# Assignment 3

TU Delft | AE4465 Maintenance Modelling and Analysis | Duncan van Woerkom (4878914)

## Question 1| Predicting the Remaining Useful Lifetime (RUL)

### Introduction

The goal of this assignment is to create two machine learning models than can estimate the Remaining Useful Lifetime of turbofan engines in the CMAPSS dataset. In this report only the FD001 dataset will be used. The structure of the report will start off with the data analysis, followed by the data pre-processing, the machine learning algorithm and to finish off the evaluation of the algorithm.

### Data Quality, analysis and processing

To be able to pre-process the dataset for the machine learning algorithm, it is important to first analyse the data and by doing this learn how to process the data. To start off the dataset has been printed using the function .head(), by doing this the overview of the data can be learned. The headers have been added of the Equipment ID, Cycles, the operational settings and the sensors. Another method to analyse the data is with using Data Quality Metrics. This helps with the analyzation of the data and gives an idea of how to pre-process the data. The first metric is to check the .describe().T function. This function gives an overview and statistical data of every column, such as standard deviation, mean min and max. This is important, so an analysis can be made of faulty or disturbed sensors. Another way the data will be analysed is via an analysis of the data quality metrics. The second Data Quality metric that is used to analyze that data is the metric of data completeness. By analyzing the data completeness. By analyzing the data completeness, an idea can be found about how the data should be pre-processed. The formula of the data completeness can be found in Equation 1

$$Completeness = \frac{\text{Total non-NaN values}}{\text{Total values}} * 100 \tag{1}$$

The next data pre-processing method that has been used is the smoothing of the noisy data. The data has been smoothed with a Savitsky-Golay filter. The smoothing should result in a smooth denoised signal, that gives a good representation for the signal. The next step in the pre-processing of the data phase is dimensionality reduction. Dimensionality reduction is a common processing technique that is used in machine learning and data analysis to reduce the number of features or variables in a dataset while retaining as much information as possible. This is done because often the computational efficiency can be improved and overfitting can be reduced, as highly dimensional data is more prone to overfitting. For the dimensionality reduction, PCA dimensionality reduction is used for the dataset. For this the accompanying PCA dimensionality reduction functions are imported from the sklearn package. After an analysis of the the variance ratio of each component, the decision can be made of how many components to include. After the PCA dimensionality reduction, it is important to scale the data. The scaling prevents certain data from dominating the model and can speed the model up as well, as many machine learning algorithms work better when the data is scaled.

### Splitting, grid search, training and predicting - Random Forest Regression & Neural Networks

To be able to apply a machine learning algorithm it is important to split the dataset into a training and a testing dataset. For the splitting and training, a k-fold cross validation algorithm has been designed. The designed algorithm is slighlty different compared to the traditional algorithm. The traditional algorithm splits the dataset into training and test sets. The traditional algorithm will do this a k amount of times (usually 5 or 10 different times). Every single time a different engines are chosen for the test and train set. So every part of the dataset will be used for the testing and training. The designed algorithm works slightly different, because the engine units cannot be mixed. The difference is the fact that the self made algorithm chooses 20 engines for the testing data and 80 engines for the training data, while the regular algorithm does not take into account the engine units while splitting. For every train/test iteration it chooses a different set of engines.

For the first model Random Forest Regression has been chosen. Random Forest Regression is a machine learning algorithm that combines numerous decision trees for the training of the algorithm. This Random Forest Model has

been set up and it has been fit for the training data. Using a grid search method an optimal max depth and amount of estimators have been determined. The grid search resulted in a max depth of 10 and 200 estimators. The Random Forest algorithm has been imported from the sklearns package.
The second model that has been used is the Multilayer Perceptron Regressor. This regressor is a type of Neural Network that implements a multi-layered perceptron (MLP) that trains using back-propagation with no activation function in the output layer. This MLP regressor is also imported from the sklearn package. The dataset that the Neural Network is trained on is the same as the Random Forest Regressor (including the cross-validation).

## Kalman filter

To further improve the machine learning algorithms, a Kalman flter can be used. The Kalman filter is a filter that can blend two different estimates of systems together to make a more precise estimate. It is often used when combining measurements with calculations, for example to be able to estimate a position of navigation systems better. Not only is the Kalman filter useful in navigation, it can also be used to make estimates combining the results out of two different machine learning algorithms.
To start the algorithm, first the initial matrices $A$ (transition matrix), $H$ (observation matrix), $Q$ (process noise covariance), $P(t)$ (error covariance estimate matrix) and $R(t)$ (observation noise coverage) have to be set up together with the state and observation vectors $x(t)$ and $z(t)$ respectively. See the equations below for the vectors and matrices. In the noise covariance matrix, the variable $q$ is an arbitrary constant which affects the smoothness of the state estimate time series, ensuring that lower values result in more smoothing, here a value of 12 has been chosen as it allows for enough smoothing.

