Predicting Aircraft Equipment Failure using Machine Learning Classification Algorithms

Veer Kumar[1], Madhura Mokashi[2]

[1]*Student, Dept. of Electronics and Telecommunications and Engineering, Rajiv Gandhi Institute of Technology, Mumbai, Maharashtra, India*
[2]*Assistant Professor, Dept. of Electronics and Telecommunications and Engineering, Rajiv Gandhi Institute of Technology, Mumbai, Maharashtra, India*

---***---

Abstract - *Enormous amount of information and maintenance data exists in the aviation industry that can be utilized to draw meaningful insights in forecasting the future course of action. In this study, our prime objective is to use machine learning classification models to perform feature selection and predictive analysis to predict failures of aircraft systems. Maintenance and failure data for aircraft equipment across a period of two years were collected, and cleaned, which was followed by application feature engineering and feature selection techniques before model building and evaluation. We compute a metric known as Remaining Useful Life(RUL) to predict the failure of aircraft equipment, since this is a continuous variable, we then convert it into a binary classification problem by setting a threshold RUL value to indicate an impending failure so that our classification model flags a warning well in advance to the point of breakdown, thereby giving response teams sufficient time to act upon the warning. Experimental results of our classification model demonstrate the effectiveness of our model to forecast the failure of aircraft equipment.*

Key Words: Predictive Maintainance, Predictive Analytics, Classification, Feature selection.

1. INTRODUCTION

To prevent high maintenance costs and prolonged downtime, we must prevent the failure of critical safety systems such as aircraft jet engines. This predicts a system's point of failure imperative to ensure the proper functioning of the overall aircraft system. For this purpose, we use a metric known as Remaining Useful Life(RUL) to estimate how long given industrial machinery can keep operating before a system failure occurs. Once we can deploy these prognostic techniques successfully, it will give us sufficient buffer time to make interventions or take corrective measures towards restoring normalcy in the operation of malfunctioning parts. Such predictive maintenance techniques have the capability of reducing operational costs, downtime of machinery, and avoiding the risk of an unintended mishap if systems are not routinely checked for the defect. However, to fully understand predictive maintenance, we must first look at the traditional existing methods such as reactive and scheduled maintenance. Firstly, reactive maintenance is defined as the approach to repairing parts or equipment only after the asset has broken down or been run to the point of failure. Reactive maintenance may seem like an attractive choice since it offers the maximum utilization of machinery and in turn provides maximum production output, of the asset by using it to its limits. This can prove to be advantageous, only until the point where the asset fails. What's worse is that the cost of repairing the damaged machinery post-failure can be worth more than the production value received by running it to failure. Thereafter comes scheduled maintenance which consists of maintenance tasks performed while the equipment is under normal operation to avoid unexpected breakdowns that come along with increased downtime and overhead costs. Scheduled maintenance aims to extend the life of machinery, increase productivity, improve overall efficiency and reduce costs. It is hard to justify why preventive maintenance is the most suitable approach when it requires greater planned downtime, for machines that are still running at optimal levels, to be taken offline and operations to be disrupted. The disadvantages of the aforementioned methods have been overcome by predictive maintenance, which has been enabled by advances in technology that connect your asset to your data historian and/or CMMS through sensor data. It constantly monitors the performance of our machinery during normal operation to anticipate and predict failures. Predictive maintenance will analyze the data gathered from sensors connected to machine parts. We can use this data to estimate when the point of failure will occur and this information will provide maintenance teams sufficient time to troubleshoot any existing issues that may be causing the unexpected downtime. In this paper we first define the predictive maintenance problem statement we intend to solve from the aviation industry. The following sections introduce the dataset chosen, followed by proposed methodology and experimental results from the models we built to predict the failure of a given set of machinery and the evaluation metrics to assess the accuracy of our models.

2.0 Problem Statement

Our first step is to identify the target variable which we want to predict. Since our dataset consists of columns containing sensor readings, it initially isn't intuitive as to what will help us arrive at our target variable. Through some analysis, we realized that based on the maximum number of cycles until failure, we can compute the Remaining useful life or RUL. After which we will approach predictive maintenance as a binary classification problem

3.0 Dataset

We have used NASA's turbofan engine degradation simulation dataset (CMAPSS) dataset, which is a collection of four datasets of increasing complexity. Initially, the engines will function normally, however, over time they will begin to develop some kinds of fault. Our aim here is to compute a variable that will depict the Remaining Useful Life (RUL) of each turbofan engine. Thereafter, we shall use this RUL variable, to set a threshold RUL value, below or above which we will indicate the need for maintenance using a binary variable. For this research paper, we will be exploring the first dataset (FD001). The FD001 include simulations of multiple turbofan engines spread over time, every row includes the information given below:

A. Engine unit number

B. Time, in cycles

C. Three operational settings

D. 21 sensor readings

4.0 Literature Review

Maintenance issues can be completely different and the predictive information to be fed to the Predictive maintenance module needs to be modified according to the particular problem at hand [1]. The C-MAPSS dataset has been extensively used to evaluate several machine learning and deep learning approaches to RUL predictions. Firstly, According to Samuel, A.L. [5], Machine learning mainly means that if computers are allowed to solve without specifically being programmed in doing so. ML approaches are known to have significant advantages, as they can manage multivariate, high dimensional data and can extract hidden relationships within data in complex, dynamic, and chaotic environments [6,7]. ML applications provide some real-time utilities which include maintenance cost reduction, repair stop reduction, machine fault reduction, spare-part life increases and inventory reduction, operator safety enhancement, increased production, repair verification, an increase in overall profit, and many more. These advantages also have a tremendous and strong bond with the procedures of maintenance [6,8]. It is required that any maintenance strategy requires to minimize equipment failure rates, must improve equipment condition, should prolong the life of the equipment, and reduce the maintenance costs. ANN and SVM ML algorithms are applied in developing gauge degradation measurements prediction for two types of rail track including straight and curved segments by Falamarzi, A. et al. [11]. Random was developed by Breidman.L[12], this is an ensemble learning technique that consists of several decision tree classifiers, the output is collectively determined by individual trees. Binding, A. et al. [62] reported a study on forecasting the downtime of a printing machine based on real-time predictions of imminent failures. In their study, they utilized unstructured historical machine data to train the ML classification algorithms including RF, XGBoost, and LR in predicting the machine failures. Various metrics were analyzed to determine the goodness of fit of the models. Janssens, O. et al. [14] reported a study on DL for Infrared Thermal Image Based Machine Health Monitoring, wherein the study they considered Convolutional Neural Network (CNN), a Feature Learning (FL) tool, in detecting the various conditions of the machine. PdM turned out to be one of the most promising strategies amongst other strategies of maintenance that can achieve those characteristics [9], thus the strategy has been applied recently in many fields of study. PdM captivates the attention of the industries, hence it has been applied in the era of I4.0 due to it is the capability of optimizing the use and management of assets [6,10]. In this paper, our goal is to leverage machine learning classification algorithms that can be used for predictive maintenance. We will be approaching the predictive maintenance problem as a binary classification problem. For instance, if the lead time for ordering new spare parts for repairs is 20 days, we would want our system to send us a signal much before failure, so our system should flag a warning when an engine is likely to break within the next 30 days, so that response teams can take the required course of action to address the issue.

