**Project No: 6**

**Project Title: Maintenance Prediction**

**Prepared by: Rose Mary Jose**

**Date: 01-08-23**

Prepared for “Thoughtware Training Private Limited”, under Guidance of its CEO Mr. Pattabhi Raman. The project will be subject to further research, modification and exclusive use of “Thoughtware Training Private Limited”

**Objective of the project**

Maintenance prediction is a project aiming to develop a sophisticated system that forecasts equipment, machinery, or infrastructure maintenance needs. This system uses historical data, sensor readings, and advanced analytics to anticipate potential failures and proactively schedule maintenance, reducing downtime and minimizing operational disruptions.

Maintenance prediction has the potential to revolutionize maintenance practices, as traditional reactive approaches often result in costly repairs, idle production lines, and customer dissatisfaction. By analyzing equipment behavior, organizations can gain a clearer understanding of equipment behavior, identifying patterns indicating impending failures, and making informed decisions on scheduling maintenance. Benefits include improved operational efficiency, extended asset lifespan, reduced maintenance costs, and enhanced safety for workers. Maintenance prediction has widespread business applications in manufacturing, transportation, and energy, fostering a culture of proactive maintenance and aligning with the modern era's focus on data-driven strategies and operational excellence.

**Dataset**

Source - <https://data.nasa.gov/Aerospace/CMAPSS-Jet-Engine-Simulated-Data/ff5v-kuh6>

CMAPSS Jet Engine Simulated Data is the dataset that is being examined for the maintenance prediction. Data sets consists of multiple multivariate time series. Each data set is further divided into training and test subsets. Each time series is from a different engine. For this project dataset FD003 is considered which consist of TrainFD003, TestFD003 and RUL value of the 100 engines considered. The dataset consists of the following columns:

id: Engine unit number

cycle: Time, in cycles

op1, op2, op3: three operational settings that have a substantial effect on engine performance.

sensor 1 – sensor 23: 23 sensor readings. No information regarding sensors have been given. If there has been some information regarding sensor type i.e., pressure sensor, temperature sensor, vibration sensor etc. then we could have fetched some more information about degradation of engine using domain knowledge.

The dataset considered as train data is given below.

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
%matplotlib inline  
import seaborn as sns  
import io

columns=["id","cycle","op1","op2","op3","sensor1","sensor2","sensor3","sensor4","sensor5","sensor6","sensor7","sensor8",  
 "sensor9","sensor10","sensor11","sensor12","sensor13","sensor14","sensor15","sensor16","sensor17","sensor18","sensor19"  
 ,"sensor20","sensor21","sensor22","sensor23"]

train=pd.read\_csv("train\_FD003.csv",names = columns)  
test=pd.read\_csv("test\_FD003.csv",names=columns)  
test\_results=pd.read\_csv("RUL\_FD003.csv",header=None)

train.head()

id cycle op1 op2 op3 sensor1 sensor2 sensor3 sensor4   
0 1 1 -0.0005 0.0004 100 518.67 642.36 1583.23 1396.84   
1 1 2 0.0008 -0.0003 100 518.67 642.50 1584.69 1396.89   
2 1 3 -0.0014 -0.0002 100 518.67 642.18 1582.35 1405.61   
3 1 4 -0.0020 0.0001 100 518.67 642.92 1585.61 1392.27   
4 1 5 0.0016 0.0000 100 518.67 641.68 1588.63 1397.65   
sensor5 ... sensor14 sensor15 sensor16 sensor17 sensor18 sensor19   
14.62 ... 8145.32 8.4246 0.03 391.0 2388.0 100.0   
14.62 ... 8152.85 8.4403 0.03 392.0 2388.0 100.0   
14.62 ... 8150.17 8.3901 0.03 391.0 2388.0 100.0   
14.62 ... 8146.56 8.3878 0.03 392.0 2388.0 100.0   
14.62 ... 8147.80 8.3869 0.03 392.0 2388.0 100.0

In this dataset the goal is to predict the remaining useful life (RUL) of each engine in the test dataset. RUL is equivalent of number of flights remained for the engine after the last datapoint in the test dataset. So, the target variable isn’t given in the train data. Since the dataset considered consists of 100 unique unit number for engine the RUL value is given as another data frame. Data transformation is done in order to create the target variable column.