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad H = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \quad Q = \begin{bmatrix} q & 0 \\ 0 & 0 \end{bmatrix}$$

$$P(t) = \begin{bmatrix} p(t) & 0 \\ 0 & 0 \end{bmatrix} \quad x(t) = \begin{bmatrix} RuL(t) \\ -1 \end{bmatrix} \quad z(t) = \begin{bmatrix} R\hat{U}L^{RFR}(t) \\ R\hat{U}L^{NN}(t) \end{bmatrix}$$

$$R(t) = \begin{bmatrix} MSE^{RFR}_{1\to N_p} & 0 \\ 0 & MSE^{NN}_{1\to N_p} \end{bmatrix} \quad MSE^{m}_{1\to N_p} = \frac{1}{N_p}\sum_{i=1}^{N_p}\left(\hat{\tau}_i^m - \tau_i\right)^2$$

For the Kalman filter these state space matrices will be looped for every single prediction made in the previous Random Forest Regression and Neural Network. As more iterations are taken, the filter will become preciser in the estimates that it makes and its uncertainty will decrease. In this case the previous state will be used to produce estimates. The next step is to start the predicting phase after the initialization. Here the $x(t)$ and $P(t)$ will be iterated over (after every individual update). The equations for the predictions of $x(t)$ and $P(t)$ are seen in equations 3.

$$\hat{x}^-(t) = A\hat{x}(t-\Delta t) \quad P^-(t) = AP(t-\Delta t)A^\top + Q \tag{3}$$

After the predicting phase, the updating phase is starts. The predictions will have to be combined with the current observations. This is done by setting up the Kalman gain (a value from 0 to 1, where an estimate is made what the weight should be given to the separate estimates when updating the state estimate) and with the Kalman gain updating the state vector and the error covariance matrix.

$$K(t) = P^-(t)H^\top\left[HP^-(t)H^\top + R\right]^{-1} \tag{4}$$

$$\hat{x}(t) = \hat{x}^-(t) + K(t)\cdot[z(t) - H\cdot\hat{x}^-(t)] \tag{5}$$

$$P(t) = [I - K(t)H]\cdot P^-(t) \tag{6}$$

### Error Metrics, Confidence interval and Uncertain Estimate

To be able to evaluate the machine learning model, multiple error metrics are used, and of the separate machine learning models the resulting confidence intervals are set up. For the creating of an uncertain estimate the method

of bagging is used.
Three main error metrics are used, the first metric being the Mean Absolute Error. This measures the absolute difference between the predicted values and the actual values of the model. This a great measure, as it covers the magnitude of how far the predicted values are from the actual values on average and is not so sensitive to outliers. However it does not cover the direction of the error. Another similar metric is the Root Mean Squared Error, which measures the average distance between predicted and actual values, while taking into account differences between the two. It does take the direction of the error into account and it penalizes larger errors more. The third and final error metric is the Mean Absolute Percentage Error, this measures the percentage difference between predicted values and actual values. It is a good measure to compare different models with each other, as it is a relative measure. All three of the equations can be found in Equations 7. To produce the uncertain estimate, the method of bagging is used. This method works by creating multiple decision trees that use bootstrapped samples of the to be trained on data. Bootstrapped samples are randomly sampled data of the original dataset. This is the reason that some of the original samples are repeated in the new sample dataset, while others may be left out. By creating a lot of bootstrapped samples and training separate decision trees on each sample, Bagging may reduce the variance in the model's predictions and may even lead to an overall improvement of the model. When the set is trained, the algorithm combines the predictions to produce a final prediction. For this algorithm the BaggingRegressor() function is imported from sklearn.

$$MAE = \frac{1}{n}\sum_{j=1}^{n}\left|y_j - \hat{y}j\right| \quad RMSE = \sqrt{\sum i = 1^{n}\frac{\left(\hat{y}i - y_i\right)^2}{n}} \quad MAPE = \frac{1}{n}\sum i = 1^{n}\frac{\left|y_i - \hat{y}\right|}{y_i} \tag{7}$$

To calculate the 95% confidence interval of the Mean Absolute Error, Equation 8 is used. Here the mean of the list each individual mean absolute error (for every prediction) is used, together with its standard deviation and a $z = 1.96$. The $n$ is dependent on the sample size.

$$CI = \bar{x} \pm z\frac{s}{\sqrt{n}} \tag{8}$$

# Results

## Data pre-processing and Data Quality

To be able to effectively apply the machine learning algorithms, the data pre-processing is first applied. The first action has been to make the data readable and clear to humans. Making sure the proper headers are in place and the data is in a clear format. Now the data has been slightly cleaned up, it can be reviewed with the data quality metrics. To start off the first metric of the .describe().T gives us an overview of the data with its statistical attributes of the sensors. It can immediately be seen that multiple sensors are broken and that all sensors vary in their variance and deviation. Thus the dataset will benefit from PCA analysis and scaling. The second Data Quality metric is the metric of completeness, the completeness off the dataset is 100%. This is the perfect score, it can be explained from the fact that the file used in this assignment is the file from the lesson and not the online file. The lesson file has had 2 columns removed and headers installed. Thus the current file does not benefit from the removal of NaN values, however the real dataset does. Also, after some visual data analysis, the data turned out to need smoothing. The Savitsky-Golay filter was applied to all noisy sensor data (Thus not including 'Cycle' and 'Equipment' Columns) resulted in smooth signals. An example of this smoothing is found in Figure 1. The following step was the PCA dimensionality reduction, which after analysis with pca.explained_variance_ratio_ and trail and error resulted in a system with 6 components. Finally the data was scaled for the machine learning algorithms to work optimally. To be able to apply the machine learning algorithms, the Remaining Useful Lifetimes (RUL) were calculated and added to the dataframe. An overview of the data is given in Figure 2.

