Master of Science Thesis

Artificial Intelligence Methods for Predictive Maintenance

Student: Dimitrios Koikas
Registration Number: AIDL-0021

MSc Thesis Supervisor

Grigorios Nikolaou
Lecturer

ATHENS-EGALEO, February 2024

This MSc Thesis has been accepted, evaluated and graded by the following committee:

Supervisor	Member	Member
Grigorios Nikolaou Digitally signed by Grigorios Nikolaou Date: 2024.07.26 11:41:51 +03'00'	DIMITRIOS KANTZOS Digitally signed by DIMITRIOS KANTZOS Date: 2024.07.25 21:44:08 +03'00'	ELENI AIKATERINI LELIGKOU Digitally signed by ELENI AIKATERINI LELIGKOU Date: 2024.07.25 12:15:01 +03'00'
Nikolaou Grigorios	Cantzos Demetrios	Leligkou Aikaterini-Eleni
Dept. of Industrial Design and Production Engineering	Dept. of Industrial Design and Production Engineering	Dept. of Industrial Design and Production Engineering
University of West Attica	University of West Attica	University of West Attica

Declaration of the author of this MSc thesis

I, Dimitrios Koikas Panagiotis with the following student registration number: AIDL-0021, postgraduate student of the MSc programme in "Artificial Intelligence and Deep Learning", which is organized by the Department of Electrical and Electronic Engineering and the Department of Industrial Design and Production Engineering of the Faculty of Engineering of the University of West Attica, hereby declare that:
I am the author of this MSc thesis andany help I may have received is clearly mentioned in the thesis. Additionally, all the sources I have used (e.g., to extract data, ideas, words or phrases) are cited with full reference to the corresponding authors, the publishing house or the journal; this also applies to the Internet sources that I have used. Ialsoconfirmthatlhave personally written this thesis and the intellectual property rights belong to myself and to the University of West Attica. This work has not been submitted for any other degree or professional qualification except as specifiedin it.
Any violations of my academic responsibilities, as stated above,constitutessubstantial reason for the cancellation of the conferred MSc degree.

The author
Dimitrios Koikas

[illegible]

(Signature)

I knew exactly what to do but, in a much more real sense, I had no idea what to do.

-Michael Scott

[0]Front page image: https://www.bitdeal.net/ai-in-predictive-maintenance

ABSTRACT

The rapid evolution of Artificial Intelligence over the past few years has revolutionised applications in several fields. Such field is the Predictive Maintenance (PdM) concept. In contrast to reactive or scheduled maintenance, it aims to **predict** when machinery or infrastructure is likely to fail. This approach is proven to be cost-efficient, preventing unplanned downtime and optimising operational efficiency.

Therefore, it was decided to explore this interesting concept using Machine Learning algorithms on a benchmark dataset.

Two problems were approached:

- Regression problem - Predict Remaining Useful Life (RUL)
- Classification problem - Classify the condition of the machine (No fault or type of fault)

The process began with performing an Exploratory Data Analysis (EDA) on the provided dataset. After the necessary manipulation of the data and graphical evaluation, new Features were engineered to serve the purpose of the tasks. Several Models were tested in order to select the best performing one.

All of the above will be analysed in the following chapters.

Keywords: Predictive Maintenance, Remaining Useful Life, Fault Classification

Table of Contents

List of Figures

List of Tables

INTRODUCTION

Predictive Maintenance (PdM) is a proactive approach used by several organisations to maintain their machinery and equipment in optimal condition. Unlike traditional maintenance practices, that rely on specific schedules or reactive repairs after a breakdown, Predictive Maintenance uses operational data to predict when equipment is likely to fail. By doing so, preventive actions can be taken before failures occur, leading to minimised downtime, reduced costs, and optimised productivity.

The concept of Predictive Maintenance revolves around the idea of leveraging data from sensors, Internet of Things (IoT) devices, and other sources to monitor the health and performance of machinery in real-time. This data is then analysed using analytics techniques such as Machine Learning algorithms to identify patterns, trends, and anomalies that may indicate issues or failures in the future.

One of the key benefits (and aims) of this concept is its ability to provide early warnings about equipment degradation or failure. By detecting subtle changes in performance metrics or signs of wear, maintenance teams can intervene proactively to resolve issues before they escalate into major problems. This not only helps in extending the lifespan of equipment, but also ensures smooth operation and minimises the risk of unexpected downtime, which can be costly for businesses.

Subject

This Thesis will explore the application of such concept in the Transportation sector. Machinery faults occurring in public transportation vehicles during their routine operations result in various - direct and indirect - problems, especially when they lead to trip interruptions. These unpredictable effects extend beyond just the operator company, affecting customers who rely on transportation services for their daily needs and responsibilities. Therefore, timely detection of such faults can prevent trip cancellations and highlight (or prevent) the need to take vehicles out of service.

It is easy to understand the importance and value maintenance brings to the table. Hundreds of trip cancellations occur in Europe every year. To put things into perspective, 170 trips were canceled in Portugal during 2017, as mentioned by the authors of the dataset used in this Thesis.

Aim and Objectives

The aim of this Thesis is to explore Predictive Maintenance techniques with Machine Learning Models to develop and evaluate methodologies for estimating Remaining Useful Life (RUL) and classifying the state of industrial equipment. By definition, the main objective is to enhance asset reliability and minimise downtime by proactively identifying potential failures and optimising maintenance activities. To achieve this, two (2) research questions are set to be investigated.

Firstly, it seeks to assess the effectiveness of different Machine Learning algorithms in accurately predicting the RUL of equipment based on historical operational data and sensor measurements. This involves exploring the performance of Regression models in capturing the degradation patterns of equipment over time.

Secondly, the study aims to investigate state Classification methods to categorise equipment into various operational states, such as Normal or different Fault states. This objective involves evaluating the capability of classification algorithms, including Decision Trees, Ensemble techniques and Neural Networks, in distinguishing between different equipment conditions based on feature extraction available data.

Furthermore, the study seeks to determine the optimal features and feature engineering techniques for improving the accuracy and robustness of the classification models. By addressing these research questions, the study aims to provide insights into the selection and implementation of predictive maintenance strategies that leverage Machine Learning models to optimise asset management practices and enhance operational efficiency in industrial settings.

Methodology

The Methodology is split in the following parts:

- Dataset Exploration and Processing: Historical operational data and sensor measurements from the equipment under examination are analysed, modifications are made if necessary and conclusions are extracted.
- Feature Engineering: In this step, relevant features that capture the degradation patterns and operational states of the equipment are selected from the preprocessed data. New features are created to serve as target variables for the Regression and Classification models.
- Model Development for RUL Estimation: Regression models are trained on the feature-engineered dataset to predict the Remaining Useful Life of the equipment. Various algorithms - including but not limited to - Linear Regression, Decision Trees and Gradient Boosting, will be evaluated to identify the most accurate and reliable model for RUL estimation.
- Model Development for State Classification: Another set of Machine Learning models, such as Decision Trees or Deep Learning classifiers, are developed to classify the operational states of the machinery. These models are trained on the feature-engineered dataset to differentiate between Normal and Fault states based on the extracted features.
- Model Evaluation: For each problem (Regression or Classification), the performance of the developed models is evaluated using appropriate metrics, such as Mean Absolute Error (MAE) or Root Mean Squared Error (RMSE) for RUL estimation, and Accuracy, Precision, and Recall for state Classification. The models are validated using part of the dataset as Test data to evaluate their generalisation capabilities.

Structure

In Chapter 1: THE DATASET, the dataset used in this Thesis is presented and described.

In Chapter 2: EXPLORATORY DATA ANALYSIS, the dataset is explored and comparisons between Noraml and Fault states are being made with the help of plots.

In Chapter 3: FEATURE ENGINEERING, the dataset is manipulated according to the results of EDA and new features are created to be used in the models.

In Chapter 4: REMAINING USEFUL LIFE ESTIMATION, several models are developed and evaluated to address the Regression problem of RUL prediction.

In Chapter 5: STATE CLASSIFICATION, several models are developed and evaluated to address the Classification problem of machine state prediction.

In Chapter 6: CONCLUSIONS, the conclusions of this study are presented.

In Chapter 7: FUTURE WORK, some thoughts for possible expansion of this Thesis are presented.

Chapter 1

THE DATASET

The dataset is the result of a Predictive Maintenance project executed by a group of researchers from the University of Porto in collaboration with urban metro in the city of Porto, in Portugal. The dataset contains information gathered during 2022 (January to June) aiming to create Machine Learning techniques for identifying anomalies and predicting equipment failures in online systems.

The Data Acquisition System consists of analog and digital sensors. These sensors acquire real-time information such as pressure, temperature and current consumption along with discrete signals. Along with this, the metro vehicle was equipped with a GPS device, providing geographical latitude, longitude and speed.

On top of every metro vehicle, the Air Production Unit (APU) is installed. APU's role is to provide air in several units, performing different functions. The most critical of them is the secondary suspension unit. Responsible for ensuring the vehicle's height in desired level, irrespective of the number of passengers onboard, its importance is easy to understand. Therefore, APU's failure immediately sets the corresponding vehicle out of order and non-scheduled repair is needed.

The data are collected in 1-second intervals (1 Hz). In the following sections, the collected signal types will be cited and described.

1.1 Analog sensors

As mentioned earlier, the analog sensors measure pressure, temperature and electric current consumption at several APU components. Eight signals are collected.