**EDA**

The shape of a dataset, accessed using the ‘shape’ attribute in pandas or NumPy, provides insights into the structure of the data in terms of rows and columns. It's a fundamental piece of information that helps you understand the dimensions and size of your dataset.

Shape of the train data is **24720 rows and 28 columns.**

The dataset's info returns details about the data frame. It contains information on the total number of columns, column labels, datatypes, the number of columns that aren't null, memory utilization, and range index. The information makes it obvious that the dataset under consideration consists of 6 features of dtype int64, and 22 features of float64. Missing value’s presence can be detected from the non-null count.

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 19535 entries, 0 to 19534  
Data columns (total 28 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 id 19535 non-null int64   
 1 cycle 19535 non-null int64   
 2 op1 19535 non-null float64  
 3 op2 19535 non-null float64  
 4 op3 19535 non-null int64   
 5 sensor1 19535 non-null float64  
 6 sensor2 19535 non-null float64  
 7 sensor3 19534 non-null float64  
 8 sensor4 19534 non-null float64  
 9 sensor5 19534 non-null float64  
 10 sensor6 19534 non-null float64  
 11 sensor7 19534 non-null float64  
 12 sensor8 19534 non-null float64  
 13 sensor9 19534 non-null float64  
 14 sensor10 19534 non-null float64  
 15 sensor11 19534 non-null float64  
 16 sensor12 19534 non-null float64  
 17 sensor13 19534 non-null float64  
 18 sensor14 19534 non-null float64  
 19 sensor15 19534 non-null float64  
 20 sensor16 19534 non-null float64  
 21 sensor17 19534 non-null float64

24 sensor20 19534 non-null float64  
 25 sensor21 19534 non-null float64  
 26 sensor22 0 non-null float64  
 27 sensor23 0 non-null float64  
dtypes: float64(25), int64(3)  
memory usage: 4.2 MB

To obtain the dataset's summary statistics, describe function is used. The Describe function of the numerical column provides an overview of the central tendency, dispersion, and distributional form of the dataset. The 25th, 50th, and 75th percentile values are returned, together with the count, mean, standard deviation, minimum value, and maximum value. From the descriptive statistics it is observable that columns such as op3, sensor1, sensor5, sensor6, sensor10, sensor16, sensor18, sensor19 have same value so those columns can be dropped. Missing values can be detected from the count of the descriptive statistics. By comparing the mean and median almost all columns follow a normal distribution.