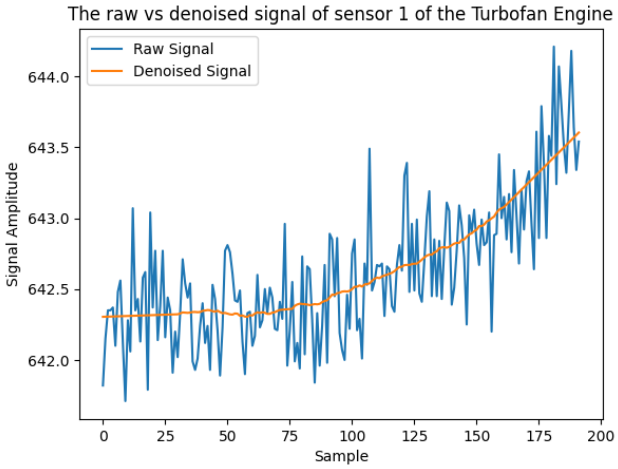



Figure 1: One of the denoised sensors

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| Equipment | 20631.0 | 51.506568 | 29.227633 | 1.0 | 26.000000 | 52.000000 | 77.000000 | 100.0 |
| Cycle | 20631.0 | 108.807862 | 68.880990 | 1.0 | 52.000000 | 104.000000 | 156.000000 | 362.0 |
| 0 | 20631.0 | 0.208263 | 0.119703 | 0.0 | 0.144515 | 0.179577 | 0.227891 | 1.0 |
| 1 | 20631.0 | 0.372096 | 0.194261 | 0.0 | 0.222139 | 0.359674 | 0.485174 | 1.0 |
| 2 | 20631.0 | 0.499184 | 0.114172 | 0.0 | 0.426203 | 0.498041 | 0.572605 | 1.0 |
| 3 | 20631.0 | 0.450928 | 0.115984 | 0.0 | 0.372073 | 0.451898 | 0.528991 | 1.0 |
| 4 | 20631.0 | 0.468740 | 0.100196 | 0.0 | 0.406093 | 0.470980 | 0.530429 | 1.0 |
| 5 | 20631.0 | 0.387939 | 0.098646 | 0.0 | 0.328323 | 0.388201 | 0.450694 | 1.0 |
| RUL | 20631.0 | 107.807862 | 68.880990 | 0.0 | 51.000000 | 103.000000 | 155.000000 | 361.0 |

Figure 2: Overview of the Scaled and PCA transformed data

## Performance

Error Metrics, Cross Validation, Grid Search and Data Quality

To research the performance of the model it is important to use metrics for its analysis. The previously discussed metrics will be analysed in this subsection. In an early version of the software the machine learning algorithm resulted in a Mean Absolute Error of around 8-9 for both regressors. After analysing the data the conclusion was made that overfitting occurred. To combat this problem a cross validation algorithm was set up (see section Splitting, grid search, training and predicting - Random Forest Regression  Neural Networks). With this cross validation algorithm the overall performance of the two Machine Learning Methods were still quite accurate. For the arguments of the algorithms, a grid search method was applied to find the optimal results (this is commented in the code). The Random Forest Regression and MLP Regressor had a Mean Absolute Error of 24.26 and 25.71 respectively. The implemented bagging method that was applied resulted in similar, however not improved results, thus has been commented out of the code. After the applying of the Kalman filter the Mean Absolute Error resulted in a MAE of 24.64, being a logical value for the filter. Two data sampling tests were done, one with 50 samples and one with 100 samples. In Table 1 the corresponding RMSE, MAE, MAPE and Confidence interval are displayed.

Table 1: results overview

| | Sample size = 50 | | | Sample size = 100 | | |
|---|---|---|---|---|---|---|
| Model | RF Regressor | MLP Regressor | Kalman Filter | RF Regressor | MLP Regressor | Kalman Filter |
| MAE | 23.50 | 27.16 | 24.50 | 23.34 | 26.77 | 24.80 |
| RMSE | 31.03 | 32.41 | 30.87 | 31.36 | 33.87 | 31.56 |
| MAPE | 19.18% | 26.12% | 23.73% | 23.34% | 26.77% | 24.80% |
| CI | [17.75, 29.26] | [22.13, 32.18] | [19.16, 29.83] | [19.18, 27.50] | [22.81, 30.72] | [20.93. 28.67] |

Alpha-Lambda plots and performance analysis

A good way to analyse the performance of the predictions of the machine learning models is through Alpha-Lambda plots. Here the Predicted Remaining Useful Lifetime is set against the Actual Remaining Useful Lifetime. In Figure 3, Figure 4 and Figure 5 the corresponding plots for the RFR and SVR ML methods are presented. These plots also contain a linear regression line and an upper and lower bound of the Remaining Useful Lifetime based on an $\alpha = 0.2$.