5.0 Libraries used

1. Pandas: It is a fast and efficient tool to carry out data analysis and data manipulation. In addition to that, it allows us to import data from varied file formats such as JSON, CSV's, SQL, and Microsoft EXCEL, enabling us to perform complex data cleaning and wrangling operations on these files to read, interpret and gather insights from raw data.

2. Numpy: It is an array processing package that allows us to perform several scientific computations on arrays, including several high-level mathematical functions.

3. Sklearn: This is the fundamental machine learning library in Python, It contains a wide variety of modules to aid machine learning and statistical modeling processes such as classification, regression, clustering, and dimensionality reduction.

4. Matplotlib: It is a well-known data visualization package in python, a cross-platform library that can be used to create two-dimensional graphs from data in arrays. Here we can use simple functions for adding plot elements, such as lines, images, text, etc. to the axes in the current figure.

6.0 Algorithms

1. SVM

Support Vector Machine(SVM) is a supervised machine learning algorithm that we most often use for classification or regression problem statements. It uses a method known as kernel trick to transform our input data, to identify suitable boundaries to classify the labels that have been fed to it.

2. RANDOM FOREST

The Random forest classification model is made up of several decision trees. In simple terms, it combines the results from numerous decision trees to reach a single result. The main difference between decision trees and random forests is that decision trees consider all the possible feature splits, however, random forests will only select a subset of those features.

3. KNN

The K-nearest neighbors algorithm is a supervised machine learning algorithm, which will store all the available cases and will classify new data-point-based similarity metrics like say distance function. It follows an instance-based learning approach, wherein, entire training instances are used to predict output from unseen data.

4. LOGISTIC REGRESSION

Logistic regression is used to predict the probability of the target variable. For Logistic Regression, the dependent variable must be binary which can either be 0 or 1, yes or no, success or failure. It is one of the most fundamental supervised learning methods that has found significant applications in spam detection, and also in the detection of Diabetes and Cancer disease.

5. NAÏVE BAYES

The Naïve Bayes algorithm has been built upon the foundation of the Bayes Theorem. Using Bayes theorem, we can find out the probability of the occurrence of A given B has occurred. There are two major assumptions, first, we consider each of our features to be independent, and second, all the features/predictors have an equal effect on the outcome variable.

6. DECISION TREE

The Decision tree, just as the name suggests, is a tree-like structure to depict that the final predictions stem from a series of feature-based splits. It all starts with a root node where the population divides based on several features. After splitting the root nodes, we obtain decision nodes. In simple terms, the decision tree consists of a group of if-else statements, it will check if the given condition is true and move on to the next associated to the prior decision.

7.0 Methodology:

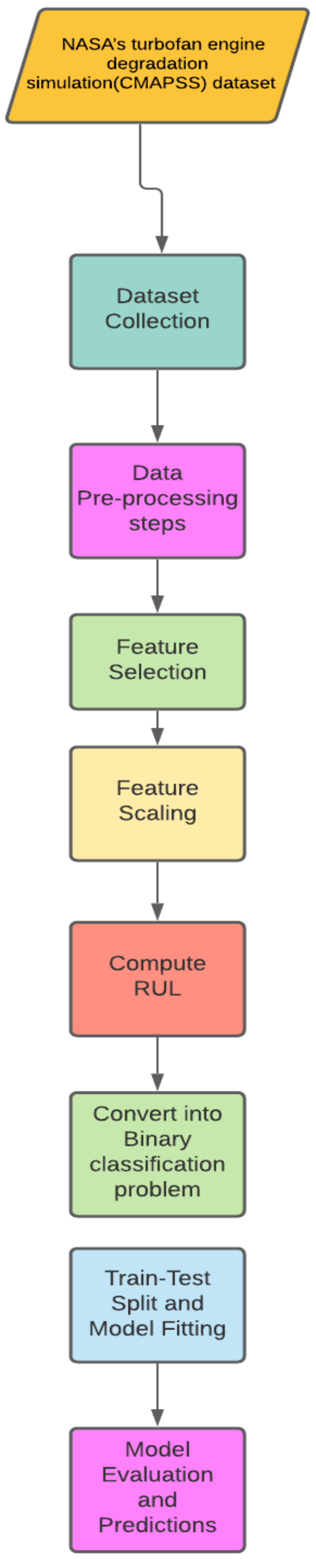

NASA's turbofan engine degradation simulation(CMAPSS) dataset
Dataset Collection
Data Pre-processing steps
Feature Selection
Feature Scaling
Compute RUL
Convert into Binary classification problem
Train-Test Split and Model Fitting
Model Evaluation and Predictions

Figure 1. Flowchart to depict Methodology adopted

7.1 Data collection:

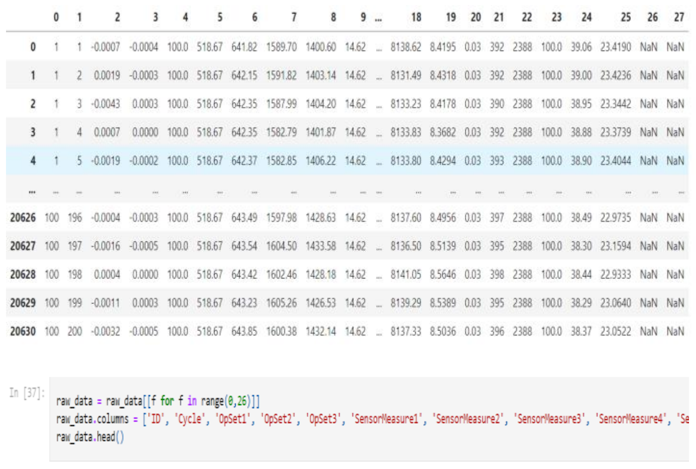
We feed the CMAPSS dataset as a CSV file and read it into pandas, converting it into a data frame, which enables us to perform various numerical operations on the dataset to transform it.