count mean std min 25% \  
id 19535.0 37.707653 2.260570e+01 1.0000 18.000000   
cycle 19535.0 138.985155 9.870855e+01 1.0000 62.000000   
op1 19535.0 -0.000031 2.189851e-03 -0.0086 -0.001500   
op2 19535.0 0.000010 2.945215e-04 -0.0006 -0.000200   
op3 19535.0 100.000000 0.000000e+00 100.0000 100.000000   
sensor1 19535.0 518.670000 0.000000e+00 518.6700 518.670000   
sensor2 19535.0 642.452298 5.261531e-01 640.8400 642.070000   
sensor3 19534.0 1587.965676 6.886438e+00 1564.3000 1583.080000   
sensor4 19534.0 1404.361703 9.889647e+00 1377.0600 1396.930000   
sensor5 19534.0 14.620000 5.329207e-15 14.6200 14.620000   
sensor6 19534.0 21.595881 1.895008e-02 21.4500 21.580000   
sensor7 19534.0 554.973003 3.395125e+00 549.6100 553.010000   
sensor8 19534.0 2388.064125 1.658582e-01 2386.9000 2388.000000   
sensor9 19534.0 9063.607761 2.058838e+01 9017.9800 9050.880000   
sensor10 19534.0 1.301147 3.386339e-03 1.2900 1.300000   
sensor11 19534.0 47.411179 3.026905e-01 46.6900 47.180000   
sensor12 19534.0 522.888120 3.213649e+00 517.7700 521.070000   
sensor13 19534.0 2388.064301 1.655494e-01 2386.9300 2388.000000   
sensor14 19534.0 8143.794710 1.704738e+01 8099.6800 8133.750000   
sensor15 19534.0 8.398508 5.971477e-02 8.1563 8.364700   
sensor16 19534.0 0.030000 1.040861e-17 0.0300 0.030000   
sensor17 19534.0 392.529794 1.784191e+00 388.0000 391.000000   
sensor18 19534.0 2388.000000 0.000000e+00 2388.0000 2388.000000   
sensor19 19534.0 100.000000 0.000000e+00 100.0000 100.000000   
sensor20 19534.0 38.978636 2.466996e-01 38.1700 38.820000   
sensor21 19534.0 23.387338 1.476704e-01 22.8995 23.292325   
sensor22 0.0 NaN NaN NaN NaN   
sensor23 0.0 NaN NaN NaN NaN

50% 75% max   
id 36.0000 57.0000 79.0000   
cycle 125.0000 190.0000 525.0000   
op1 0.0000 0.0014 0.0086   
op2 0.0000 0.0003 0.0007   
op3 100.0000 100.0000 100.0000   
sensor1 518.6700 518.6700 518.6700   
sensor2 642.4000 642.7900 645.1100   
sensor3 1587.4100 1592.3575 1615.3900   
sensor4 1402.7900 1410.6200 1441.1600   
sensor5 14.6200 14.6200 14.6200   
sensor6 21.6100 21.6100 21.6100   
sensor7 553.9400 555.7600 570.4900   
sensor8 2388.0700 2388.1400 2388.6000   
sensor9 9059.5200 9069.8400 9234.3500   
sensor10 1.3000 1.3000 1.3200   
sensor11 47.3600 47.6000 48.4400   
sensor12 521.8800 523.5700 537.4000   
sensor13 2388.0600 2388.1400 2388.6100   
sensor14 8140.7300 8149.0400 8290.5500   
sensor15 8.4010 8.4381 8.5705   
sensor16 0.0300 0.0300 0.0300   
sensor17 392.0000 394.0000 399.0000   
sensor18 2388.0000 2388.0000 2388.0000   
sensor19 100.0000 100.0000 100.0000   
sensor20 38.9700 39.1300 39.8400   
sensor21 23.3842 23.4750 23.9505   
sensor22 NaN NaN NaN   
sensor23 NaN NaN NaN

**Data Preparation**

**Detecting and handling missing values**: To ensure accuracy, reliability, and robustness of analyses and models, data preparation is essential for detecting and handling missing values and outliers. Due to the fact that missing values result from errors or inadequate recording and outliers represent anomalies or exceptional circumstances, these problems may result in biased conclusions and incorrect predictions. For the sake of maintaining data integrity and arriving with suitable conclusion, these issues must be regularly addressed.

For detecting the missing data is to use Python functions like isnull() and sum(). The isnull().sum() function helps to quickly figure out the amount of data missing from each column. In this step, one can observe which columns have missing values and deal with them. In the dataset considered there is presence of missing value and it is observed that two columns sensor 22 and sensor 23 has any values. So those two columns can be dropped.