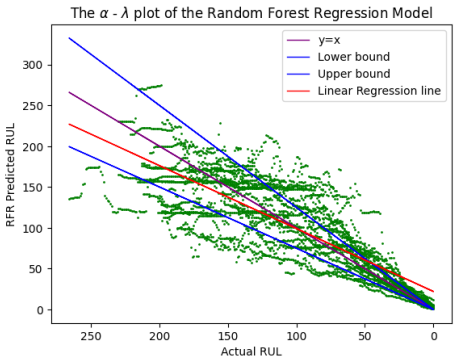



Figure 3: Alpha Lambda plot of the predictions made with the RFR

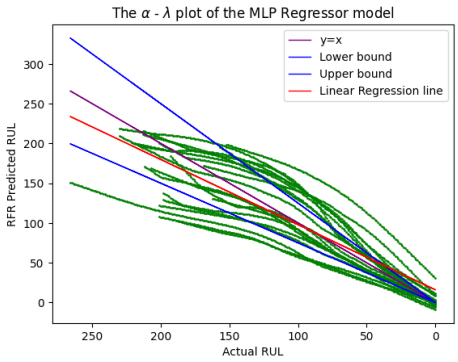



Figure 4: Alpha Lambda plot of the predictions made with the MLP Regressor

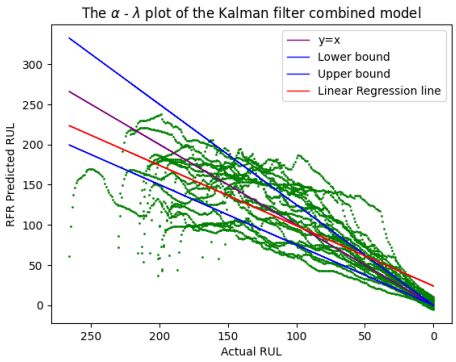



Figure 5: Alpha Lambda plot of the predictions made with the Kalman filter

After analyzing the Alpha - Lambda plots it can be noted that the Random Forest Regressor has a slightly more random prediction, whereas the MLP Regressor and the Kalman filter have almost continuous prediction lines going through the Alpha - Lambda plots. The observation can be made that in all three of the plots, the trend of each individual engine leads at a steady slope to the (0,0) coordinate. Meaning that early predictions of the models are not always as accurate, as these are quite spread out, however predictions closer to the breakdown point are more accurate and in the last 30 cycles a good estimate can often be made when maintenance is due. When observing the Alpha - Lambda plot of the Kalman filter trends of both the machine learning algorithms can be found. This insinuates that the Kalman filter does indeed work and choose which prediction to favour over the course of the cycles or per specific engine. There is definitely room for improvement in the results. Further enhancing the pre-processing of the data and the tuning of the algorithms can result in a better result. The MAE of both algorithms are slightly over 24, while there is definitely room for the MAE to go under 20. Further research could be done for better predictions.

## Conclusion

This Assignment has been quite an extensive and tough research on the C-MAPSS dataset. During the course of the project I have learnt how to do research into predicting events with Machine Learning Algorithms. I have experimented with many different algorithms and now know the beginnings of many different algorithms. The way of working towards the machine learning models by data pre-processing and analyzation also had quite an impact on me. By further enhancing the potential of Data to be absorbed by these algorithms a lot can be achieved. The result of this paper is already successful, however there is still room for improvement and a lot to learn. This has been quite an introduction to predictive maintenance and definitely not the last.

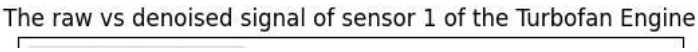
```
import numpy as np
import pandas as pd
import math
import scipy.stats
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import PCA
from scipy.signal import savgol_filter
from copy import deepcopy
from sklearn.model_selection import cross_val_score
from scipy import signal
from sklearn.neural_network import MLPClassifier, MLPRegressor
from scipy.stats import t
import missingno as msno
import scipy.stats as st
from sklearn.ensemble import BaggingRegressor

#Noise reduction
# units = np.unique(df1['Equipment'].values)
# df2 = df1.copy()

# for unit in units:
#     for name_signal in range(21):
#         signal = df1.loc[df1.Equipment == unit, str(name_signal + 1)]
#         signal_mav = signal.rolling(20).mean().values
#         signal_mav[:30] = signal[:30].values
#         df2.loc[df2.Equipment == unit, str(name_signal + 1)] = signal_mav



df1 = pd.read_csv('Lecture 10/data/les05_CMAPSStrain001.txt', sep = ' ')
df = pd.read_csv('Dataset/RUL_FD001.csv')
df = pd.DataFrame(df)
df1 = pd.DataFrame(df1)

df1.describe().T
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **Equipment** | 20631.0 | 51.506568 | 2.922763e+01 | 1.0000 | 26.0000 | 52.0000 | 77.0000 | 100.0000 |
| **Cycle** | 20631.0 | 108.807862 | 6.888099e+01 | 1.0000 | 52.0000 | 104.0000 | 156.0000 | 362.0000 |
| **Op1** | 20631.0 | -0.000009 | 2.187313e-03 | -0.0087 | -0.0015 | 0.0000 | 0.0015 | 0.0087 |
| **Op2** | 20631.0 | 0.000002 | 2.930621e-04 | -0.0006 | -0.0002 | 0.0000 | 0.0003 | 0.0006 |
| **Op3** | 20631.0 | 100.000000 | 0.000000e+00 | 100.0000 | 100.0000 | 100.0000 | 100.0000 | 100.0000 |

```
df1_unsmoothed = deepcopy(df1)

equipment_ID = np.unique(df1['Equipment'].values)
sensors = list(df1.columns[5:])
print(sensors)
savgol_window = 61

for equipment in equipment_ID:
    equipment_data = df1.loc[df1['Equipment'] == equipment]
    for sensor in sensors:
        equipment_data[sensor] = savgol_filter(equipment_data[sensor].values, window_length=savgol_window, polyorder=1, axis=0)
    df1.loc[df1['Equipment'] == equipment] = equipment_data

# for col in df1.columns:
#     if col not in ['Cycle', 'Equipment']:
#         df1[col] = savgol_filter(df1[col], window_length=125, polyorder=2, mode = 'mirror')
# df1_first_two_cols = df1.iloc[:, :2]
# smooth_data = df1.iloc[:,2:]

# for column in range(len(smooth_data.columns)):
#     smooth_data =signal.savgol_filter(smooth_data, window_length=21, polyorder=3, )

# smooth_data = pd.DataFrame(smooth_data)
# smooth_data = pd.concat([df1_first_two_cols, smooth_data], axis=1)

# smooth_data = pd.DataFrame(smooth_data)
# smooth_data.columns = df1.columns
```

```
['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21']
C:\Users\Duncan van Woerkom\AppData\Local\Temp\ipykernel_21844\4039113168.py:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  equipment_data[sensor] = savgol_filter(equipment_data[sensor].values, window_length=savgol_window, polyorder=1, axis=0)
```