7.2 Data preprocessing:

The CMAPSS dataset consists of raw data, and there are certain transformations the dataset must undergo before we can begin our machine learning model-building process.

7.2.1 -Renaming columns

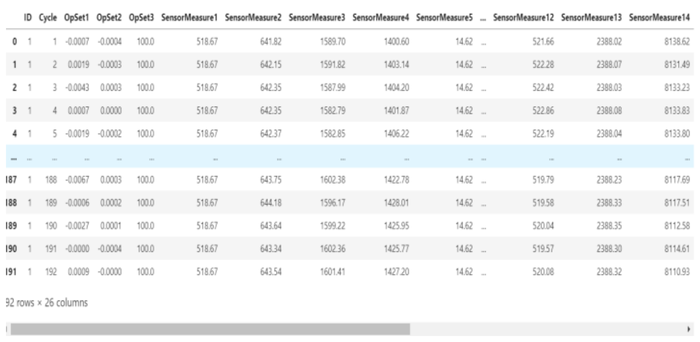
We will first begin by truncating our dataset to exclude the last two columns, since we can observe that the entries in these last two columns only consist of 'NaN' values, thereby, being extraneous and redundant to our analysis. Thereafter, we will rename every feature in our dataset with its appropriate label, to make our dataset more intuitive, to get a fair idea of what columns exist and their respective functionality. We will use the columns attribute to name all the columns with their correct labels. After naming is complete, we finally have one column for each identification of engine and the number of cycles, a total of three operational setting features, and twenty-one sensor labels.



	0	1	2	3	4	5	6	7	8	9	...	18	19	20	21	22	23	24	25	26	27
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	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	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	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	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	100.0	38.90	23.4044	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
20626	100	196	-0.0004	-0.0003	100.0	518.67	643.49	1597.98	1428.63	14.62	...	8137.60	8.4956	0.03	397	2388	100.0	38.49	22.9735	NaN	NaN
20627	100	197	-0.0016	-0.0005	100.0	518.67	643.54	1604.50	1433.58	14.62	...	8136.50	8.5139	0.03	395	2388	100.0	38.30	23.1594	NaN	NaN
20628	100	198	0.0004	0.0000	100.0	518.67	643.42	1602.46	1428.18	14.62	...	8141.05	8.5646	0.03	398	2388	100.0	38.44	22.9333	NaN	NaN
20629	100	199	-0.0011	0.0003	100.0	518.67	643.23	1605.26	1426.53	14.62	...	8139.29	8.5389	0.03	395	2388	100.0	38.29	23.0640	NaN	NaN
20630	100	200	-0.0032	-0.0005	100.0	518.67	643.85	1600.38	1432.14	14.62	...	8137.33	8.5036	0.03	396	2388	100.0	38.37	23.0522	NaN	NaN

```
In [37]:
raw_data = raw_data[[f for f in range(0,26)]]
raw_data.columns = ['ID', 'Cycle', 'OpSet1', 'OpSet2', 'OpSet3', 'SensorMeasure1', 'SensorMeasure2', 'SensorMeasure3', 'SensorMeasure4', 'Se
raw_data.head()
```

Figure 2. Renaming labels using columns mehtods



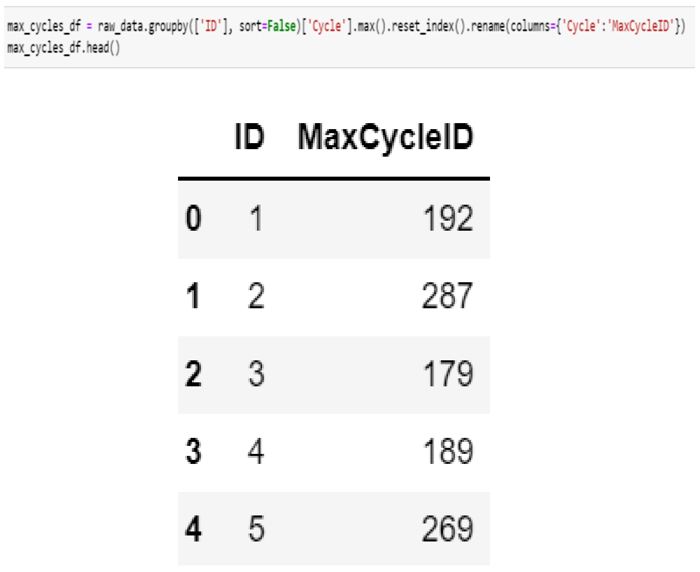
	ID	Cycle	OpSet1	OpSet2	OpSet3	SensorMeasure1	SensorMeasure2	SensorMeasure3	SensorMeasure4	SensorMeasure5	...	SensorMeasure12	SensorMeasure13	SensorMeasure14
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	521.66	2388.02	8138.62
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	522.28	2388.07	8131.49
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	522.42	2388.03	8133.23
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	522.86	2388.08	8133.83
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	522.19	2388.04	8133.80
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
187	1	188	-0.0067	0.0003	100.0	518.67	643.75	1602.38	1422.78	14.62	...	519.79	2388.23	8117.69
188	1	189	-0.0006	0.0002	100.0	518.67	644.18	1596.17	1428.01	14.62	...	519.58	2388.33	8117.51
189	1	190	-0.0027	0.0001	100.0	518.67	643.64	1599.22	1425.95	14.62	...	520.04	2388.35	8112.58
190	1	191	-0.0000	-0.0004	100.0	518.67	643.34	1602.36	1425.77	14.62	...	519.57	2388.30	8114.61
191	1	192	0.0009	-0.0000	100.0	518.67	643.54	1601.41	1427.20	14.62	...	520.08	2388.32	8110.93

92 rows × 26 columns

Figure 3. All columns with renamed labels

7.2.2 Computing maximum cycles:

The cycle column indicates the number of journeys our jet engine has completed. If we take a look at a specific engine by 'ID', we can observe how the number of cycles progressively increases with each successive journey. Here, the final row is considered to be the final journey, which is significant since it is the maximum cycle value. We will create a column called 'MaxCycleID' to represent the maximum number of journeys made by a particular engine before it experienced a state of failure. To perform this operation, we shall group the dataset by 'ID' and then reference the 'Cycle' column, getting the maximum value for Cycles using the max() method.