train.isnull().sum()

id 0  
cycle 0  
op1 0  
op2 0  
op3 0  
sensor1 0  
sensor2 0  
sensor3 1  
sensor4 1  
sensor5 1  
sensor6 1  
sensor7 1  
sensor8 1  
sensor9 1  
sensor10 1  
sensor11 1  
sensor12 1  
sensor13 1  
sensor14 1  
sensor15 1  
sensor16 1  
sensor17 1  
sensor18 1  
sensor19 1  
sensor20 1  
sensor21 1  
sensor22 19535  
sensor23 19535  
dtype: int64

train = train.drop(['sensor22','sensor23'], axis = 1)

train = train.dropna(axis = 0,how ='any')  
train.isnull().sum()

**Detecting and handling duplicated values:** Duplicated values are identical or repeated entries in a dataset, resulting from errors, system glitches, or unintentional repetitions. Identifying and handling duplicates is crucial in data preprocessing to maintain integrity and ensure accurate analyses. Functions like duplicated() in Python can detect duplicates and mark subsequent occurrences as duplicates. Here in the considered dataset no duplicated values are present.

train.duplicated().sum()

0

**Creating the target variable ‘RUL’:** The test\_result data frame is of shape (100,1) it contains a column called rul. A new column named "id" is added to the test\_results data frame with values ranging from 1 to the total number of rows in the data frame. Then the train data frame is grouped by the "id" column and calculates the maximum value of the "cycle" column for each group. the results are stored in a data frame called 'rul' with columns "id" and "max". A new column "rul\_failed" is added to the test\_results data frame. This contains the sum of the "rul" value and the corresponding "max" value from the rul data frame. Then the test\_results data frame is merged into train data frame based on the "id" column. Later a new column "RUL" is created in the train data frame, calculated as the difference between the "rul\_failed" and the "cycle" value for each record. This calculation represents the actual Remaining Useful Life for each engine. Then the column "rul" is dropped from the train data frame and the resulting data frame is named as df\_train.

test\_results.columns=["rul"]  
test\_results['id']=test\_results.index+1  
test\_results.head()

rul id  
0 44 1  
1 51 2  
2 27 3  
3 120 4  
4 101 5

rul = pd.DataFrame(train.groupby('id')['cycle'].max()).reset\_index()  
rul.columns = ['id', 'max']  
test\_results['rul\_failed']=test\_results['rul']+rul['max']  
test\_results

rul id rul\_failed  
0 44 1 303.0  
1 51 2 304.0  
2 27 3 249.0  
3 120 4 392.0  
4 101 5 314.0

train=train.merge(test\_results,on=['id'],how='left')  
train["RUL"]=train["rul\_failed"]-train["cycle"]

df\_train = train.drop(['rul'],axis = 1)

df\_train.head()

id cycle op1 op2 op3 sensor1 sensor2 sensor3 sensor4 \  
0 1 1 -0.0005 0.0004 100 518.67 642.36 1583.23 1396.84   
1 1 2 0.0008 -0.0003 100 518.67 642.50 1584.69 1396.89   
2 1 3 -0.0014 -0.0002 100 518.67 642.18 1582.35 1405.61   
3 1 4 -0.0020 0.0001 100 518.67 642.92 1585.61 1392.27   
4 1 5 0.0016 0.0000 100 518.67 641.68 1588.63 1397.65   
  
 sensor5 ... sensor14 sensor15 sensor16 sensor17 sensor18 sensor19   
0 14.62 ... 8145.32 8.4246 0.03 391.0 2388.0 100.0   
1 14.62 ... 8152.85 8.4403 0.03 392.0 2388.0 100.0   
2 14.62 ... 8150.17 8.3901 0.03 391.0 2388.0 100.0   
3 14.62 ... 8146.56 8.3878 0.03 392.0 2388.0 100.0   
4 14.62 ... 8147.80 8.3869 0.03 392.0 2388.0 100.0   
  
 sensor20 sensor21 rul\_failed RUL   
0 39.11 23.3537 303.0 302.0   
1 38.99 23.4491 303.0 301.0   
2 38.85 23.3669 303.0 300.0   
3 38.96 23.2951 303.0 299.0   
4 39.14 23.4583 303.0 298.0