```
raw_signal = df1_unsmoothed.loc[df1_unsmoothed.Equipment == 1, str(2)]
denoised_signal = df1.loc[df1.Equipment == 1, str(2)]

plt.plot(raw_signal, label='Raw Signal')
plt.plot(denoised_signal, label='Denoised Signal')
plt.xlabel('Sample')
plt.ylabel('Signal Amplitude')
plt.title('The raw vs denoised signal of sensor 1 of the Turbofan Engine')
plt.legend()
plt.show()
```




```
# remove columns with all missing values
df1 = df1.dropna(axis=1, how='all')
df1 = df1.fillna(method = 'ffill')
```

```
#PCA Dimensionality Reduction
df_last_24 = df1.iloc[:, -24:]
n_components = 6

pca = PCA(n_components=n_components)
pca.fit(df_last_24)
df_last_24_transformed = pca.transform(df_last_24)

print(pca.explained_variance_ratio_)

df_transformed = pd.concat([df1.iloc[:, :2], pd.DataFrame(df_last_24_transformed)], axis=1)
```

```
[9.10089588e-01 8.91847183e-02 3.81423347e-04 3.01830390e-04
 2.98153614e-05 6.78667666e-06]
```




```
failure_cycle = []
for engine in df1['Equipment']:
    failure_cycle.append(((df1[df1['Equipment'] == engine]['Cycle']).values)[-1])
df_transformed['failure cycle'] = failure_cycle

RUL = df_transformed['failure cycle'] - df1['Cycle']
df_transformed['failure cycle'] = RUL
df_transformed = df_transformed.rename(columns={'failure cycle': 'RUL'})
df_transformed.describe().T
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **Equipment** | 20631.0 | 5.150657e+01 | 29.227633 | 1.000000 | 26.000000 | 52.000000 | 77.000000 | 100.000000 |
| **Cycle** | 20631.0 | 1.088079e+02 | 68.880990 | 1.000000 | 52.000000 | 104.000000 | 156.000000 | 362.000000 |
| **0** | 20631.0 | -1.449919e-13 | 29.052573 | -50.546655 | -15.471828 | -6.962175 | 4.763896 | 192.159773 |
| **1** | 20631.0 | 1.669457e-13 | 9.094687 | -17.420355 | -7.020487 | -0.581559 | 5.293980 | 29.396521 |
| **2** | 20631.0 | 1.202450e-13 | 0.594766 | -2.600443 | -0.380183 | -0.005952 | 0.382481 | 2.608947 |
| **3** | 20631.0 | -4.221914e-13 | 0.529083 | -2.057002 | -0.359712 | 0.004424 | 0.356099 | 2.504704 |
| **4** | 20631.0 | -6.357706e-14 | 0.166289 | -0.777933 | -0.103971 | 0.003717 | 0.102380 | 0.881692 |
| **5** | 20631.0 | -8.139005e-14 | 0.079336 | -0.312001 | -0.047947 | 0.000210 | 0.050470 | 0.492251 |
| **RUL** | 20631.0 | 1.078079e+02 | 68.880990 | 0.000000 | 51.000000 | 103.000000 | 155.000000 | 361.000000 |

```
#scaling the data
df1_scaled = df_transformed.copy()
df1_scaler = MinMaxScaler()
df1_scaled.iloc[:,2:-1] = df1_scaler.fit_transform(df1_scaled.iloc[:,2:-1])
df1_scaled = df1_scaled.loc[:, (df1_scaled != 0).any(axis=0)]
# df1_scaled = df1.drop(columns=df1.columns[(df1 == 0).all()])
df1_scaled.describe().T
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **Equipment** | 20631.0 | 51.506568 | 29.227633 | 1.0 | 26.000000 | 52.000000 | 77.000000 | 100.0 |
| **Cycle** | 20631.0 | 108.807862 | 68.880990 | 1.0 | 52.000000 | 104.000000 | 156.000000 | 362.0 |
| **0** | 20631.0 | 0.208263 | 0.119703 | 0.0 | 0.144515 | 0.179577 | 0.227891 | 1.0 |
| **1** | 20631.0 | 0.372096 | 0.194261 | 0.0 | 0.222139 | 0.359674 | 0.485174 | 1.0 |
| **2** | 20631.0 | 0.499184 | 0.114172 | 0.0 | 0.426203 | 0.498041 | 0.572605 | 1.0 |
| **3** | 20631.0 | 0.450928 | 0.115984 | 0.0 | 0.372073 | 0.451898 | 0.528991 | 1.0 |
| **4** | 20631.0 | 0.468740 | 0.100196 | 0.0 | 0.406093 | 0.470980 | 0.530429 | 1.0 |
| **5** | 20631.0 | 0.387939 | 0.098646 | 0.0 | 0.328323 | 0.388201 | 0.450694 | 1.0 |
| **RUL** | 20631.0 | 107.807862 | 68.880990 | 0.0 | 51.000000 | 103.000000 | 155.000000 | 361.0 |

```
# X_train = pd.DataFrame(df1_scaled.iloc[:, 0:-1])
# Y_train = pd.DataFrame(df1_scaled.iloc[:, -1])
```

```
# num_runs = 1
# mae_list = []
# MSRE_list = []
# for num in range(num_runs):
#     X_training = pd.DataFrame(df1_scaled.iloc[:, 2:-1])
#     Y_training = pd.DataFrame(df1_scaled.iloc[:, -1])

#     print(Y_training)

#     X_train, X_test, y_train, y_test = train_test_split(X_training, Y_training, test_size = 0.20, random_state = 0)
#     RFR = RandomForestRegressor(max_depth = 10, n_estimators = 200) #[zonder max depth, MAE = 25.03, RMSE = 35,9]