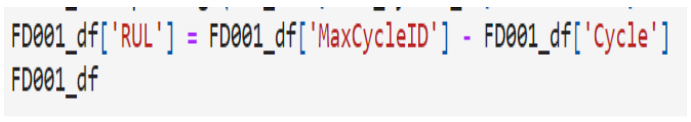
```
max_cycles_df = raw_data.groupby(['ID'], sort=False)['Cycle'].max().reset_index().rename(columns={'Cycle':'MaxCycleID'})
max_cycles_df.head()
```

	ID	MaxCycleID
0	1	192
1	2	287
2	3	179
3	4	189
4	5	269

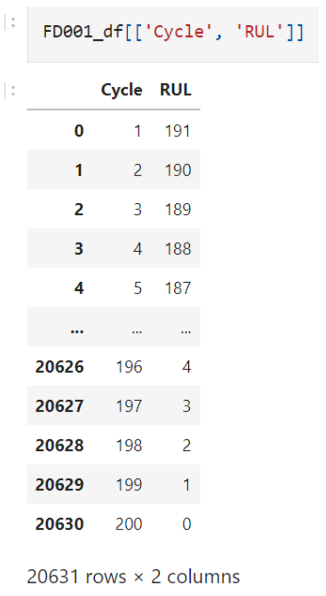
Figure 4. Dataframe to display MaxCycleID

7.2.3 Calculate Remaining Useful Life (RUL):

The remaining useful life is the equivalent of the number of flights that remained for the engine after the last data point in the test dataset. Here we will use the 'MaxCycleID' column we created earlier and subtract it from the 'Cycles' column. In simple terms, this will display the Remaining useful life at every row entry before the engine ultimately runs to failure.



```
FD001_df['RUL'] = FD001_df['MaxCycleID'] - FD001_df['Cycle']
FD001_df
```



```
FD001_df[['Cycle', 'RUL']]
```

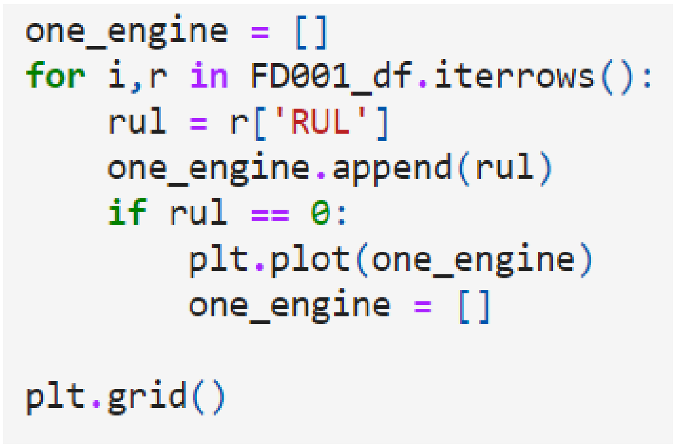
	Cycle	RUL
0	1	191
1	2	190
2	3	189
3	4	188
4	5	187
...	...	...
20626	196	4
20627	197	3
20628	198	2
20629	199	1
20630	200	0

20631 rows × 2 columns

Figure 5. Computing RUL using MaxCycleID and Cycle

7.2.4 Visualizing run to failure cycles:

Now, we visualized all of the hundred jet engines in our dataset run-to-failure cycles. For this purpose, we create an empty list called "one_engine", this is supposed to depict the run-to-failure cycle of just one jet engine in our dataset. Consequently, we iterate through our dataset using the errors method and store the RUL values in a variable called 'rul' which is then appended to the one_engine list. Finally, at some point the rul will reach zero, this is when we plot the trajectory which is followed by assigning an empty list to one engine again so that the next engine's RUL values can be stored. This is an iterative process and will continue to go on till we finish iterating through all the hundred engines in our dataset and plot their subsequent RUL plots.



```
one_engine = []
for i,r in FD001_df.iterrows():
    rul = r['RUL']
    one_engine.append(rul)
    if rul == 0:
        plt.plot(one_engine)
        one_engine = []

plt.grid()
```

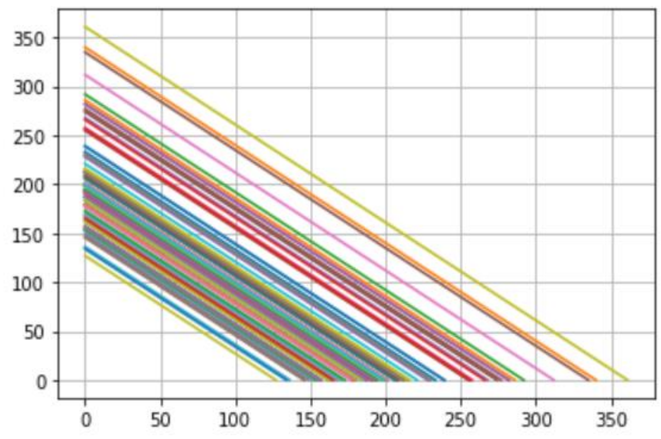
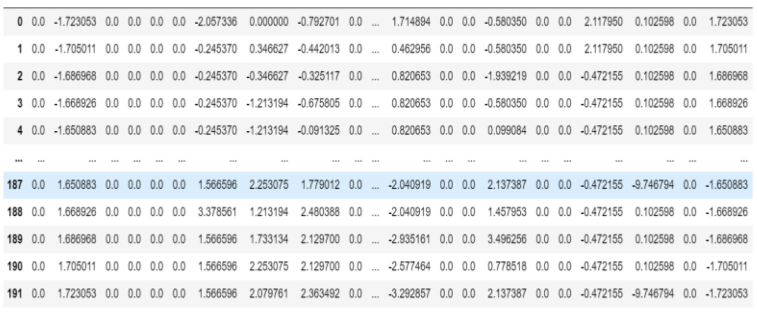


Figure 6. Visualization of run-to-failure cycles

7.3 Feature Scaling:

As we can infer from the above dataset, all of our features have different units of measurement, thus, there is a wide variation in the range of feature values between these columns. To account for this difference, we need to standardize our features by re-scaling them, we do this by applying a feature scaling technique known as StandardScaler() function.