The above steps transform the data and calculate the remaining useful life (RUL) of engines based on the rul value provided in the test\_results data frame.

cycle=30  
df\_train['label'] = df\_train['RUL'].apply(lambda x: 1 if x <= cycle

else 0)

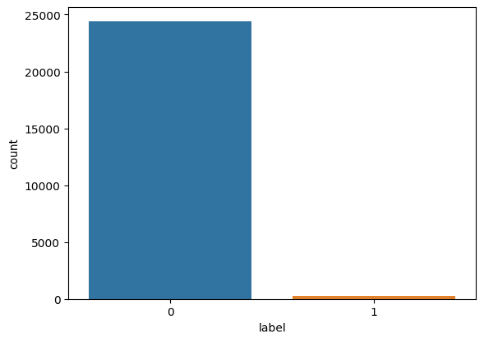
op\_set=["op"+str(i) for i in range(1,4)]  
sensor=["sensor"+str(i) for i in range(1,22)]

Now a new column is created as "label" in the data frame df\_train which takes the binary value. A threshold value is kept as 30 for the "RUL" column, if the RUL is less than or equal to 30, then 1 else 0 is given for the label column. This is the target variable for the classification. Basically, what the classification do is it will predict as 1 when the RUL value of an engine is less than or equal to 30, so the maintenance needs to be done as soon as possible.

df\_train['label'].value\_counts()

0 19368  
1 166  
Name: label, dtype: int64

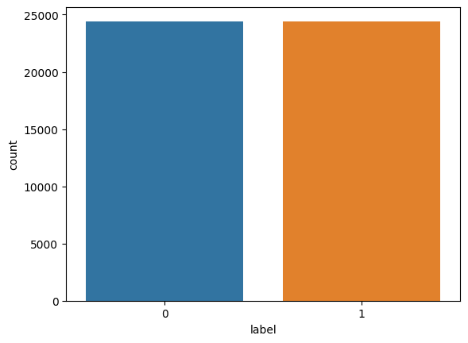
**Dealing with imbalanced data**: Imbalanced data occurs when classes in a dataset have unequal representation. To address this, techniques include oversampling the minority class,



from imblearn.over\_sampling import RandomOverSampler  
ros = RandomOverSampler(random\_state=42)  
X, y = ros.fit\_resample(X, y)  
# Create a new dataframe with resampled data  
df\_train= pd.concat([pd.DataFrame(X), pd.Series(y, name='label')], axis=1)  
df\_train['label'].value\_counts()

0 19368  
1 19368  
Name: label, dtype: int64

under sampling the majority class, and using algorithms that adjust class weights. Such approaches help models learn from both classes, improving performance on underrepresented ones. In order to balance the data RandomOverSampler is used from imblearn library of python. After balancing the label column value count is given as:



**Model Selection**

Model selection is the process of choosing the most appropriate machine learning algorithm or model from a set of candidates to solve a specific problem. It involves evaluating different models based on their performance metrics and selecting the one that best fits the data and the task at hand.

In this project, our objective is to predict the Remaining Useful Life (RUL) value for engines. Given that we are dealing with a predictive task, a supervised learning approach is suitable. We have chosen to employ a range of classification models, including ***Logistic Regression, Decision tree classifier, and Random Forest classifier***. These models are well-suited for classification tasks where we aim to predict a binary value for the variable label.

Logistic Regression: Logistic Regression is a binary classification algorithm that estimates the probability of a binary outcome. It uses a logistic function to model the relationship between features and the probability of the outcome, making it useful for problems like spam detection.

Decision Tree Classifier: Decision Tree Classifier is a supervised machine learning model that learns a tree-like structure to make decisions. It splits data based on features, aiming to maximize information gain or Gini impurity. It's interpretable but can overfit.

