precision*, it is stagnating due to the low rate of *Recall* compared to the *Precision*.

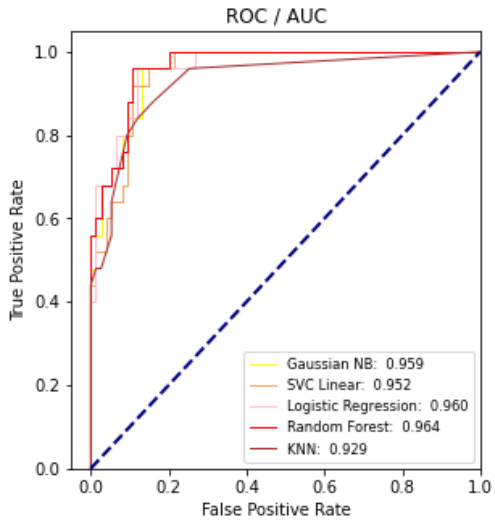


Figure 23: AUC/ROC curves for all models

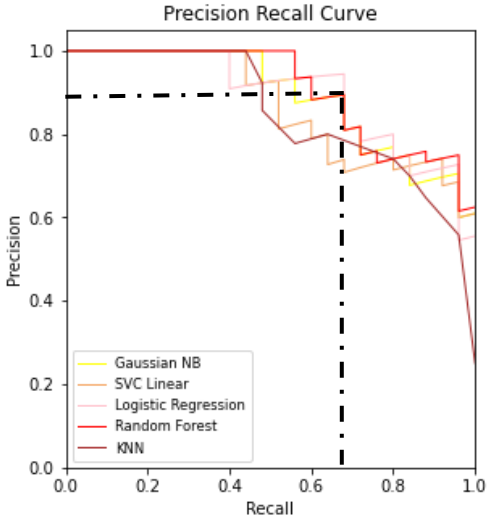


Figure 24: Precision Recall curves for all models

Regarding the ROC/AUC curves (*Figure 23*), the curves near to the optimal solution (AUC=1) show that models can distinguish in general quite well both classes. They have a good measure of separability.
On the precision recall curves (*Figure 24*), the high area under the curve represents both recall and precision returned by the different classifiers. The black lines show the best score reached by *Random Forest.*

Partly due to the tuning, results are quite similar and as good as each other. However, to define a final prediction, *Logistic Regression* and *Random Forest* (very similar to *Logistic Regression* regarding the results) models are selected to continue the analysis.

Model cross validation

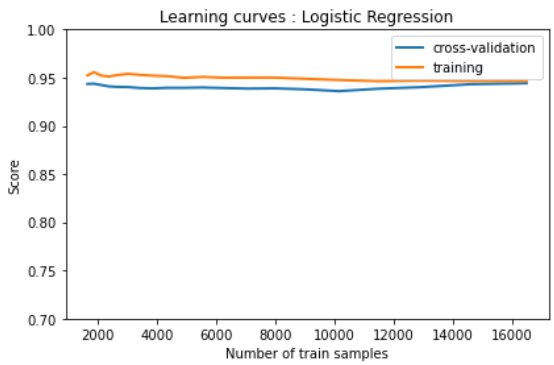


Figure 25: Learning curves for Logistic Regression

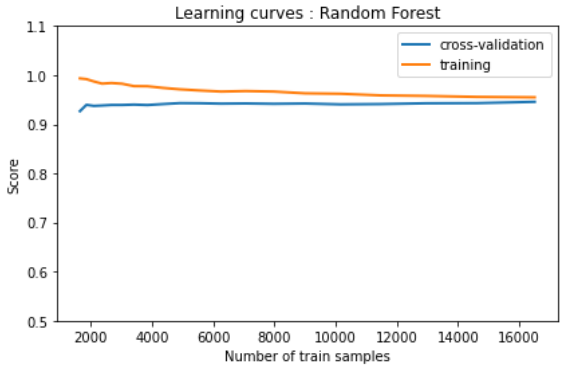


Figure 26: Learning curves for Random Forest

The learning curves are plotted *Figure 25* and *Figure 26*. The models predict the training data very well, such as the unknown data. It is noticeable the convergence to a score near to 0.95 as the number of train samples grows. Curves become more and more closer but never cross.

Since both models are similarly efficient, *Logistic Regression* model will be selected for final predictions because of its learning curves a bit closer.