Here is the description:

- TP2 - Pressure on the compressor (bar).
- TP3 - Pressure generated at the pneumatic panel (bar).
- H1 - Valve that is activated when the pressure read by the pressure switch of the command is above the operating pressure of 10.2 bar (bar).
- DV pressure - Pressure exerted due to pressure drop generated when air dryers towers discharge the water. When it is equal to zero, the compressor is working under load (bar).
- Reservoirs - Pressure inside the air tanks installed on the trains (bar).
- Oil_Temperature - Temperature of the oil present on the compressor (°C).
- Flowmeter - Airflow on the pneumatic control panel (m^3/h).

- Motor_Current - Motor's current, which should present the following values:
  - close to 0 A when the compressor turns off
  - close to 4 A when the compressor is working offloaded
  - close to 7 A when the compressor is operating under load (A)

1.2 Digital sensors

The digital signals are either 0 (zero) when inactive or 1 (one) when an event triggers them. Eight signals are collected as well.

Here is the description:

- COMP - Electrical signal of the air intake valve on the compressor. It is active when there is no admission of air on the compressor, meaning that the compressor turns off or working offloaded.
- DV electric - Electrical signal that commands the compressor outlet valve. When it is active, it means that the compressor is working under load; when it is not active, it means that the compressor is off or offloaded.
- Towers - Signal that defines which tower is drying the air and which tower is draining the humidity removed from the air. When it is not active, it means that tower one is working; when it is active, it means that tower two is working.
- MPG - Is responsible for activating the intake valve to start the compressor under load when the pressure in the APU is below 8.2 bar. Consequently, it will activate the sensor COMP, which assumes the same behaviour as the MPG sensor.
- LPS - Signal activated when the pressure is lower than 7 bars.
- Pressure_switch - Signal activated when pressure is detected on the pilot control valve.
- Oil_Level - The oil level on the compressor is active (equal to one) when the oil is below the expected values.
- Caudal_impulses - Signal produced by the flowmeter indicating the existence of the flow of air per second.

As for the GPS device, it returns the following information:

- gpsLong - Longitude position (°).
- gpsLat - Latitude position (°).
- gpsSpeed - Speed (km/h).
- gpsQuality - Signal quality.

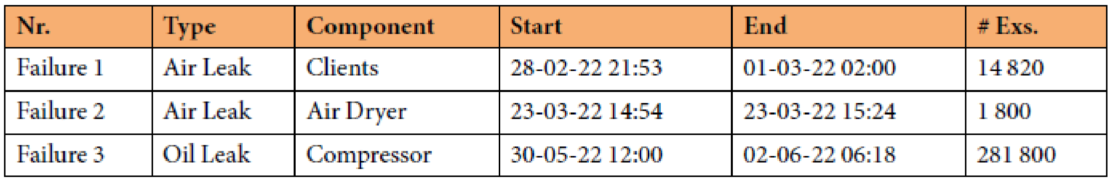
GPS signal is lost in case the train is crossing a tunnel and returns 0 (zero).

1.3 Failures

Three failures were detected in the 6-month span:

- Two Air Leak Faults
- One Oil Leak Fault

The maintenance team provided the timeframe of each failure occurrence along with the faulty component, as shown in the figure below:



Nr.	Type	Component	Start	End	# Exs.
Failure 1	Air Leak	Clients	28-02-22 21:53	01-03-22 02:00	14 820
Failure 2	Air Leak	Air Dryer	23-03-22 14:54	23-03-22 15:24	1 800
Failure 3	Oil Leak	Compressor	30-05-22 12:00	02-06-22 06:18	281 800

Figure 1.3.1: Failure occurrence[1]

This information helps us annotate the dataset. Annotated features will be used as target variables for the Machine Learning models' predictions. (see Chapter 3: FEATURE ENGINEERING)

[1]Source: Veloso, B., Gama, J., Ribeiro, R. & Pereira, P. MetroPT: A Benchmark dataset for predictive maintenance

Chapter 2

EXPLORATORY DATA ANALYSIS

Exploratory Data Analysis (EDA) is an essential initial step in the data analysis process that involves examining and understanding the structure, patterns, and characteristics of a dataset. Through EDA, engineers seek to gain insights into the underlying data to inform subsequent analysis and decision-making. One aspect of EDA involves summarizing the main characteristics of the data, such as its central tendency, dispersion, and distribution. Descriptive statistics, histograms, and box plots are commonly used to visualize these properties, providing an overview of the dataset's key features.

Moreover, EDA aims to identify relationships and patterns within the data, uncovering associations between variables and revealing trends or anomalies that may warrant further investigation. Techniques such as scatter plots, correlation matrices, and heatmaps are employed to visualize the relationships between variables and assess their strength and direction. By exploring these relationships, data hypotheses can be formed and guide subsequent analysis to better understand the underlying mechanisms driving the data.

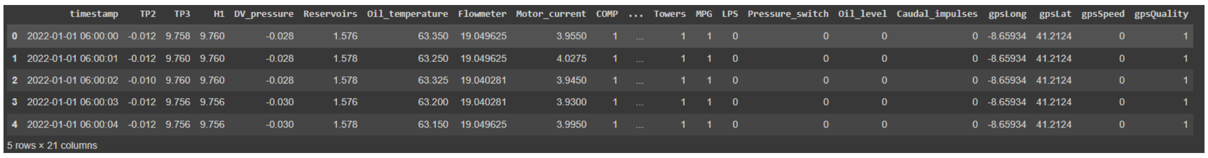
Additionally, EDA serves as a crucial step in data preprocessing, helping to detect and address missing values, outliers, and inconsistencies that may impact the quality and validity of subsequent analyses. Through data cleaning and preprocessing techniques, such as imputation, outlier detection, and feature scaling, analysts can ensure that the data is properly prepared for modeling and analysis. By thoroughly exploring and understanding the dataset through EDA, decisions can be made about which analytical techniques to apply and how to interpret the results accurately, ultimately leading to more robust and reliable insights.

In this chapter, the steps taken will be presented along with some short code snippets.

2.1 Getting to know the dataset

First of all, the dataset needs to be loaded and see it's ”head” and ”tail”. This is to make sure of the Timeframe, have a first look at dataset's structure and see its shape.

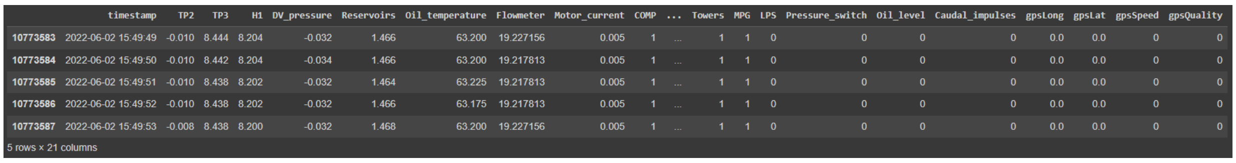
```
import pandas as pd
df_raw = pd.read_csv('dataset_train.csv',
  parse_dates=['timestamp'],
  infer_datetime_format=True,
)
df_raw.head()
df_raw.tail()

```



	timestamp	TP2	TP3	H1	DV_pressure	Reservoirs	Oil_temperature	Flowmeter	Motor_current	COMP	...	Towers	MPG	LPS	Pressure_switch	Oil_level	Caudal_impulses	gpsLong	gpsLat	gpsSpeed	gpsQuality
0	2022-01-01 06:00:00	-0.012	9.758	9.760	-0.028	1.576	63.350	19.049625	3.9550	1	...	1	1	0	0	0	0	-8.65934	41.2124	0	1
1	2022-01-01 06:00:01	-0.012	9.760	9.760	-0.028	1.578	63.250	19.049625	4.0275	1	...	1	1	0	0	0	0	-8.65934	41.2124	0	1
2	2022-01-01 06:00:02	-0.010	9.760	9.760	-0.028	1.578	63.325	19.040281	3.9450	1	...	1	1	0	0	0	0	-8.65934	41.2124	0	1
3	2022-01-01 06:00:03	-0.012	9.756	9.756	-0.030	1.576	63.200	19.040281	3.9300	1	...	1	1	0	0	0	0	-8.65934	41.2124	0	1
4	2022-01-01 06:00:04	-0.012	9.756	9.756	-0.030	1.578	63.150	19.049625	3.9950	1	...	1	1	0	0	0	0	-8.65934	41.2124	0	1

5 rows × 21 columns

Figure 2.1.1: Initial dataset head rows



	timestamp	TP2	TP3	H1	DV_pressure	Reservoirs	Oil_temperature	Flowmeter	Motor_current	COMP	...	Towers	MPG	LPS	Pressure_switch	Oil_level	Caudal_impulses	gpsLong	gpsLat	gpsSpeed	gpsQuality
10773583	2022-06-02 15:49:49	-0.010	8.444	8.204	-0.032	1.466	63.200	19.227156	0.005	1	...	1	1	0	0	0	0	0.0	0.0	0	0
10773584	2022-06-02 15:49:50	-0.010	8.442	8.204	-0.034	1.466	63.200	19.217813	0.005	1	...	1	1	0	0	0	0	0.0	0.0	0	0
10773585	2022-06-02 15:49:51	-0.010	8.438	8.202	-0.032	1.464	63.225	19.217813	0.005	1	...	1	1	0	0	0	0	0.0	0.0	0	0
10773586	2022-06-02 15:49:52	-0.010	8.438	8.202	-0.032	1.466	63.175	19.217813	0.005	1	...	1	1	0	0	0	0	0.0	0.0	0	0
10773587	2022-06-02 15:49:53	-0.008	8.438	8.200	-0.032	1.468	63.200	19.227156	0.005	1	...	1	1	0	0	0	0	0.0	0.0	0	0