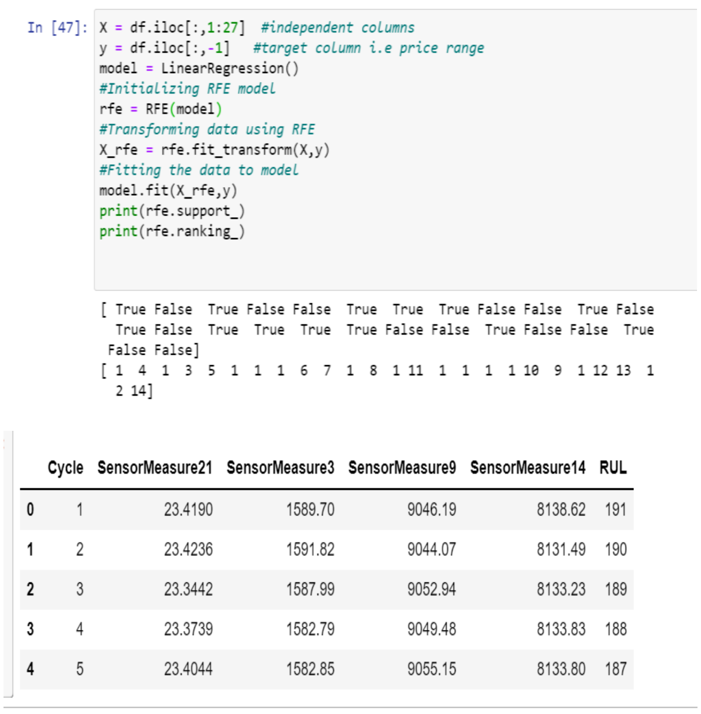


0	0.0	-1.723053	0.0	0.0	0.0	0.0	-2.057336	0.000000	-0.792701	0.0	...	1.714894	0.0	0.0	-0.580350	0.0	0.0	2.117950	0.102598	0.0	1.723053
1	0.0	-1.705011	0.0	0.0	0.0	0.0	-0.245370	0.346627	-0.442013	0.0	...	0.462956	0.0	0.0	-0.580350	0.0	0.0	2.117950	0.102598	0.0	1.705011
2	0.0	-1.686968	0.0	0.0	0.0	0.0	-0.245370	-0.346627	-0.325117	0.0	...	0.820653	0.0	0.0	-1.939219	0.0	0.0	-0.472155	0.102598	0.0	1.686968
3	0.0	-1.668926	0.0	0.0	0.0	0.0	-0.245370	-1.213194	-0.675805	0.0	...	0.820653	0.0	0.0	-0.580350	0.0	0.0	-0.472155	0.102598	0.0	1.668926
4	0.0	-1.650883	0.0	0.0	0.0	0.0	-0.245370	-1.213194	-0.091325	0.0	...	0.820653	0.0	0.0	0.099084	0.0	0.0	-0.472155	0.102598	0.0	1.650883
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
187	0.0	1.650883	0.0	0.0	0.0	0.0	1.566596	2.253075	1.779012	0.0	...	-2.040919	0.0	0.0	2.137387	0.0	0.0	-0.472155	-9.746794	0.0	-1.650883
188	0.0	1.668926	0.0	0.0	0.0	0.0	3.378561	1.213194	2.480388	0.0	...	-2.040919	0.0	0.0	1.457953	0.0	0.0	-0.472155	0.102598	0.0	-1.668926
189	0.0	1.686968	0.0	0.0	0.0	0.0	1.566596	1.733134	2.129700	0.0	...	-2.935161	0.0	0.0	3.496256	0.0	0.0	-0.472155	0.102598	0.0	-1.686968
190	0.0	1.705011	0.0	0.0	0.0	0.0	1.566596	2.253075	2.129700	0.0	...	-2.577464	0.0	0.0	0.778518	0.0	0.0	-0.472155	0.102598	0.0	-1.705011
191	0.0	1.723053	0.0	0.0	0.0	0.0	1.566596	2.079761	2.363492	0.0	...	-3.292857	0.0	0.0	2.137387	0.0	0.0	-0.472155	-9.746794	0.0	-1.723053

Figure 7. Columns after performing feature scaling

7.4 Feature Selection:

Feature selection refers to the methods used to extract the most relevant features from a dataset. Selecting fewer features can allow machine learning algorithms to run more efficiently and be more effective. We will be using the Recursive Feature Elimination(RFE) method that works by recursively eliminating attributes and building a model on those attributes that remain. It uses an accuracy metric to rank the feature according to their importance. It then assigns a ranking of all the variables, 1 being the most important. It also provides its support, True being a relevant feature and False being an irrelevant feature.



```
In [47]: X = df.iloc[:,1:27]  #independent columns
         y = df.iloc[:,-1]   #target column i.e price range
         model = LinearRegression()
         #Initializing RFE model
         rfe = RFE(model)
         #Transforming data using RFE
         X_rfe = rfe.fit_transform(X,y)
         #Fitting the data to model
         model.fit(X_rfe,y)
         print(rfe.support_)
         print(rfe.ranking_)
```