Random Forest Classifier: Random Forest Classifier is an ensemble learning algorithm used for classification tasks. It combines multiple decision trees to make predictions. Each tree is built on a subset of the data and features, reducing overfitting. The final prediction is determined by a majority vote of individual tree predictions. It's effective, handles complex relationships, and offers good generalization.

**Model Evaluation methods**

Model evaluation techniques are essential processes to assess the performance of machine learning models. They provide insights into how well a model generalizes to new, unseen data and aids in selecting the best model for a given problem. Evaluation techniques, such as accuracy, precision, recall, F1-score, and ROC curves for classification tasks, and metrics like RMSE and MAE for regression tasks, help quantify the model's predictive capabilities. These techniques help make informed decisions about model selection, hyperparameter tuning, and identifying areas for improvement in model performance.

In classification problems, evaluation metrics measure the performance of a model's predictions. Common metrics include accuracy, which measures overall correctness, and precision, recall, and F1-score, which provide insights into the model's ability to identify and correctly classify positive instances, considering false positives and false negatives.

Confusion Matrix: A confusion matrix is a table that summarizes the performance of a classification model. It shows the true positive, true negative, false positive, and false negative values. From this matrix, various metrics like precision, recall, and F1-score can be derived, providing a more comprehensive evaluation of model performance.

Accuracy Score: Accuracy is a common evaluation metric for classification models. It calculates the ratio of correctly predicted instances to the total instances in the dataset. However, it can be misleading if classes are imbalanced.

**Split data into Train and Test**

Splitting a dataset into three subsets—training, validation, and testing—is a fundamental practice in machine learning to assess model performance, prevent overfitting, and ensure generalization. In this scenario, an 80:20 split using the train\_test\_split function creates distinct training and testing sets. For the validation dataset, the test data frame is considered. This enables models to learn patterns from the training set, tune hyperparameters on the validation set, and ultimately evaluate performance on the untouched testing set. Such a split ensures a balanced approach to model development, refinement, and assessment, contributing to robust and accurate predictions in the delivery time prediction context.

from sklearn.model\_selection import train\_test\_split  
X=df\_train.drop(["id","cycle","op3","sensor1","sensor5","sensor6","sensor10","sensor16","sensor18","sensor19","RUL",'label'],axis=1)  
y=df\_train['label']

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X,y, test\_size = 0.20, random\_state = 42)

**Model building**

The train data is fitted with the selected three models from the Sklearn library and classification into 0 and 1 is done for both training and testing datasets and their respective confusion matrix and accuracy score are calculated using the confusion\_matrix and accuracy\_score functions from the Sklearn library. The confusion matrix and accuracy score of train and test data of three models are given below.

1. Logistic Regression:

Train data

confusion\_matrix(Y\_train,train\_pred)

accuracy\_score(Y\_train,train\_pred)

array([[13985, 1515],  
 [ 911, 14577]])

0.9217116303085066

Test data

confusion\_matrix(Y\_test,test\_pred)

accuracy\_score(Y\_test,test\_pred)

array([[3485, 383],  
 [ 230, 3650]])

0.9208828084667011

1. Decision tree Classifier:

Train data

confusion\_matrix(Y\_train,dtc\_train\_pred)

accuracy\_score(Y\_train,dtc\_train\_pred)

array([[15500, 0],  
 [ 0, 15488]])

1.0

Test data

confusion\_matrix(Y\_test,dtc\_test\_pred)

accuracy\_score(Y\_test,dtc\_test\_pred)

array([[3859, 9],  
 [ 0, 3880]])

0.9988384099122354

1. Random Forest Classifier:

Train data

confusion\_matrix(Y\_train,rfc\_train\_pred)

accuracy\_score(Y\_train,rfc\_train\_pred)

array([[15500, 0],  
 [ 0, 15488]])

1.0

Test data

confusion\_matrix(Y\_test,rfc\_test\_pred)

accuracy\_score(Y\_test,rfc\_test\_pred)

array([[3863, 5],  
 [ 0, 3880]])