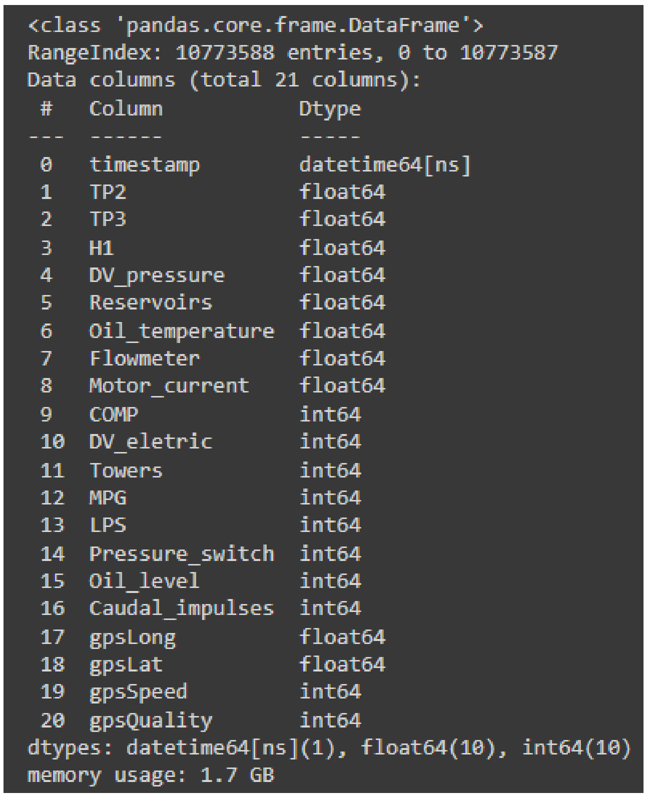
5 rows × 21 columns

Figure 2.1.2: Initial dataset tail rows

Dataset's shape is 10.773.587 rows by 21 columns.

Make sure that datatypes on all columns are as expected (float for analog signals, integers for digital signals).

```
df_raw.info()

```



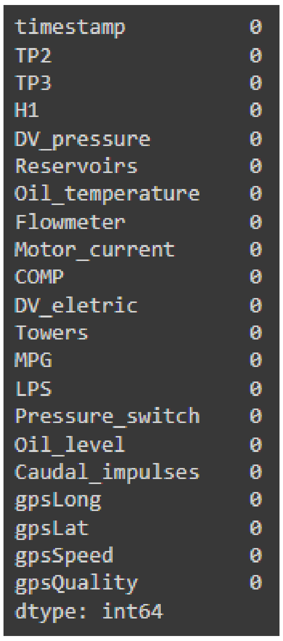
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10773588 entries, 0 to 10773587
Data columns (total 21 columns):
 #   Column           Dtype
---  ------           -----
 0   timestamp        datetime64[ns]
 1   TP2              float64
 2   TP3              float64
 3   H1               float64
 4   DV_pressure      float64
 5   Reservoirs       float64
 6   Oil_temperature  float64
 7   Flowmeter        float64
 8   Motor_current    float64
 9   COMP             int64
 10  DV_eletric       int64
 11  Towers           int64
 12  MPG              int64
 13  LPS              int64
 14  Pressure_switch  int64
 15  Oil_level        int64
 16  Caudal_impulses  int64
 17  gpsLong          float64
 18  gpsLat           float64
 19  gpsSpeed         int64
 20  gpsQuality       int64
dtypes: datetime64[ns](1), float64(10), int64(10)
memory usage: 1.7 GB
```

Figure 2.1.3: Datatype information

All datatypes are in order.

It is important to check if there are missing data (null values) in each column.

```
df_raw.isnull().sum()

```

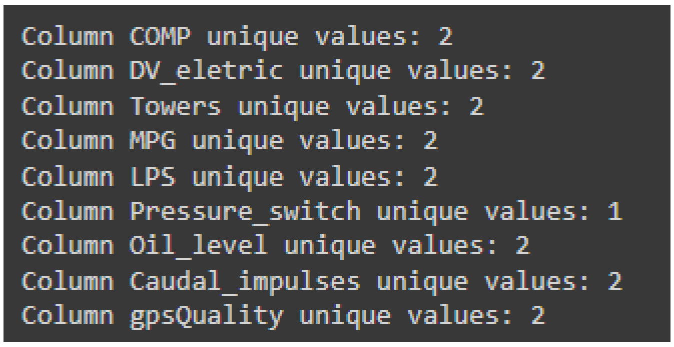


```
timestamp          0
TP2                0
TP3                0
H1                 0
DV_pressure        0
Reservoirs         0
Oil_temperature    0
Flowmeter          0
Motor_current      0
COMP               0
DV_eletric         0
Towers             0
MPG                0
LPS                0
Pressure_switch    0
Oil_level          0
Caudal_impulses    0
gpsLong            0
gpsLat             0
gpsSpeed           0
gpsQuality         0
dtype: int64
```

Figure 2.1.4: Dataset Null value check

Ensure that all digital sensors' readings are indeed binary. This is done by selecting all "int64" datatype columns and counting the unique values in each of them.

```
digitalReadings = df_raw.select_dtypes(include=['int64'])
digitalCols = list(digitalReadings.columns)
digitalCols.remove("gpsSpeed") # definitely not an on-off signal

for col_name in digitalCols:
  print(f"Column {col_name} unique values: {digitalReadings[col_name].
nunique()} " )

```



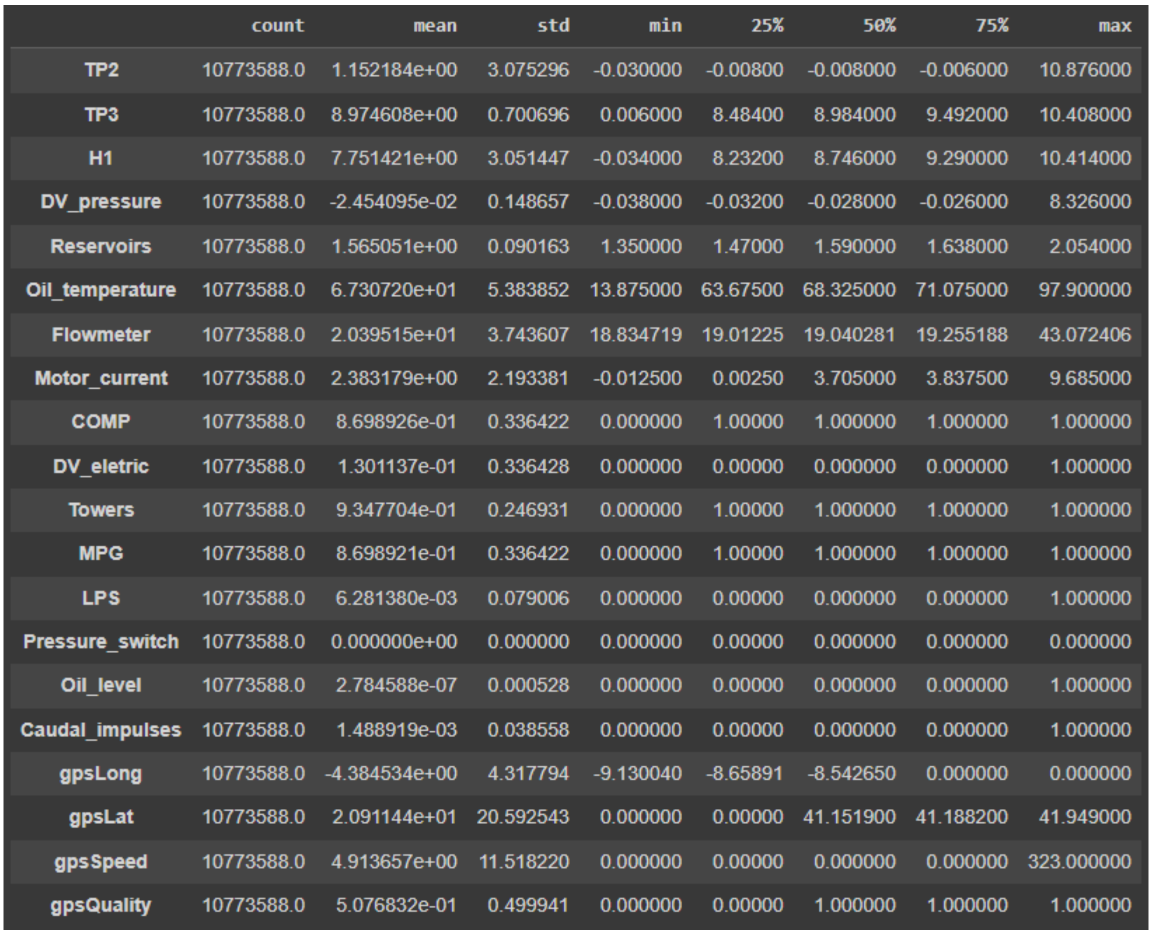
```
Column COMP unique values: 2
Column DV_eletric unique values: 2
Column Towers unique values: 2
Column MPG unique values: 2
Column LPS unique values: 2
Column Pressure_switch unique values: 1
Column Oil_level unique values: 2
Column Caudal_impulses unique values: 2
Column gpsQuality unique values: 2
```

Figure 2.1.5: Dataset Binary value check

We see that the values are indeed Binary. Interestingly enough, Pressure_Switch readings have one unique value. This means that this feature brings no variability in the dataset, meaning that it needs to be eliminated later on, during the Feature Engineering stage. (see Chapter 3: FEATURE ENGINEERING)

Finally, let's see a description of the dataset regarding its min/max values and value distribution