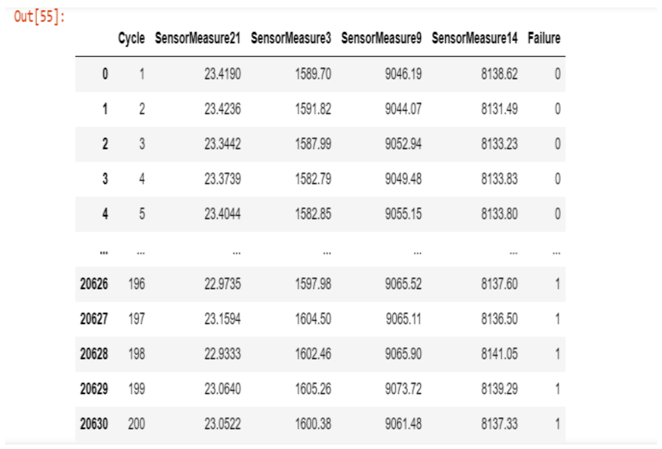
```
[ True False  True False False  True  True  True False False  True False
  True False  True  True  True  True False False  True False False  True
 False False]
[ 1  4  1  3  5  1  1  1  6  7  1  8  1 11  1  1  1  1 10  9  1 12 13  1
  2 14]
```

	Cycle	SensorMeasure21	SensorMeasure3	SensorMeasure9	SensorMeasure14	RUL
0	1	23.4190	1589.70	9046.19	8138.62	191
1	2	23.4236	1591.82	9044.07	8131.49	190
2	3	23.3442	1587.99	9052.94	8133.23	189
3	4	23.3739	1582.79	9049.48	8133.83	188
4	5	23.4044	1582.85	9055.15	8133.80	187

Figure 8. Applying RFE to select relevant feautres

7.5 Creating a binary outcome column:

We now created a binary outcome variable and named it 'Failure'. This was to convert our problem statement into a classification problem, which was previously a regression problem because of the presence of continuous variables. To achieve this, we have created a discrete target variable 'Failure', with a view that the system notifies us whether the engine is going to fail shortly. We set a threshold value for RUL below which the Failure variable would turn from zero to one, flagging a maintenance issue. We can set the threshold value in terms of days, as per our preference, however for this study, we have selected a threshold value of 30 days, since it avoids the problems that arise due to an imbalanced dataset by setting the threshold value slightly lower.



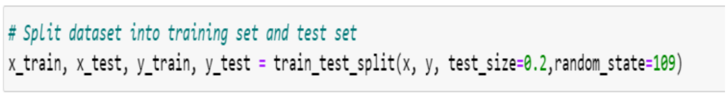
Out[55]:

	Cycle	SensorMeasure21	SensorMeasure3	SensorMeasure9	SensorMeasure14	Failure
0	1	23.4190	1589.70	9046.19	8138.62	0
1	2	23.4236	1591.82	9044.07	8131.49	0
2	3	23.3442	1587.99	9052.94	8133.23	0
3	4	23.3739	1582.79	9049.48	8133.83	0
4	5	23.4044	1582.85	9055.15	8133.80	0
...	...	...	...	...	...	...
20626	196	22.9735	1597.98	9065.52	8137.60	1
20627	197	23.1594	1604.50	9065.11	8136.50	1
20628	198	22.9333	1602.46	9065.90	8141.05	1
20629	199	23.0640	1605.26	9073.72	8139.29	1
20630	200	23.0522	1600.38	9061.48	8137.33	1

Figure 9. Creation of Binary Output variable called 'Failure'

7.6 Train-Test Split and Model Fitting:

Now, we divide our dataset into training and testing data. Our objective for doing this split is to assess the performance of our model on unseen data and to determine how well our model has generalized on training data. This is followed by a model fitting which is an essential step in the model building process.



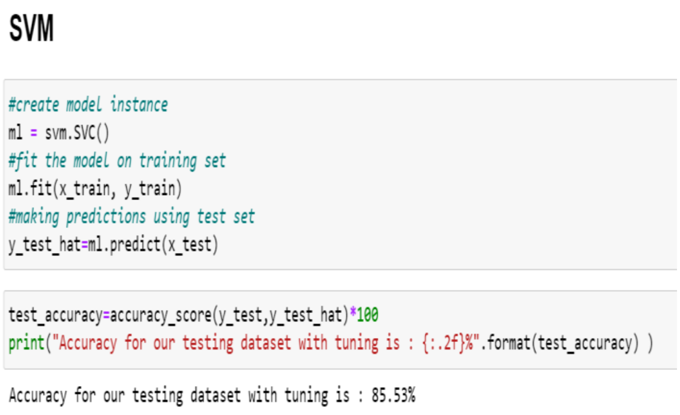
```
# Split dataset into training set and test set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,random_state=109)
```

Figure 11. Applying Train-Test split

7.7 Model Evaluation and Predictions:

This is the final step, in which we assess how well our model has performed on testing data using certain scoring metrics, I have used 'accuracy_score' to evaluate my model. First, we create a model instance, this is followed by fitting the training data on the model using a fit method and then we will use the predict method to make predictions on x_test or the testing data, these predictions will be stored in a variable called y_test_hat. For model evaluation, we will feed the y_test and y_test_hat into the accuracy_score function and store it in a variable called test_accuracy, a variable that will hold the testing accuracy of our model. We followed these steps for a variety of classification algorithm models and obtained corresponding test accuracy scores.