0.9993546721734641

**Model’s Summary**

The Decision Tree and Logistic Regression models show higher False Negative counts, indicating that they are missing a significant number of positive cases. While Decision Tree overfits to the training data (with a True Positive of 19636 and a False Negative of 0), it doesn't generalize well to the test data (with a True Positive of 4773 and a False Negative of 20). Logistic Regression also has a high False Negative count (1636 in training and 390 in testing). Random Forest performs better by aggregating multiple decision trees, reducing overfitting, and providing a better balance between precision and recall. This makes Random Forest more robust and reliable in handling imbalanced datasets and generalizing well to new data. But for validating model three models are used.

**Model Validation**

Similar to train data frame test data frame also has the same columns.

op1 op2 sensor2 sensor3 sensor4 sensor7 sensor8 sensor9   
0 -0.0017 -0.0004 641.94 1581.93 1396.93 554.56 2387.93 9048.65   
1 0.0006 -0.0002 642.02 1584.86 1398.90 554.10 2387.94 9046.53   
2 0.0014 -0.0003 641.68 1581.78 1391.92 554.41 2387.97 9054.92   
3 0.0027 0.0001 642.20 1584.53 1395.34 554.58 2387.94 9055.04   
4 -0.0001 0.0001 642.46 1589.03 1395.86 554.16 2388.01 9048.59

sensor11 sensor12 sensor13 sensor14 sensor15 sensor17 sensor20   
0 47.09 521.89 2387.94 8133.48 8.3760 391 39.07   
1 47.08 521.85 2388.01 8137.44 8.4062 391 39.04   
2 47.15 522.10 2387.94 8138.25 8.3553 391 39.10   
3 47.26 522.45 2387.96 8137.07 8.3709 392 38.97   
4 46.94 521.91 2387.97 8134.20 8.4146 391 39.09

sensor21 rul\_failed RUL label   
0 23.4468 277 276 0   
1 23.4807 277 275 0   
2 23.4244 277 274 0   
3 23.4782 277 273 0   
4 23.3950 277 272 0   
[16596 rows x 19 columns]

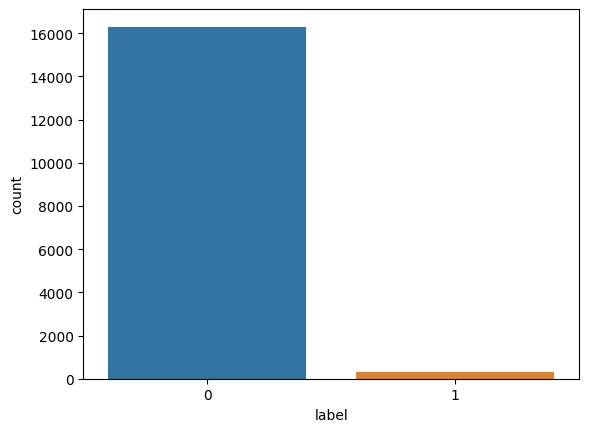
The test data frame has 16596 rows and 28 columns.

The test data frame has 22 columns of float dtype and 6 of int dtype. Similar preprocessing steps are done on test data frame as train data frame. From rul data frame the max of each engine value is found and the rul\_failed is calculated from the rul and max columns of test\_result data frame. The test\_results is merged with the test data frame with “id” column and from the difference of cycle and rul\_failed RUL is calculated. Columns such as id, cycle, op3, sensor1, sensor5, sensor6, sensor10, sensor16, sensor18, sensor19, sensor22, sensor23 are dropped from test data frame and the data frame is renamed as df\_test. A new column label is created which gives binary value 1 if the RUL is less than or equal to 30, else 0 is given. Thus, the target column is created. Since that column is imbalanced balancing is done using the RandomOverSampling function from imblearn library from python.