```
df_raw.describe().T
```



	count	mean	std	min	25%	50%	75%	max
TP2	10773588.0	1.152184e+00	3.075296	-0.030000	-0.00800	-0.008000	-0.006000	10.876000
TP3	10773588.0	8.974608e+00	0.700696	0.006000	8.48400	8.984000	9.492000	10.408000
H1	10773588.0	7.751421e+00	3.051447	-0.034000	8.23200	8.746000	9.290000	10.414000
DV_pressure	10773588.0	-2.454095e-02	0.148657	-0.038000	-0.03200	-0.028000	-0.026000	8.326000
Reservoirs	10773588.0	1.565051e+00	0.090163	1.350000	1.47000	1.590000	1.638000	2.054000
Oil_temperature	10773588.0	6.730720e+01	5.383852	13.875000	63.67500	68.325000	71.075000	97.900000
Flowmeter	10773588.0	2.039515e+01	3.743607	18.834719	19.01225	19.040281	19.255188	43.072406
Motor_current	10773588.0	2.383179e+00	2.193381	-0.012500	0.00250	3.705000	3.837500	9.685000
COMP	10773588.0	8.698926e-01	0.336422	0.000000	1.00000	1.000000	1.000000	1.000000
DV_eletric	10773588.0	1.301137e-01	0.336428	0.000000	0.00000	0.000000	0.000000	1.000000
Towers	10773588.0	9.347704e-01	0.246931	0.000000	1.00000	1.000000	1.000000	1.000000
MPG	10773588.0	8.698921e-01	0.336422	0.000000	1.00000	1.000000	1.000000	1.000000
LPS	10773588.0	6.281380e-03	0.079006	0.000000	0.00000	0.000000	0.000000	1.000000
Pressure_switch	10773588.0	0.000000e+00	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000
Oil_level	10773588.0	2.784588e-07	0.000528	0.000000	0.00000	0.000000	0.000000	1.000000
Caudal_impulses	10773588.0	1.488919e-03	0.038558	0.000000	0.00000	0.000000	0.000000	1.000000
gpsLong	10773588.0	-4.384534e+00	4.317794	-9.130040	-8.65891	-8.542650	0.000000	0.000000
gpsLat	10773588.0	2.091144e+01	20.592543	0.000000	0.00000	41.151900	41.188200	41.949000
gpsSpeed	10773588.0	4.913657e+00	11.518220	0.000000	0.00000	0.000000	0.000000	323.000000
gpsQuality	10773588.0	5.076832e-01	0.499941	0.000000	0.00000	1.000000	1.000000	1.000000

Figure 2.1.6: Dataset Value Description

The values make sense. Binary values are indeed 0 (zero) and 1 (one) and the Analog readings are not extreme. There are some minor and non-senscial negative values, but they will be updated later on as 0 (zero). (see Chapter 3: FEATURE ENGINEERING)

2.2 Graphical Representation

In order to graphically compare the Normal and Fault states, relevant Timeframes need to be selected. The Fault Timeframes are known, so Normal Timeframes with similar size will be sampled. Since the ”Oil Leak Fault” has more than 280.000 samples, it will be downsampled for readability purposes. (see Figure 1.3.1)

To begin with, let’s define the Timeframes.

```
# Fault Timeframes
T1, T2 = pd.Timestamp("2022-02-28 21:53:00"), pd.Timestamp("2022-03-01
02:00:00")
T3, T4 = pd.Timestamp("2022-03-23 14:54:00"), pd.Timestamp("2022-03-23
15:24:00")
T5, T6 = pd.Timestamp("2022-05-30 12:00:00"), pd.Timestamp("2022-06-02
06:18:00")

# Normal Timeframes
T1_ok, T2_ok = pd.Timestamp("2022-02-26 21:53:00"), pd.Timestamp
("2022-02-27 02:00:00")
T3_ok, T4_ok = pd.Timestamp("2022-03-20 14:54:00"), pd.Timestamp
("2022-03-20 15:24:00")
T5_ok, T6_ok = pd.Timestamp("2022-05-21 18:00:00"), pd.Timestamp
("2022-05-21 22:00:00")

```

The respective dataframes are created.

```
# Fault dataframes
fault1_df = fault2_df = fault3_df = df

fault1_df = df[(df['timestamp'] >= T1) & (df['timestamp'] <= T2)]
fault2_df = df[(df['timestamp'] >= T3) & (df['timestamp'] <= T4)]
fault3_df = df[(df['timestamp'] >= T5) & (df['timestamp'] <= T6)]

fault1_df = fault1_df.reset_index(drop=True)
fault2_df = fault2_df.reset_index(drop=True)
fault3_df = fault3_df.reset_index(drop=True)

# Normal dataframes
ok1_df = ok2_df = ok3_df = df

ok1_df = df[(df['timestamp'] >= T1_ok) & (df['timestamp'] <= T2_ok)]
ok2_df = df[(df['timestamp'] >= T3_ok) & (df['timestamp'] <= T4_ok)]
ok3_df = df[(df['timestamp'] >= T5_ok) & (df['timestamp'] <= T6_ok)]

ok1_df = ok1_df.reset_index(drop=True)
ok2_df = ok2_df.reset_index(drop=True)
ok3_df = ok3_df.reset_index(drop=True)

```

Printing the dataframes' shapes, it is confirmed they are now comparable and ready to be plotted.

```
# Fault dataframes shape
print(fault1_df.shape)
print(fault2_df.shape)
print(fault3_df.shape)

# Normal dataframes shape
print(ok1_df.shape)
print(ok2_df.shape)
print(ok3_df.shape)

```

Event	Fault DF Length	Normal DF Length
Air Leak Fault 1	14.821	14.821
Air Leak Fault 2	1.801	1.801
Oil Leak Fault	13.802	14.401

Table 2.1: Normal and Fault Dataframes Length

The function below will be called to create and display the plots.

The function arguments are the following:

- faultDataframe - Pass the dataframe with the Fault values
- normalDataframe - Pass the dataframe with the Normal values
- featuresList - Pass either the Analog features list or the Digital features list
- type - Define if the features are Analog or Digital (to select the proper plot function)

```
def plotTimeSeries(faultDataframe, NormalDataframe, featuresList, type):
  fig, axs = plt.subplots(len(featuresList), 1, figsize=(20, 20), sharex=
True)

  for i, feature in enumerate(featuresList):
  if type='Analog':
    axs[i].plot(faultDataframe.index, faultDataframe[feature], label=f"{
feature} - Fault State", color='red')
    xaxislabel='Index'
  else:
    axs[i].step(faultDataframe.index, faultDataframe[feature], label=f"{
feature} - Fault State", where='post', color='red')
    xaxislabel='Index'
  axs[i].set_ylabel(feature)
  axs[i].legend(loc='upper right')
  axs[i].grid(True)

  for i, feature in enumerate(featuresList):
    axs[i].plot(NormalDataframe.index, NormalDataframe[feature], label=f"{
feature} - Normal State")
    axs[i].set_ylabel(feature)
```

```
        axs[i].legend(loc='upper right')
        axs[i].grid(True)

    axs[-1].set_xlabel(xaxislabel)
    plt.tight_layout()
    plt.show()

```

Now the graphs will be plotted using the following commands:

```
plotTimeSeries(fault1_df, ok1_df, analogCols, type='Analog')
plotTimeSeries(fault1_df, ok1_df, digitalCols, type='Digital')

plotTimeSeries(fault2_df, ok2_df, analogCols, type='Analog')
plotTimeSeries(fault2_df, ok2_df, digitalCols, type='Digital')