8.0 OUTPUTS:



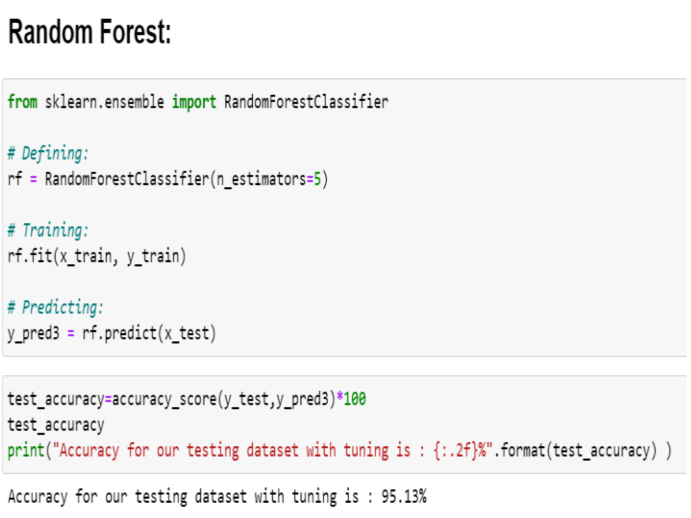
SVM

```
#create model instance
ml = svm.SVC()
#fit the model on training set
ml.fit(x_train, y_train)
#making predictions using test set
y_test_hat=ml.predict(x_test)
```

```
test_accuracy=accuracy_score(y_test,y_test_hat)*100
print("Accuracy for our testing dataset with tuning is : {:.2f}%".format(test_accuracy) )
```

```
Accuracy for our testing dataset with tuning is : 85.53%
```

Figure 12. Output for SVM classifier model

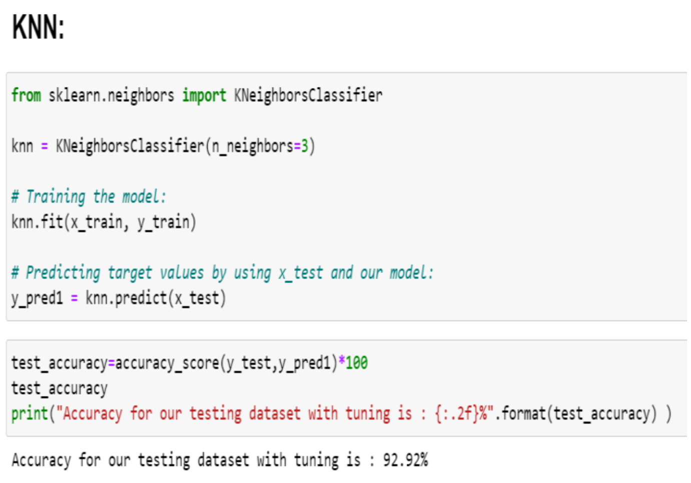

Random Forest:

```
from sklearn.ensemble import RandomForestClassifier

# Defining:
rf = RandomForestClassifier(n_estimators=5)

# Training:
rf.fit(x_train, y_train)

# Predicting:
y_pred3 = rf.predict(x_test)
```

```
test_accuracy=accuracy_score(y_test,y_pred3)*100
test_accuracy
print("Accuracy for our testing dataset with tuning is : {:.2f}%".format(test_accuracy) )
```

```
Accuracy for our testing dataset with tuning is : 95.13%
```

Figure 13. Output for Random Forest classifier model

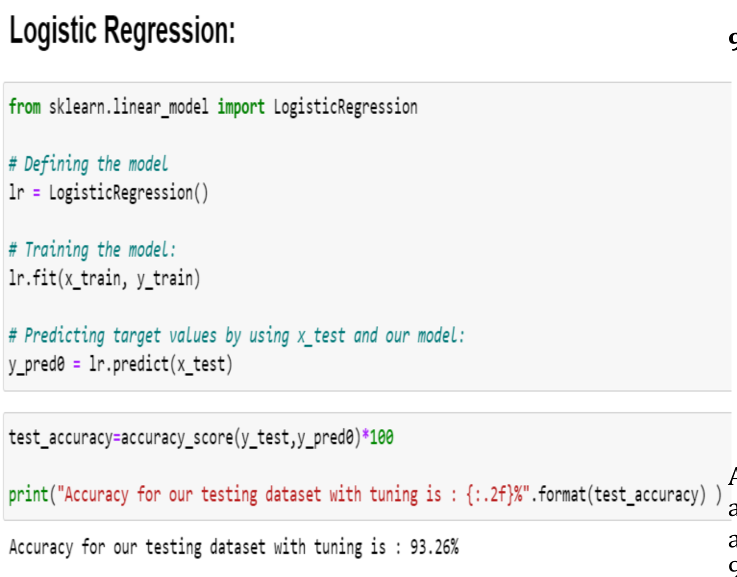

KNN:

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)

# Training the model:
knn.fit(x_train, y_train)

# Predicting target values by using x_test and our model:
y_pred1 = knn.predict(x_test)
```

```
test_accuracy=accuracy_score(y_test,y_pred1)*100
test_accuracy
print("Accuracy for our testing dataset with tuning is : {:.2f}%".format(test_accuracy) )
```

```
Accuracy for our testing dataset with tuning is : 92.92%
```

Figure 14. Output for KNN classifier model

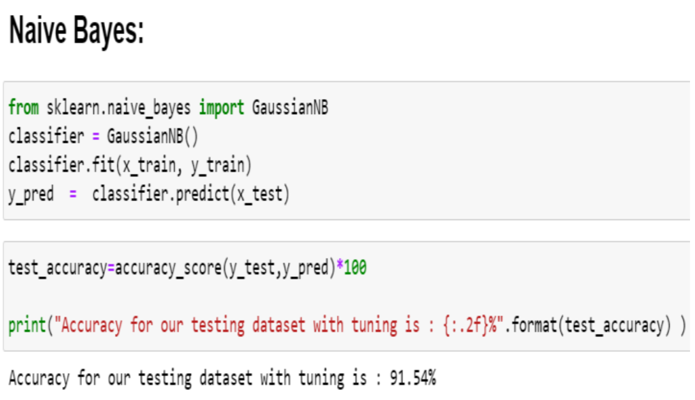

Logistic Regression:

```
from sklearn.linear_model import LogisticRegression

# Defining the model
lr = LogisticRegression()

# Training the model:
lr.fit(x_train, y_train)

# Predicting target values by using x_test and our model:
y_pred0 = lr.predict(x_test)
```

```
test_accuracy=accuracy_score(y_test,y_pred0)*100

print("Accuracy for our testing dataset with tuning is : {:.2f}%".format(test_accuracy) )
```