test\_X = df\_test.drop(['label','RUL'],axis = 1)  
test\_y = df\_test['label']

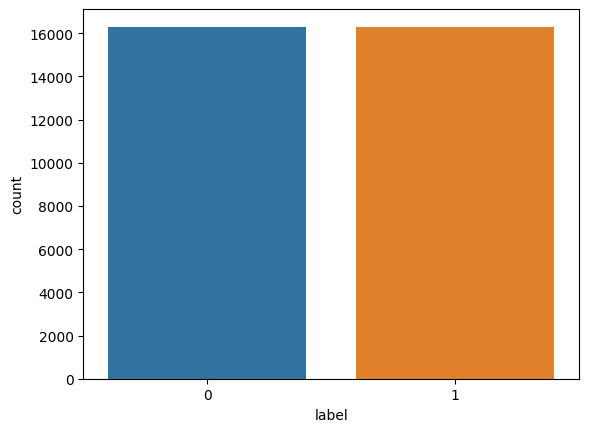
df\_test['label'].value\_counts()

0 16305  
1 291  
Name: label, dtype: int64



from imblearn.over\_sampling import RandomOverSampler  
ros = RandomOverSampler(random\_state=42)  
test\_X, test\_y = ros.fit\_resample(test\_X, test\_y)  
# Create a new dataframe with resampled data  
df\_test= pd.concat([pd.DataFrame(test\_X), pd.Series(test\_y, name='label')], axis=1)  
df\_test['label'].value\_counts()

0 16305  
1 16305  
Name: label, dtype: int64



The target and feature variables are separated and the prediction is done using logistic regression, decision tree and random forest classifier. The confusion matrix and accuracy score of the three models is given below.

Logistic regression:

confusion\_matrix(test\_y,pred1)

accuracy\_score(test\_y,pred1)

array([[15423, 882],  
 [ 501, 15804]])

0.9575896964121435

Decision tree classifier:

confusion\_matrix(test\_y,pred2)

accuracy\_score(test\_y,pred2)

array([[16303, 2],  
 [13011, 3294]])

0.600950628641521

Random Forest Classifier:

confusion\_matrix(test\_y,pred3)

accuracy\_score(test\_y,pred3)

array([[16305, 0],  
 [14455, 1850]])

0.5567310640907697

Considering the models' performance on this validation dataset, Logistic Regression appears to have a slightly better balance between True Positives and True Negatives compared to the other models. However, the imbalanced False Negative count in all models suggests room for improvement in sensitivity/recall. Further fine-tuning and perhaps ensemble methods could be explored for better results.

dff = pd.DataFrame({'actual':test\_y,'predicted':pred1})  
dff = dff[dff['actual']==1]  
dff.head()

actual predicted  
587 1 1  
588 1 1  
589 1 1  
590 1 1  
3347 1 1

**Conclusion**

The study aimed to develop effective classification models for predictive maintenance, focusing on identifying impending equipment failures. Three popular algorithms - Logistic Regression, Decision Tree, and Random Forest - were evaluated for training and testing on a dataset. The Random Forest model showed the highest accuracy during training and testing, identifying both positive and negative instances. Its ensemble nature, combining multiple decision trees, contributed to its robustness and adaptability to complex patterns. However, in the validation phase, Logistic Regression outperformed the others, showing a more balanced performance in detecting both classes, making it a safer option when false negatives can have significant consequences.

In conclusion, Logistic Regression emerged as the better choice for the validation dataset due to its complex structure and ensemble nature. This highlights the importance of thorough model evaluation on independent datasets to ensure generalizability. In predictive maintenance, avoiding false negatives is crucial to prevent costly failures, Logistic Regression's consistent performance makes it the recommended model. Further exploration of model hyperparameters and ensemble techniques could potentially enhance the performance of both models in real-world scenarios.

**References**

<https://www.kaggle.com/code/darkside92/nasa-turbofan-engine-rul-predictive-maintenance>