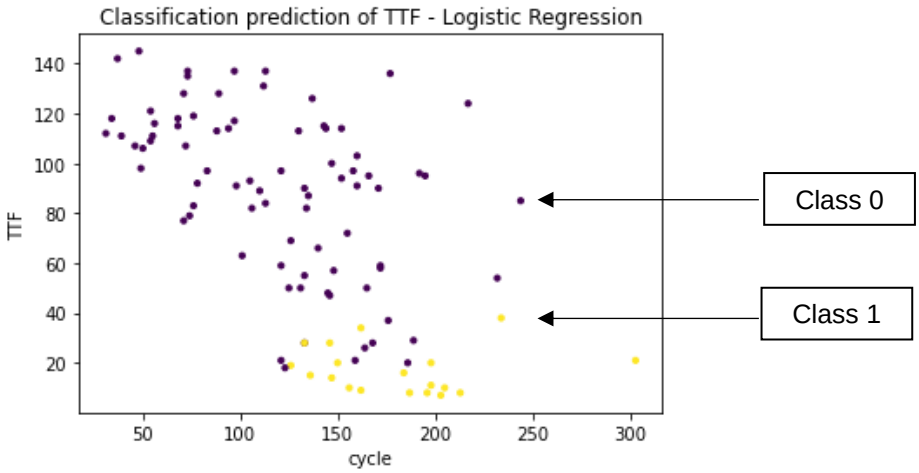
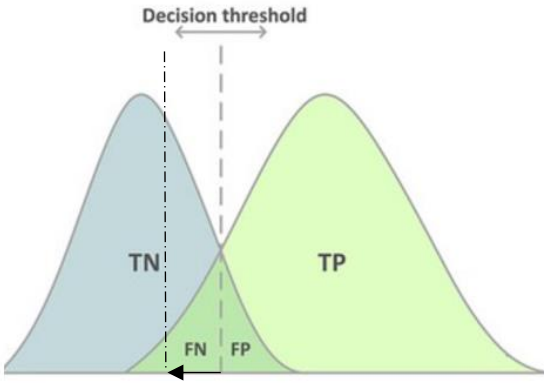


Figure 27: Classification prediction of TTF by Logistic Regression

Department: MSc AAI

Threshold setting

The discrimination threshold is the probability at which the positive class is chosen over the negative class. Generally, this is set to 50% but it can result in poor performance due to unbalance datasets.
In this case, it would be adequate to set an appropriate threshold to not biased the classifier with the dominant Class 1 and adjust the sensitivity to avoid False Negatives (moving black line on *Figure 28*).



Note: Moving the decision threshold to the left, it reduces false negatives and maximizing sensitivity.

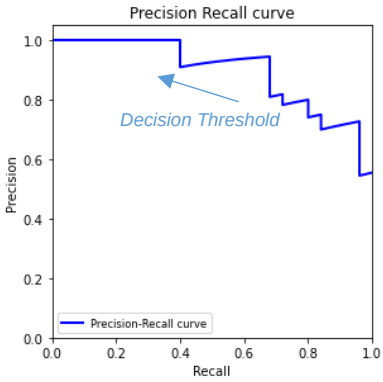


Figure 28: Decision threshold illustration

Since the *Recall* isn't perfect and some faulty engines aren't detected, it is necessary to find the appropriate decision threshold using the precision-recall curve and the ROC/AUC curve. These curves are useful tools to visualize the sensitivity (precision) and specificity (recall) trade-off in a classifier. They help inform where to set the decision threshold of the model to maximize either sensitivity or specificity.

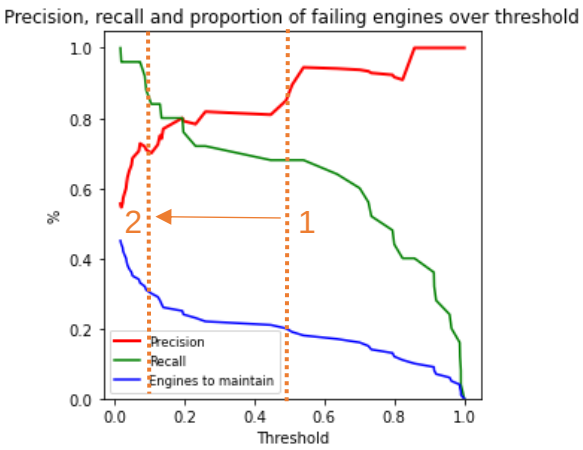


Figure 29: Precision/Recall curves for Logistic Regression

Precision-Recall Thresholds:

	Threshold	Precision	Recall
0	0.017107	0.555556	1.00
1	0.019606	0.545455	0.96
2	0.023482	0.558140	0.96
3	0.024863	0.571429	0.96
4	0.028717	0.585366	0.96
5	0.033244	0.600000	0.96
6	0.034927	0.615385	0.96
7	0.037339	0.631579	0.96
8	0.042093	0.648649	0.96
9	0.048759	0.666667	0.96
10	0.051711	0.685714	0.96
11	0.069709	0.705882	0.96
12	0.073202	0.727273	0.96
13	0.086655	0.718750	0.92
14	0.090765	0.709677	0.88
21	0.197103	0.791667	0.76
22	0.231860	0.782609	0.72
23	0.259949	0.818182	0.72
24	0.446820	0.809524	0.68
25	0.490873	0.850000	0.68
26	0.508002	0.894737	0.68
27	0.540483	0.944444	0.68
28	0.636382	0.941176	0.64
29	0.701139	0.937500	0.60
30	0.726019	0.933333	0.56
31	0.734359	0.928571	0.52
32	0.794935	0.923077	0.48
33	0.799847	0.916667	0.44

Figure 30: Data curves for Logistic Regression

On *Figure 29*, an orange line is drawn on the graph to point out the move to do to have the higher recall. This new line corresponds to a *Recall* around 96%, a *Precision* around 72% and matches to a threshold around 0.08. This means a positive class is chose 8 times over 100 over a negative class. It targets around 30% of the engines to be maintain per period. This move beckons that there are less False Negative engines (faulty but seen as correct), but the Precision has decreased due to more False Positive engines (correct but considered as faulty). This means the Arline takes less risk, but loose more money leaving more engines on the ground. Here it is the trade-off to make!