#     # Grid search
#     # param_grid = {'n_estimators': range(50,350,50] 'max_depth': [5,8,10,12]}

#     # grid_search = GridSearchCV(RFR, param_grid, cv=5, scoring='neg_mean_squared_error')
#     # grid_search.fit(X_train, y_train)

#     # print('Best hyperparameters of the RFR:', grid_search.best_params_)
#     # print('Best score:', np.sqrt(-grid_search.best_score_))

#     RFR.fit(X_train, y_train)
#     y_pred = RFR.predict(X_test)

#     mae = mean_absolute_error(y_test, y_pred)
#     mae_list.append(mae)

#     mse = mean_squared_error(y_test, y_pred)
#     MSRE = np.sqrt(mse)
#     MSRE_list.append(MSRE)

# mae_mean = np.mean(mae_list)
# mae_std = np.std(mae_list)

# # Calculate the confidence interval
# confidence_interval = (mae_mean - 1.96 * (mae_std / np.sqrt(num_runs)), mae_mean + 1.96 * (mae_std / np.sqrt(num_runs)))

# # Results
# print("Mean MAE: {:.2f}".format(mae_mean))
# print("Standard deviation of MAE: {:.2f}".format(mae_std))
# print("95% Confidence Interval: ({:.2f}, {:.2f})".format(confidence_interval[0], confidence_interval[1]))




# plt.title('Random Forest Regression plot with a 95% Confidence Interval')
# plt.xlabel('MAE')
# plt.ylabel('probability density')

# # Plot the MAE samples
# plt.hist(mae_list, bins=10)
# plt.axvline(mae_mean, color='r', linestyle='-')


# scores = cross_val_score(RFR, X_train, y_train, cv=5)
# print("Average score:", scores.mean())
# # Plot the lower and upper bounds of the confidence interval as horizontal lines
# # plt.axhline(lower_bound, color='g', linestyle='--')
# # plt.axhline(upper_bound, color='g', linestyle='--')

# plt.show()


# score = RFR.score(X_test, y_test)
# print(score)


train_list = []
test_list = []

for engine in equipment_ID:
    train_list.append(pd.DataFrame(df1_scaled.loc[df1_scaled['Equipment'] == engine, df1_scaled.columns[1:-1]]))
    test_list.append(pd.DataFrame(df1_scaled.iloc[(df1_scaled['Equipment'] == engine).values, 1:-1]))




score_list = []
mae2_list = []

NN_score_list = []
NN_mae_list = []
```

```
RFR_bagging_mae_list = []
RFR_bagging_score_list = []

for i in range(1):
    # Cross validation without splitting equipment_ID's
    train_equipment = pd.Series(equipment_ID).sample(n=80, random_state=i)
    test_equipment = pd.Series(equipment_ID).drop(train_equipment.index)

    train_data = df1_scaled.loc[df1_scaled['Equipment'].isin(train_equipment)].drop('Equipment', axis=1)
    train_X = train_data.drop('RUL', axis=1)
    train_X = train_X.drop('Cycle', axis=1)
    train_y = train_data['RUL']


    test_data = df1_scaled.loc[df1_scaled['Equipment'].isin(test_equipment)].drop('Equipment', axis=1)
    test_X = test_data.drop('RUL', axis=1)
    test_X = test_X.drop('Cycle', axis=1)
    test_y = test_data['RUL']


    # RFR = RandomForestRegressor(max_depth = 10, n_estimators = 200,random_state=i)
    RFR = RandomForestRegressor(max_depth = 10, n_estimators = 10,random_state=i)
    RFR.fit(train_X, train_y)

    # bagging
    # bagging_rfr = BaggingRegressor(base_estimator=RFR, max_samples = 0.8, n_estimators= 50, random_state=i)
    # bagging_rfr.fit(train_X, train_y)


    NN = MLPRegressor(max_iter=300, activation = 'relu', hidden_layer_sizes = (100,100))
    NN.fit(train_X , train_y)

    # Evaluation RFR
    y_prediction = RFR.predict(test_X)
    mae2 = mean_absolute_error(test_y, y_prediction)
    mse_RFR = mean_squared_error(test_y, y_prediction)
    mae2_list.append(mae2)

    score = RFR.score(test_X, test_y)
    score_list.append(score)

    # Evaluation NN
    NN_pred = NN.predict(test_X)
    mae_NN = mean_absolute_error(test_y, NN_pred)

    NN_mae_list.append(mae_NN)
    mse_NN = mean_squared_error(test_y, NN_pred)

    NN_score = NN.score(test_X, test_y)
    NN_score_list.append(NN_score)

    # # # Evaluation Bagging
    # y_pred_bagging = bagging_rfr.predict(test_X)
    # RFR_bagging_mae = mean_absolute_error(test_y, y_pred_bagging)

    # RFR_bagging_mae_list.append(RFR_bagging_mae)
    # mse_RFR_bagging = mean_squared_error(test_y, y_pred_bagging)

    # RFR_bagging_score = bagging_rfr.score(test_X, test_y)
    # RFR_bagging_score_list.append(RFR_bagging_score)
```

```
c:\Users\Duncan van Woerkom\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\neural_network\_multilayer_perceptron
  warnings.warn(
```

```
# print('MAE RFR:', mean_absolute_error(test_y, y_prediction))
# print('MAE NN:', mean_absolute_error(test_y, NN_pred))


def MAE_list_maker(prediction, test):
    mae_list =np.abs(prediction-test)
    return mae_list



from scipy.stats import t
import random
```

```
RFR_mae_full_list = MAE_list_maker(y_prediction, test_y)