plotTimeSeries(fault3_df, ok3_df, analogCols, type='Analog')
plotTimeSeries(fault3_df, ok3_df, digitalCols, type='Digital')

```

2.2.1 First Fault event comparison

Here are the plots for Analog and Digital features during the First Fault event.

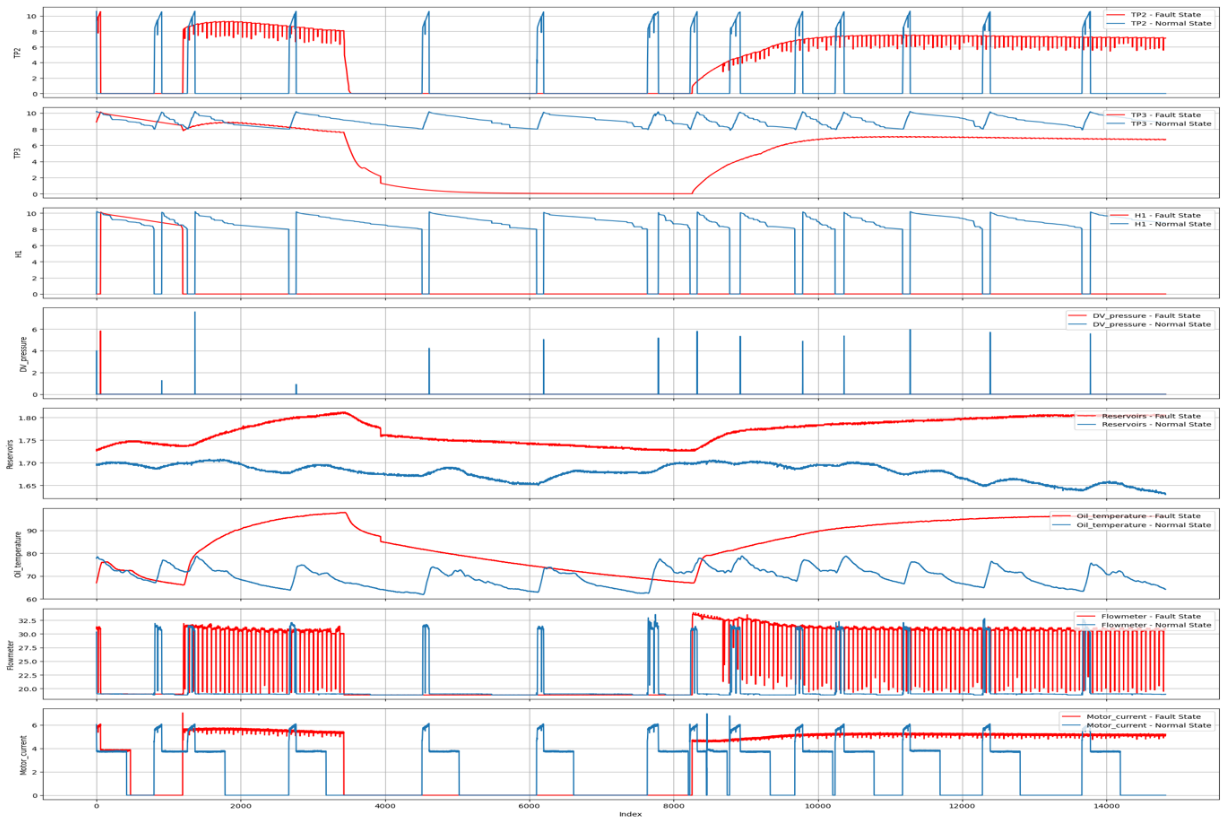


Figure 2.2.1: Analog features plot for First Fault event

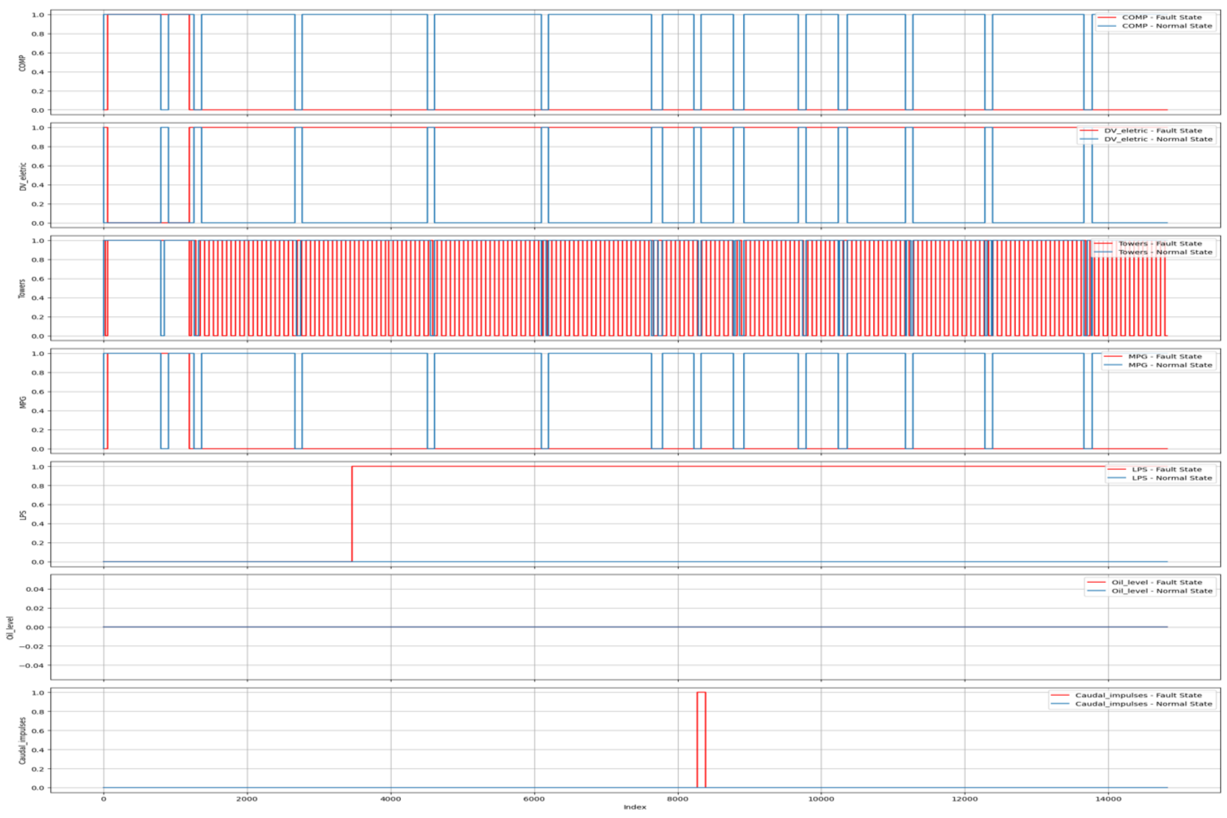


Figure 2.2.2: Digital features plot for First Fault event

2.2.2 Second Fault event comparison

Here are the plots for Analog and Digital features during the First Fault event.

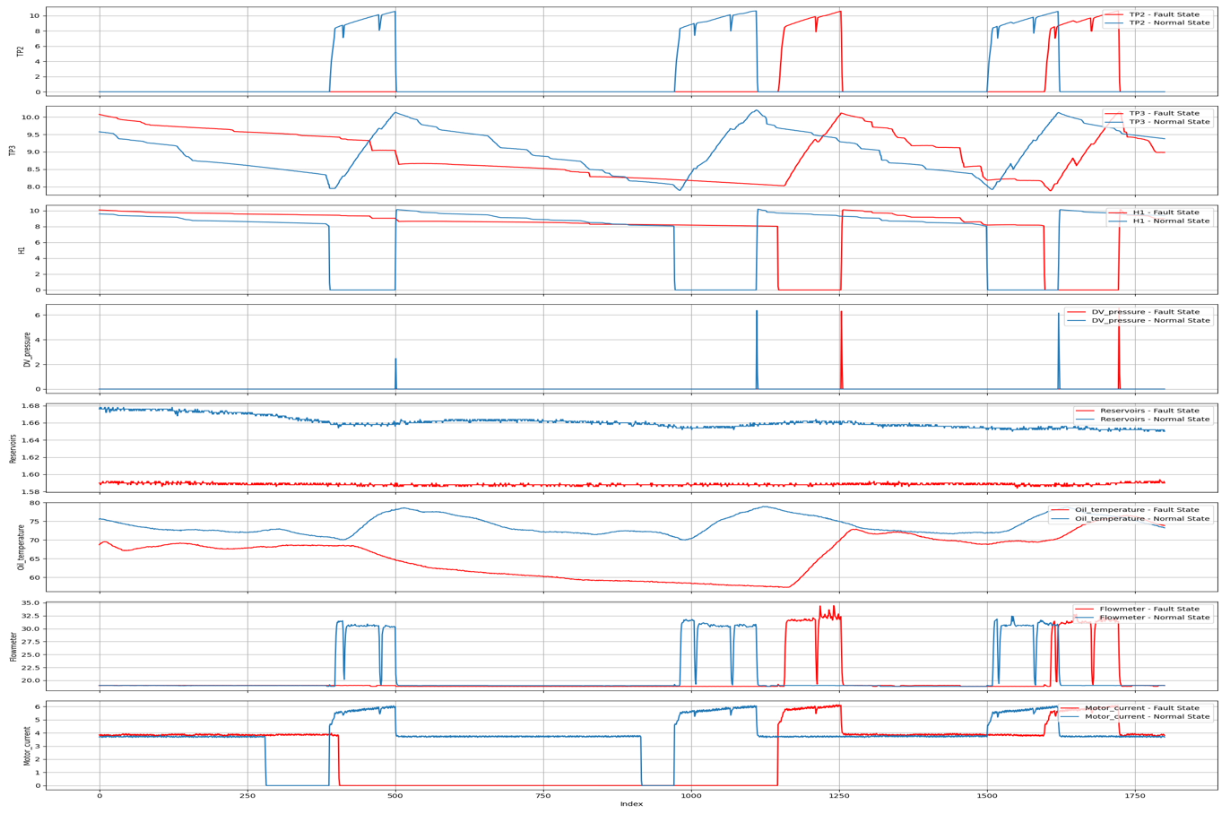


Figure 2.2.3: Analog features plot for Second Fault event

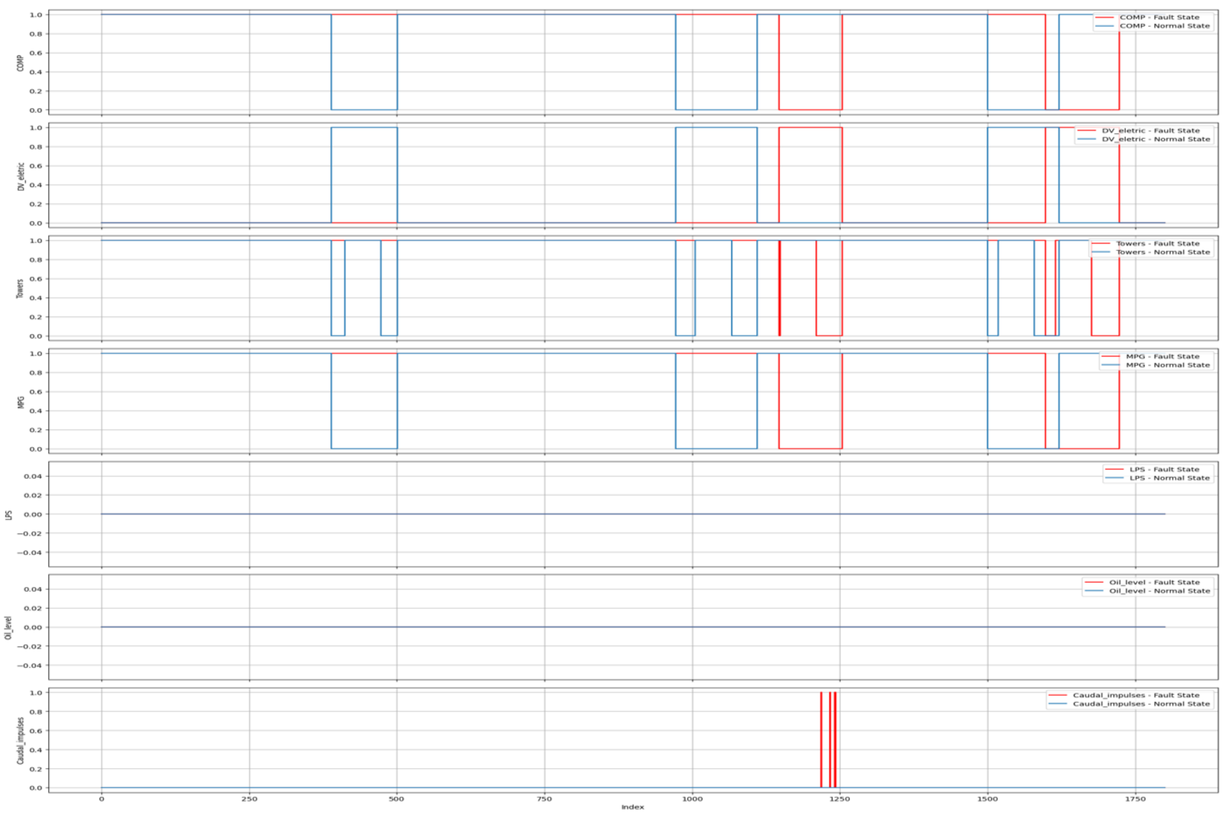


Figure 2.2.4: Digital features plot for Second Fault event

2.2.3 Third Fault event comparison

Here are the plots for Analog and Digital features during the First Fault event.

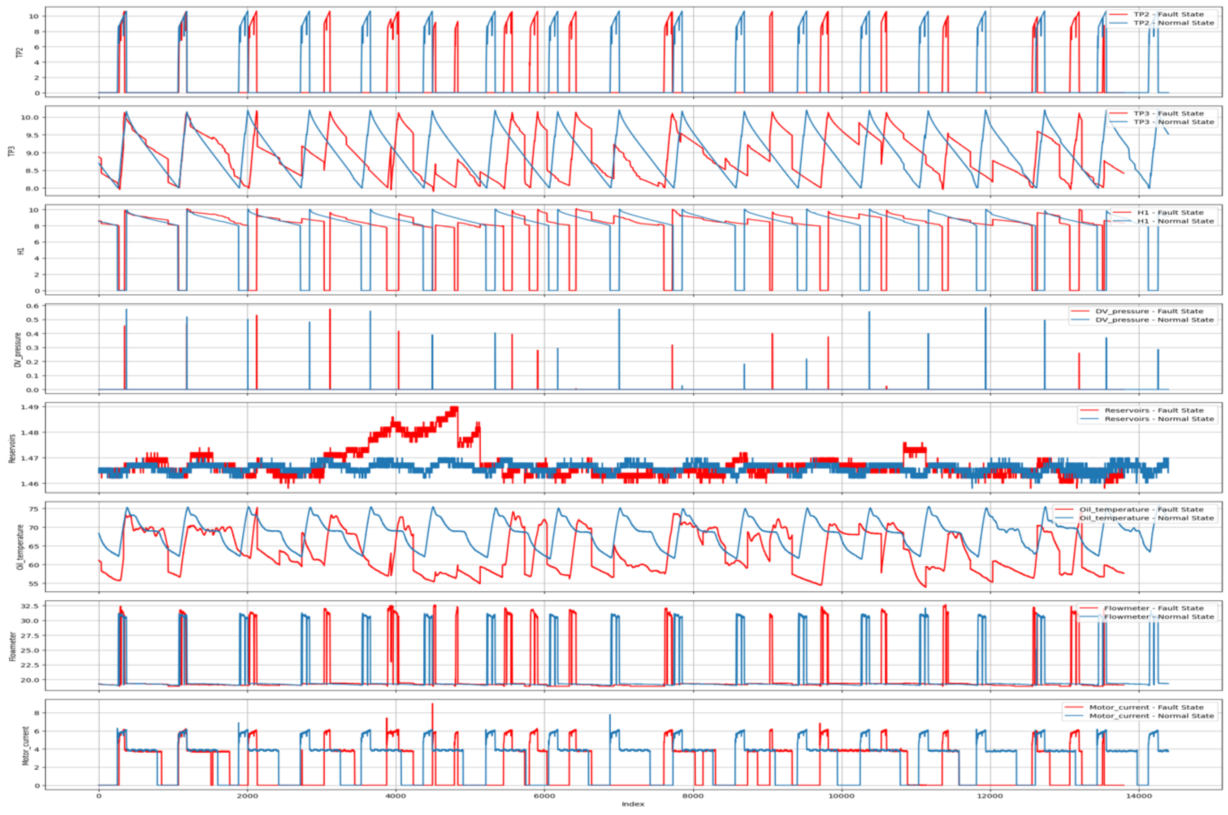


Figure 2.2.5: Analog features plot for Third Fault event

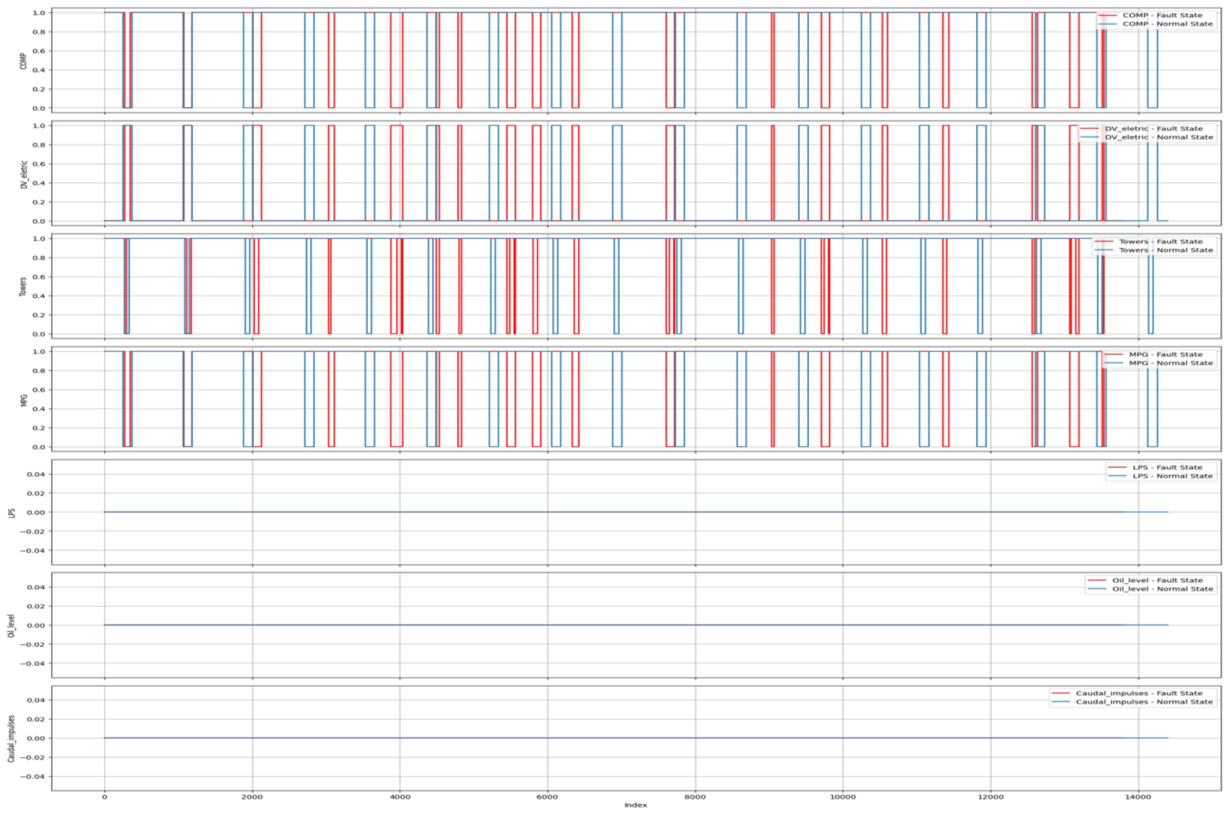


Figure 2.2.6: Digital features plot for Third Fault event

2.2.4 Conclusions from Graphical comparison

Observing the plots above, the following conclusions can be drawn:

- The ”Oil_level” feature is not helpful, since its values do not differ. Even when there is an ”Oil Leak Fault”.
- ”Caudal_impulses” is a signal produced from the ”sensitive” Flowmeter. However, it does not seem to have a noteworthy behaviour.
- In Fault State 2 and 3, ”TP2”, ”TP3”, ”H1” and ”DV pressure” seem to only have a phase difference. They behave differently in Fault State 1.
- ”Reservoir” measurements are distinct in Fault State 1 and 2.

Therefore, ”Oil_level” and ”Caudal_impulses” will be dropped during Feature Engineering (see Chapter 3: FEATURE ENGINEERING). The rest of the features will be used for predictions regarding all cases.

Chapter 3

FEATURE ENGINEERING

Feature engineering is a crucial aspect of Machine Learning and Data Analysis that involves transforming raw data into a format that is suitable for training predictive models. It encompasses a variety of techniques aimed at extracting relevant information from the data and creating *new features* that can enhance the performance of Machine Learning algorithms. Feature engineering is not only about selecting the right features but also involves processes like scaling, normalization, encoding categorical variables, handling missing values, and creating interaction terms or polynomial features. The goal is to represent the data in a way that highlights meaningful patterns and relationships, improving the predictive accuracy and generalisation of the models.