```
Accuracy for our testing dataset with tuning is : 93.26%
```

Figure 15. Output for Logistic Regression classifier model

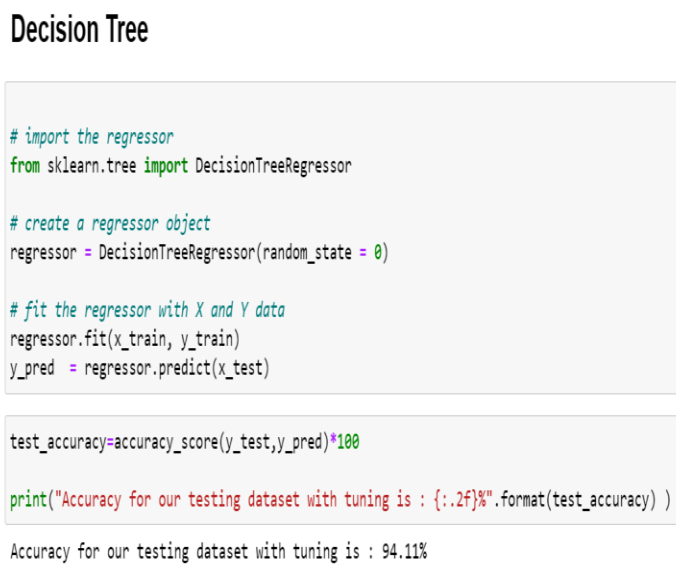

Naive Bayes:

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(x_train, y_train)
y_pred  =  classifier.predict(x_test)
```

```
test_accuracy=accuracy_score(y_test,y_pred)*100

print("Accuracy for our testing dataset with tuning is : {:.2f}%".format(test_accuracy) )
```

```
Accuracy for our testing dataset with tuning is : 91.54%
```

Figure 16. Output for Naïve Bayes classifier model


Decision Tree

```
# import the regressor
from sklearn.tree import DecisionTreeRegressor

# create a regressor object
regressor = DecisionTreeRegressor(random_state = 0)

# fit the regressor with X and Y data
regressor.fit(x_train, y_train)
y_pred  = regressor.predict(x_test)
```

```
test_accuracy=accuracy_score(y_test,y_pred)*100

print("Accuracy for our testing dataset with tuning is : {:.2f}%".format(test_accuracy) )
```

```
Accuracy for our testing dataset with tuning is : 94.11%
```

Figure 17. Output for Decision Tree classifier model

9.0 RESULTS AND INFERNECES:

Algorithms	Test Accuracy
SVM	85.53%
Random Forest	95.13%
KNN	92.92%
Logistic Regression	93.26%
Naïve Bayes	91.54%
Decision Tree	94.11%

Table 1.Test accuracies for the various classification models implemented in Tabular format

As we can see from the above table, the Random Forest algorithm does perform marginally better than the other algorithms under consideration, with a test accuracy of 95.13%. While on the other end of the spectrum we have the Support Vector Machine algorithm performing poorly when compared with other models.

10.0 CONCLUSION

The main objective of predictive maintenance is to predict when equipment failures can occur. Then prevent that failure by taking relevant actions. Predictive Maintenance System (PMS) monitors future failures and will schedule maintenance in advance.

11.0 REFERENCES

[1]M. Luo, Z. Xu, H.L. Chan, and M. Alavi. Online predictive maintenance approach for semiconductor equipment. In Industrial Electronics Society, IECON 2013-39th Annual Conference of the IEEE, pages 3662–3667. IEEE, 2013.

[2] Khelif. (2014). Unsupervised Kernel Regression Modeling Approach for RUL Prediction.

[3] Saxena A., Goebel K. Turbofan engine degradation simulation data set. NASA Ames Prognostics Data Repository (https://tiarcnasagov/tech/dash/groups/pcoe/ prognostic-data-repository/), NASA Ames Research Center, Moffett Field, CA2008;

[4] Gressling, Thorsten. "68 Production: predictive maintenance". Data Science in Chemistry: Artificial Intelligence, Big Data, Chemometrics, and Quantum Computing with Jupyter, Berlin, Boston: De Gruyter, 2020, pp. 342-346. https://doi.org/10.1515/9783110629453-068

[5] Samuel, A.L. Some studies in machine learning using the game of checkers. IBM J. Res. Dev. 2000, 44, 206–226.

[6] Carvalho, T.P.; Soares, F.A.; Vita, R.; Francisco, R.D.; Basto, J.P.; Alcalá, S.G. A systematic literature review of machine learning methods applied to predictive maintenance. Comput. Ind. Eng. 2019, 137, 106024

[7] Wuest, T.; Weimer, D.; Irgens, C.; Thoben, K.D. Machine learning in manufacturing: advantages, challenges, and applications. Prod. Manuf. Res. 2016, 4, 23–45.

[8] Wan, J.; Tang, S.; Li, D.; Wang, S.; Liu, C.; Abbas, H.; Vasilakos, A.V. A Manufacturing Big Data Solution for Active Preventive Maintenance. IEEE Trans. Ind. Inform. 2017, 13, 2039–2047.

[9] Jezzini, A.; Ayache, M.; Elkhansa, L.; Makki, B.; Zein, M. Effects of predictive maintenance(PdM), Proactive maintenance(PoM) & Preventive maintenance(PM) on minimizing the faults in medical instruments. In Proceedings of the 2013 2nd International Conference on Advances in Biomedical Engineering, Tripoli, Lebanon, 11–13 September 2013; pp. 53–56.

[10] Umar, A.; Chinnam, R.B.; Tseng, F. An HMM and polynomial regression-based approach for remaining useful life and health state estimation of cutting tools. Comput. Ind. Eng. 2019, 128, 1008–1014.

[11]Falamarzi, A.; Moridpour, S.; Nazem, M.; Cheraghi, S. Prediction of tram track gauge deviation using artificial neural network and support vector regression. Aust. J. Civ. Eng. 2019, 17, 63–71.

[12] Breiman, L. Random Forests; Working Paper CISL: Cambridge, UK, 2001; Volume 15, pp. 1–19. ISBN 9783110941975.

[13]Binding, A.; Dykeman, N.; Pang, S. Machine Learning Predictive Maintenance on Data in the Wild. In Proceedings of the 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), Limerick, Ireland, 15–18 April 2019; pp. 507–512

[14]Janssens, O.; Van De Walle, R.; Loccufier, M.; Van Hoecke, S. Deep Learning for Infrared Thermal Image Based Machine Health Monitoring. IEEE/ASME Trans. Mechatron. 2018, 23, 151–159.