NN_mae_full_list = MAE_list_maker(NN_pred, test_y)

# y_pred_bagging = MAE_list_maker(y_pred_bagging,test_y)
random.seed(123)
def metrics_calc(pre_CI_list, prediction, test_y, confidence_level = 0.95, sample_size = 100):

    pre_CI_list_new = pd.DataFrame(pre_CI_list).sample(n = sample_size, random_state = 2)
    prediction_new = pd.DataFrame(prediction).sample(n = sample_size, random_state = 2)
    test_y_new = pd.DataFrame(test_y).sample(n = sample_size, random_state = 2)

    pre_CI_list_new = np.array(pre_CI_list_new)
    prediction_new = np.array(prediction_new)
    test_y_new = np.array(test_y_new)


    MAPE = 100 * np.mean(np.abs((test_y_new - prediction_new)/ len(test_y_new)))
    MAE = (1/len(test_y_new)) * sum(np.abs(test_y_new - prediction_new))
    RMSE = np.sqrt(np.mean((np.array(prediction_new) - np.array(test_y_new))**2))

    mean = np.mean(pre_CI_list_new)
    std_dev = np.std(pre_CI_list_new)

    df = sample_size - 1
    alpha = 1 - confidence_level
    t_critical = t.ppf(1 - alpha / 2, df)

    margin_of_error = t_critical * std_dev / np.sqrt(sample_size)

    lower_bound = mean - margin_of_error
    upper_bound = mean + margin_of_error

    #
    print("Metrics of the list:\n Confidence Interval: [{:.2f}, {:.2f}]\n MAPE: {:.2f}\n MAE: {} \n RMSE: {}".format(lower_bound, upper_b



alpha = 0.25

def get_trendline(x,y):
    slope, intercept = np.polyfit(x, y, 1)
    return slope*x + intercept


# plt.xlabel('Actual RUL')
# plt.ylabel('RFR Predicted RUL')
# plt.title(r'The $\alpha$ - $\lambda$ plot of the Random Forest Regressor model')
# plt.scatter(test_y, y_prediction
#  , s= 1, color = 'green')
# plt.plot(test_y, test_y, label = "y=x", color = 'purple', linewidth = 1)
# plt.plot(test_y, test_y*(1-alpha), label = "Lower bound", color = 'blue', linewidth = 1)
# plt.plot(test_y, test_y*(1+alpha), label = "Upper bound", color = 'blue', linewidth = 1)
# plt.plot(test_y, get_trendline(test_y, y_prediction), color = 'red' ,linestyle = '-' , label = 'Linear Regression line', linewidth = 1)
# plt.gca().invert_xaxis()
# plt.legend()
# plt.show()

# plt.xlabel('Actual RUL')
# plt.ylabel('RFR Predicted RUL')
# plt.title(r'The $\alpha$ - $\lambda$ plot of the MLP Regressor model')
# plt.scatter(test_y, NN_pred
#  , s= 1, color = 'green')
# plt.plot(test_y, test_y, label = "y=x", color = 'purple', linewidth = 1)
# plt.plot(test_y, test_y*(1-alpha), label = "Lower bound", color = 'blue', linewidth = 1)
# plt.plot(test_y, test_y*(1+alpha), label = "Upper bound", color = 'blue', linewidth = 1)
# plt.plot(test_y, get_trendline(test_y, NN_pred), color = 'red' ,linestyle = '-' , label = 'Linear Regression line', linewidth = 1)
# plt.gca().invert_xaxis()
# plt.legend()
# plt.show()



# np.savetxt('RFR_mae_list', mae2_list, fmt='%.18e', delimiter=' ', newline='\n', header='', footer='', comments='# ', encoding=None)
# np.savetxt('NN_mae_list', NN_mae_list, fmt='%.18e', delimiter=' ', newline='\n', header='', footer='', comments='# ', encoding=None)


#Kalman filter
kalman_joint = []
q = 12
A = pd.DataFrame([[1 , 1], [0 , 1]])
```

```
H = pd.DataFrame([[1 , 0], [1 , 0]])
Q = pd.DataFrame([[q**2 , 0], [0 , 0]])

#Initialization of the state vector matrix and the covariance matrix
x_t = pd.DataFrame([[np.mean([y_prediction[0], NN_pred[0]])], [-1]])
P_t = pd.DataFrame([[np.mean([(y_prediction[0] - test_y.iloc[0]),(NN_pred[0] - test_y.iloc[0])])**2 , 0], [0 , 0]])

for i in range(len(y_prediction)):
    # Current iteration predicted values
    y_prediction_t = y_prediction[:i + 1] # predicted with RFR
    NN_pred_t = NN_pred[:i + 1] # predicted with NN
    test_y_t = test_y.iloc[:i + 1] # test_y values
    R_t = pd.DataFrame([[mean_squared_error(test_y_t, y_prediction_t), 0], [0, mean_squared_error(test_y_t, NN_pred_t)]]) # Observation r