One of the key reasons why feature engineering is important is its direct impact on the performance of Machine Learning models. The quality of the features used to train a model can significantly influence its ability to make accurate predictions. By engineering relevant features, engineers can provide the model with more informative input, enabling it to learn complex patterns and relationships in the data more effectively. Moreover, feature engineering can help mitigate issues such as overfitting by reducing the dimensionality of the feature space or by creating features that are more robust to noise and outliers. Ultimately, well-engineered features can lead to models that are more efficient and capable of generalising to never-before seen data.

This concept plays a crucial role in bridging the gap between raw data and actionable insights. It allows domain experts to leverage their knowledge and intuition about the data to create features that capture meaningful aspects of the underlying processes. By incorporating domain-specific knowledge into the feature engineering process, practitioners can design models that are not only accurate but also aligned with the requirements and constraints of the problem. Additionaly, feature engineering can uncover hidden patterns and relationships in the data that may not be apparent at first glance, enabling more informed decision-making and facilitating a deeper understanding of the underlying mechanisms driving the observed phenomena. Overall, feature engineering is an essential step in the Machine Learning pipeline that can unlock the full potential of data and enhance the effectiveness of predictive modeling tasks.

3.1 Apply conclusions from EDA

Based on the findings of the previous chapter, there are values that need to be corrected and features that need to be dropped. Analog reading negative and close to 0 (zero), must be considered as 0. Also, features that seem to not have a significant impact will be dropped. GPS device-related features will be dropped as well, since the approach of this Thesis does not leverage position knowledge (Maintenance Station, Parking Station) in any way.

Let's begin with fixing the negative values (see Figure 2.1.6).