4.4 Final prediction

```
Confusion Matrix:
[[73  2]
 [ 8 17]]
```

```
Best Parameters:
 LogisticRegression(C=10, random_state=123)
```

Figure 31: Confusion matrix and best parameters for Logistic Regression

	Logistic Regression
Precision	0.894737
Recall	0.680000
F1 Score	0.772727
ROC / AUC	0.960000

Figure 32: Logistic Regression's scores (threshold = 0.5)

Y test	Y predict
1	1
0	0
0	0
1	1
1	1
1	1
1	**0**
0	0
0	0
1	1

Figure 33: Results preview extract (threshold = 0.5)

As the metrics reveal, the model is almost perfect. To predict the *TTF*, classification methods are more efficient than the regression ones.
The most suitable threshold has been fixed to 0.08 instead of the default value of 0.5, to decrease the False Negative rate. Moving the threshold, the model is more efficient and the proportion of engines to be maintain per period increases from around 20 to 30 engines (*Figure 29*). This means the faulty engines previously considered as correct are now well classed as faulty, and the risk of accident has decreased. It contains also more engines which are correct and wrongly classed as faulty rather than before, but in this way, it isn't too risky, since it isn't too serious to send an engine in maintenance even if it doesn't need it. Nevertheless, this affects in-game costs, leaving more engines on the ground than it should.

Finally, before moving it, it is necessary to assure that the maintenance capacity can support the additional operations.

	Logistic Regression
Precision	0.72
Recall	0.92
F1 Score	0.81

Figure 34: Logistic Regression's scores (threshold = 0.08)

Y test	Y predict
1	1
0	0
0	0
1	1
1	1
1	1
1	**1**
0	0
0	0
1	1

Figure 35: Results preview extract (threshold = 0.08)

5 Conclusion

Along this project, it appeared the Random Forest Regressor algorithm is the more efficient to predict the engines' Time-To-Failure by regression, but it didn't reach exact results for each prediction. At the opposite, a binary classification approach revealed very performant to classify failing engines in a given period, and particularly with the Logistic Regression algorithm.
Regarding the results expected, it is clearly unexpectable to have too many faulty engines classed as correct (low recall). Since the recall was not high enough, the performance has been improved adapting the decision threshold to limit the number of False Negatives and making a trade-off about the correct engines considered as faulty (False Positive) which is not too damaging for the risk. Indeed, an Airline needs to balance its risk appetite about maintenance and money. Political and economic values need to be studied to make the following trade-off: to prevail money avoiding leaving too many engines on the ground or take a risk over potential engines failures and letting fly the more engines as possible. Sensitively, it would make sense to limit the risk for passengers and crews as it has been done in this report, but the question must be asked.
To go further, a multiclass classification could have been performed to plan the maintenance more precisely according to the engine's failures, or maybe remove some training data in the Class 0 to improve performance.

6 Contents of figures

Figure 1: Training dataset extract 4
Figure 2: Training and test data for engine 1 4
Figure 3: Testing dataset extract 5
Figure 4: Correlation matrix between features 6
Figure 5: Scatter matrix to display relationships and distribution among features 7
Figure 6: Charts for sensor 1 8
Figure 7: Charts for sensor 2 8
Figure 8: Charts for sensor 3 8
Figure 9: Charts for sensor 4 8
Figure 10: Time series for sensor 2 selecting random sample engines 10
Figure 11: Metrics results for each algorithm 11
Figure 12: Residuals for the best algorithms 11
Figure 13: RMSE VS Degree 12
Figure 14: Random Forest's parameters 12
Figure 15: Learning curves for Polynomial Regression 13
Figure 16: Learning curves for Random Forest 13
Figure 17: Overfitting and underfitting examples for explanation 13
Figure 18: Random forest's scores 14
Figure 19: Training data repartition 15
Figure 20: Precision / Recall comparison 17
Figure 21: AUC / ROC curve explanation 17
Figure 22: Metrics results for each algorithm 18
Figure 23: AUC/ROC curves for all models 18
Figure 24: Precision Recall curves for all models 18
Figure 25: Learning curves for Logistic Regression 19
Figure 26: Learning curves for Random 19
Figure 27: Classification prediction of TTF by Logistic Regression 19
Figure 28: Decision threshold illustration 20
Figure 29: Precision/Recall curves for Logistic Regression 20
Figure 30: Data curves for Logistic Regression 20
Figure 31: Confusion matrix and best parameters for Logistic Regression 21
Figure 32: Logistic Regression's scores (threshold = 0.5) 21
Figure 33: Results preview extract (threshold = 0.5) 21
Figure 34: Logistic Regression's scores (threshold = 0.08) 21
Figure 35: Results preview extract (threshold = 0.08) 21