    # Predicting phase
    x_t = A @ x_t
    P_t = A @ P_t @ A.T + Q
    # Kalman gain
    K_t = P_t @ H.T @ np.linalg.inv(H @ P_t @ H.T + R_t)
    # Updating phase
    z_t = pd.DataFrame([[y_prediction[i]], [NN_pred[i]]])
    x_t = x_t + K_t @ (z_t - H @ x_t)
    P_t = (np.eye(2) - K_t @ H) @ P_t
    kalman_joint.append(x_t[0][0])

mae_kalman = mean_absolute_error(test_y, kalman_joint)
print(mae_kalman)
```

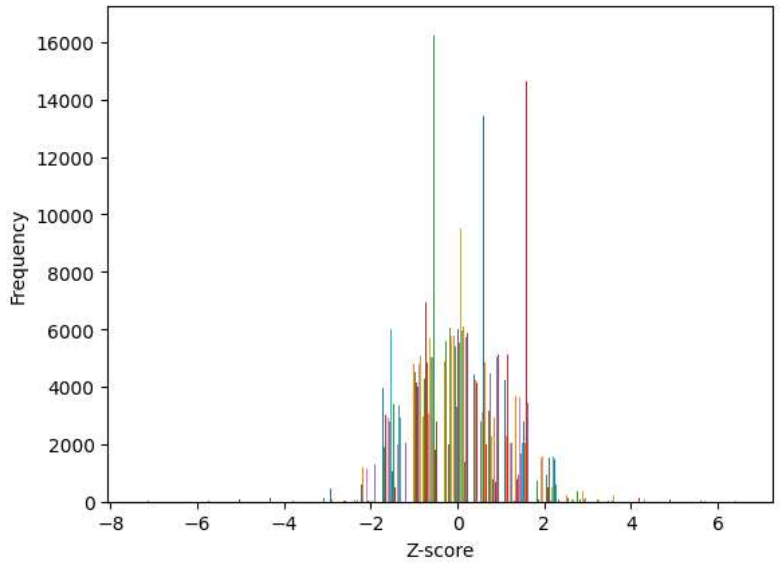
```
25.843626040500585
```

```
# plt.xlabel('Actual RUL')
# plt.ylabel('RFR Predicted RUL')
# plt.title(r'The $\alpha$ - $\lambda$ plot of the Kalman filter combined model')
# plt.scatter(test_y,kalman_joint
#  , s= 1, color = 'green')
# plt.plot(test_y, test_y, label = "y=x", color = 'purple', linewidth = 1)
# plt.plot(test_y, test_y*(1-alpha), label = "Lower bound", color = 'blue', linewidth = 1)
# plt.plot(test_y, test_y*(1+alpha), label = "Upper bound", color = 'blue', linewidth = 1)
# plt.plot(test_y, get_trendline(test_y, kalman_joint), color = 'red' ,linestyle = '-' , label = 'Linear Regression line', linewidth = 1)
# plt.gca().invert_xaxis()
# plt.legend()
# plt.show()


from scipy.stats import t

RFR_mae_full_list = MAE_list_maker(y_prediction, test_y)
NN_mae_full_list = MAE_list_maker(NN_pred, test_y)
Kalman_mae_full_list = MAE_list_maker(kalman_joint, test_y)

def metrics_calc(pre_CI_list, prediction, test_y, confidence_level = 0.95, sample_size = 100):

    pre_CI_list_new = pd.DataFrame(pre_CI_list).sample(n = sample_size, random_state = 2)
    prediction_new = pd.DataFrame(prediction).sample(n = sample_size, random_state = 2)
    test_y_new = pd.DataFrame(test_y).sample(n = sample_size, random_state = 2)

    pre_CI_list_new = np.array(pre_CI_list_new)
    prediction_new = np.array(prediction_new)
    test_y_new = np.array(test_y_new)


    MAPE = 100 * np.mean(np.abs((test_y_new - prediction_new)/ len(test_y_new)))
    MAE = float((1/len(test_y_new)) * sum(np.abs(test_y_new - prediction_new)))
    RMSE = np.sqrt(np.mean((np.array(prediction_new) - np.array(test_y_new))**2))

    mean = np.mean(pre_CI_list_new)
    std_dev = np.std(pre_CI_list_new)

    df = sample_size - 1
    alpha = 1 - confidence_level
    t_critical = t.ppf(1 - alpha / 2, df)

    margin_of_error = t_critical * std_dev / np.sqrt(sample_size)

    lower_bound = mean - margin_of_error
    upper_bound = mean + margin_of_error

    #
    print("Metrics of the list:\n Confidence Interval: [{:.2f}, {:.2f}]\n MAPE: {:.2f}\n MAE: {:.2f} \n RMSE: {:.2f}".format(lower_bound,
```

```
# metrics_calc(RFR_mae_full_list, y_prediction, test_y)
# metrics_calc(NN_mae_full_list, NN_pred, test_y)
# metrics_calc(Kalman_mae_full_list, kalman_joint, test_y)
# metrics_calc(RFR_bagging_mae_list, y_pred_bagging,test_y)
```

```
from scipy import stats
z_scores = (df1 - np.mean(df1)) / np.std(df1)

# Create a histogram of the Z-scores
plt.hist(z_scores, bins=20)
plt.xlabel("Z-score")
plt.ylabel("Frequency")
plt.show()
```

```
c:\Users\Duncan van Woerkom\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\core\fromnu
  return mean(axis=axis, dtype=dtype, out=out, **kwargs)
c:\Users\Duncan van Woerkom\AppData\Local\Programs\Python\Python310\lib\site-packages\matplotlib\axes\_
  xmin = min(xmin, np.nanmin(xi))
c:\Users\Duncan van Woerkom\AppData\Local\Programs\Python\Python310\lib\site-packages\matplotlib\axes\_
  xmax = max(xmax, np.nanmax(xi))
```




```
completeness = df.count() / len(df) * 100

# print completeness of each column
print(completeness)
```

```
112     100.0
dtype: float64
```

```
# np.savetxt('mae_list_file', mae_list, fmt='%.18e', delimiter=' ', newline='\n', header='', footer='', comments='# ', encoding=None)
# np.savetxt('MSRE_list_file', MSRE_list, fmt='%.18e', delimiter=' ', newline='\n', header='', footer='', comments='# ', encoding=None)
```