```
df.loc[ (df['TP2'] < 0), 'TP2' ] = 0
df.loc[ (df['H1'] < 0), 'H1' ] = 0
df.loc[ (df['DV_pressure'] < 0), 'DV_pressure' ] = 0
df.loc[ (df['Motor_current'] < 0), 'Motor_current' ] = 0

```

Then, non-useful features will be dropped.

```
columns_to_drop = ['Pressure_switch', 'gpsLat', 'gpsLong', 'gpsQuality', '
gpsSpeed', 'Oil_level', 'Caudal_impulses']
df = df.drop(columns = columns_to_drop)

```

3.2 New features

New features need to be introduced to serve as target features for the predictive models. Regarding the Regression problem of Remaining Useful Life prediction, there will be one feature for each event. The time of Fault occurrence is known, therefore the new features will be numerical values representing the time elapsed since the first known sample (2022-01-01 06:00:00).

As for the Classification problem of State predicting, one-hot encoded features will be introduced. In the beginning, there will be 1 feature representing the presence of each Fault State. Then, ”Air Leak” faults will be merged and a new, ”No Fault” feature will be created comparing the Fault states. For example, if either ”Air Leak” or ”Oil Leak” are equal to 1, ”No Fault” will be 0. In contrast, if both are Fault features are 0, ”No Fault” will be 1.

The features are engineered with the code below. Consider the Fault Timestamps defined as earlier (T1,T2 for First Fault and so on...)

For RUL events:

```
df_datetime = pd.DataFrame( df['timestamp'], columns=['timestamp'] )

failure_events = [
  {'start_time':T1, 'end_time':T2},
  {'start_time':T3, 'end_time':T4},
  {'start_time':T5, 'end_time':T6},
]

for i, event in enumerate(failure_events, start=1):
  start_time = pd.to_datetime(event['start_time'])
  end_time = pd.to_datetime(event['end_time'])

  df[f'RUL_event_{i}'] = (start_time - df_datetime['timestamp']).dt.
total_seconds()
```

```
        df[f'RUL_event_{i}'] = df[f'RUL_event_{i}'].clip(lower=0)

```

A negative value means that the error has already occurred, so the .clip() method is used to set such values to 0. The values need to be transformed to days before feeding them into the Machine Learning models, because it makes no physical sense to attempt predictions in seconds. Apart from this, it would bring extremely high variability to the target feature.

For Classification events:

```
df['AirLeakFault1'] = (df['timestamp'].between(T1,T2)).astype(int)
df['AirLeakFault2'] = (df['timestamp'].between(T3,T4)).astype(int)
df['OilLeakFault3'] = (df['timestamp'].between(T5,T6)).astype(int)

# Merge Air Leak Faults
df['AirLeakFault'] = df['AirLeakFault1'] | df['AirLeakFault2']
# The 'OK' class
df['NoFault'] = (df['OilLeakFault'] == 0) & (df['AirLeakFault'] == 0)
df['NoFault'] = df['NoFault'].astype(int)

# Drop not useful columns
df = df.drop(['AirLeakFault1', 'AirLeakFault2', axis=1)
df = df.rename(columns={'OilLeakFault3': 'OilLeakFault'})

```

Firstly, the values of the rows between the Fault Timestamps are set to 1 (one), indicating that the Fault is currently present. Then, ”AirLeak” faults are merged to a single column using a ”logical OR” operation, because they belong to the same class. Finally, the ”NoFault” column is introduced, which is 0 (zero) when there is a Fault and 1 (one) when the equipment is in Normal operation.

To summarise, six (6) new features were introduced:

- RUL_event_1 - Time until Fault event 1 occurs (in seconds).
- RUL_event_2 - Time until Fault event 2 occurs (in seconds).
- RUL_event_3 - Time until Fault event 3 occurs (in seconds).
- NoFault - 1 when State is Normal, 0 when there is a Fault.
- AirLeakFault - 1 when there is an Air Leak, 0 when there is not.
- OilLeakFault - 1 when there is an Oil Leak, 0 when there is not.

Chapter 4

REMAINING USEFUL LIFE ESTIMATION

Remaining Useful Life (RUL) is a critical concept in Predictive Maintenance engineering, referring to the amount of time a system or component is expected to remain operational before it becomes ineffective. It is a key metric used to assess the health and reliability of assets, ranging from machinery and equipment to all sorts of complex systems. Predicting RUL involves analysing historical data, such as sensor readings, maintenance records, and operational parameters, to model the degradation and failure patterns of the equipment over time. By forecasting the remaining operational lifespan of important parts, organizations can optimise maintenance schedules, minimise downtime, and reduce costs by performing maintenance only when deemed necessary.

Accurate RUL predictions are essential for enabling proactive maintenance strategies, such as condition-based maintenance and Predictive Maintenance, which aim to identify and address potential issues before they lead to costly disruptions. By leveraging advanced analytics techniques, including Machine Learning algorithms, organizations can develop predictive models that take into account various factors influencing asset degradation, such as operating conditions, environmental factors, and usage patterns. These models provide valuable insights into the health and performance of assets, allowing maintenance teams to prioritize activities, allocate resources more efficiently, and extend the useful life of equipment and machinery.

In this chapter, RUL estimations by several algorithms will be presented. Their performance was tested on predicting the RUL of the First Fault event and the most accurate model was selected to predict RUL for the remaining Fault events. In order to save computational resources and calculating time, the dataset was split into 3 separate sub-sets. So, the corresponding dataset will be loaded for each Fault event.

Various experiments were ran over the course of this study. When the model has the ability to accept various *hyper-parameters*, only the best-performing ones are selected and presented. Consider also that there were **Computational Resources limitations**, so models' size is restricted. Regarding tests execution, the respective class is imported and the model is created. Then, Training is executed and Predictions are performed. The results are recorded to be compared with the next models. This procedure is being followed every time.

4.1 Useful tools

There are some useful tools that help us to perform various activities regarding data manipulation. Data need to be split to training and test sets and then scaled, if required by an algorithm. Moreover, some metrics are needed in order to evaluate the models' performance. The **”sci-kit learn”** library provides all of the above as well as classes that represent Machine Learning algorithms.

The following code imports a function to split the data, some useful metric functions and a Scaling function in [0,1) range.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score,
mean_absolute_error
from sklearn.preprocessing import MinMaxScaler

```

Four metrics were used to evaluate model performance and will be explained in the next few paragraphs. For the equations, please consider the following:

- n - the number of samples in the dataset.
- y_i - the actual (true) value of the target variable for the **i**th sample.
- $\hat{y}_i$ - the predicted value of the target variable for the **i**th sample.
- $\bar{y}$ - the mean of the actual values of the target variable.

The Mean Squared Error (MSE) is a common metric used to evaluate the performance of regression models in Machine Learning. The MSE measures the *average squared deviation* of the predicted values from the actual values. A lower MSE indicates that the model's predictions are closer to the actual values, suggesting better performance. Conversely, a higher MSE indicates greater dispersion or variability between the predicted and actual values, indicating poorer model performance.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{4.1}$$

However, the Root Mean Squared Error (RMSE) is preferred over MSE because the result is to the same scale as the target variable, making it more intuitive and easy to interpret.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4.2}$$

Another useful metric is R^2 score, which represents the proportion of variance in the target variable that is predictable from the used features. It ranges in (0,1), 0 meaning that the model does not perform well in explaining the target variable variations, while 1 means that the model perfectly predicts the target values. However, a perfect R^2 score may be hiding an overfitting model. Therefore, it has to be used in conjuction with other metrics.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}i)^2}{\sum i = 1^n(y_i - \bar{y})^2} \tag{4.3}$$

Finally, Mean Absolute Error (MAE) metric will be used. The main difference with MSE is that it gives equal weight to all errors regardless of their magnitude, because there is no squaring. For this reason, it can be considered as a more accurate indicator of model performance.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4.4}$$

4.2 RUL estimation for AirLeakFault1

4.2.1 Preparation

In the beginning, the dataset is loaded and the 'RUL_event_1' feature is transformed to "days". Then, the features to be taken into account are selected. Experiments showed that excluding any of the remaining features leads to worse model performance, making some of the observations in "2.2.4: Conclusions from Graphical comparison" section obsolete. Therefore, all features were used.

```
df = pd.read_csv('df_Fault1_ALLROWS.csv',
  parse_dates=['timestamp'],
  infer_datetime_format=True,
  )
df['RUL_event_1'] = ((df['RUL_event_1']/3600.0)/24).round(3)

features = ['TP2', 'TP3', 'H1', 'DV_pressure', 'Reservoirs', '
Oil_temperature', 'Flowmeter', 'Motor_current', 'COMP', 'DV_eletric', '
Towers', 'MPG', 'LPS']

```

Next step is to define the Training and Test sets to feed the algorithms with. First, all features are assigned to X and the target variable to y. After splitting both to Train and Test subsets, Scaling will be performed to be used in algorithms that require such data.

Rows **BEFORE** the Fault occurrence will be selected. Training data will be the 90% of all data and Test data the remaining 10%.

```
X = df.loc[df['RUL_event_1'] > 0, features]
y = df.loc[df['RUL_event_1'] > 0, 'RUL_event_1']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
random_state=42)

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

```

A function to simply call and have the Metric results printed is handy.

```
def print_metrics(model, y_test, y_pred):
  mse = mean_squared_error(y_test, y_pred).round(3)
  rmse = round(math.sqrt(mse), 3)
  r2 = r2_score(y_test, y_pred).round(3)
  mae = mean_absolute_error(y_test, y_pred).round(3)

  print(f'\n{model} Test Metrics:')
  print(f'Mean Squared Error (MSE): {mse}')
  print(f'Root Mean Squared Error (RMSE): {rmse} days')
  print(f'R2 Score: {r2}')
  print(f'Mean Absolute Error (MAE): {mae} days')

```

4.2.2 Linear Regression

Linear Regression is a statistical technique used for modeling the relationship between a dependent variable and one or more independent variables (features). The relation between the variables is assumed to be linear, meaning that changes in the features are associated with a constant change in the dependent variable. In its simplest form, linear regression aims to fit a straight line to the data points in such a way that the sum of the squared differences between the observed and predicted values is minimised. This is typically done using the method of least squares, where the parameters of the linear model are estimated to minimise the sum of squared differences. Of course, the relation of variables in our case is non-linear, but it is tried for the sake of exploring the full breadth of potential models .

```python
from sklearn.linear_model import LinearRegression

linear_model = LinearRegression()
linear_model.fit(X_train_scaled, y_train)

# Test set
y_pred = linear_model.predict(X_test_scaled)

print_metrics('Linear Regression', y_test, y_pred)

```

Mean Squared Error	Root Mean Squared Error (days)	R^2 score	Mean Absolute Error (days)
256.05	16.002	0.11	13.608

Table 4.1: Linear Regression Model Test Results

As expected, the results are way off the desired prediction performance, since the variable are not linearly related.

4.2.3 Decision Tree Regressor

Decision Trees are a powerful Supervised-learning algorithm used for both Regression and Classification tasks in Machine Learning. They are easy to understand, and can capture complex relationships between features and target variables.

The structure of a Decision Tree resembles a flowchart.

- Each internal node represents a decision based on the value of a specific feature
- Each branch represents the outcome of that decision
- Each leaf node represents the final decision (prediction)

The algorithm aims to recursively partition the feature space into regions that are as homogeneous as possible with respect to the target variable.

```python
from sklearn.tree import DecisionTreeRegressor

tree_model = DecisionTreeRegressor(max_depth=32, random_state=42)
tree_model.fit(X_train, y_train)

# Test set
y_pred = tree_model.predict(X_test)

```

```
print_metrics('Decision Tree', y_test, y_pred)

```

Mean Squared Error	Root Mean Squared Error (days)	R^2 score	Mean Absolute Error (days)
118.852	10.902	0.587	4.983

Table 4.2: Decision Tree Model Test Results

The algorithm is not performing great, if we consider the R^2 score metric. However, Decision Trees can be used as a base for more advanced models.

4.2.4 Random Forest Regressor

Random Forest is an Ensemble Learning technique that combines the strength of multiple Decision Trees to improve predictive performance and generalisation. It is one of the most widely used and robust algorithms for Regression and Classification tasks.

In a Random Forest, a collection of Decision Trees is trained on different subsets of the Training Data and using different subsets of features. This randomness among the Trees is helping to reduce overfitting and improve the model's ability to generalise unseen data. During Training, each Tree in the forest outputs a prediction and the final prediction is made by aggregating the predictions of all Trees. This is done by Averaging (Regression tasks) or Voting (Classification tasks).

```
from sklearn.ensemble import RandomForestRegressor

rf_model = RandomForestRegressor(n_estimators=10, max_depth=32,
random_state=42)
rf_model.fit(X_train, y_train)

# Test set
y_pred = rf_model.predict(X_test)

print_metrics('Random Forest', y_test, y_pred)

```

Mean Squared Error	Root Mean Squared Error (days)	R^2 score	Mean Absolute Error (days)
74.356	8.623	0.742	4.794

Table 4.3: Random Forest Model Test Results

Even though Random Forest's RMSE and MAE metrics are not that different compared to Decision Tree's, R^2 score is significantly higher, indicating that it performs better.

4.2.5 Gradient Boosting Regressor

Gradient Boosting is an Ensemble technique that combines multiple weak learners (typically Decision Trees) to create a strong predictive model. The main idea behind Gradient Boosting is to iteratively train a sequence of weak learners in such a way that each subsequent learner focuses on the mistakes made by the previous learners. Each new learner is fitted to the differences between the actual and predicted values of the Ensemble's predictions. In essence, each new learner attempts to correct the errors made by the Ensemble up to that point.

```
from sklearn.ensemble import GradientBoostingRegressor

gb_model = GradientBoostingRegressor(n_estimators=20, learning_rate=0.1,
max_depth=8, random_state=42)
gb_model.fit(X_train_scaled, y_train)

# Test set
y_pred = gb_model.predict(X_test_scaled)

print_metrics('Gradient Boosting', y_test, y_pred)

```

Mean Squared Error	**Root Mean Squared Error (days)**	**R^2 score**	**Mean Absolute Error (days)**
153.284	12.381	0.467	9.334

Table 4.4: Gradient Boosting Model Test Results

Gradient Boosting is performing worse than Random Forest, possibly because of its outlier and noise sensitivity, since it builds Trees *sequentially*. This can lead to severe influence by such samples in the early stages of Training.

4.2.6 XGBoost Regressor

XGBoost, (eXtreme Gradient Boosting), is a highly efficient and scalable implementation of the Gradient Boosting algorithm. It is widely regarded as one of the most powerful Machine Learning algorithms, especially for structured/tabular data.

It builds upon the principles of Gradient Boosting, where an Ensemble of weak learners are sequentially trained to correct the errors of the previous learners. However, XGBoost contains several enhancements that significantly improve its performance, speed, and scalability:

- Regularisation: Incorporates regularisation techniques to prevent overfitting. This helps generalising, particularly on noisy datasets.
- Gradient-based optimisation: Efficiently finds the optimal tree structure by minimising a regularized objective function.
- Tree pruning: Implements advanced tree pruning techniques to remove branches that provide little to no improvement in the model's performance, reducing model complexity and improving efficiency.

```
import xgboost as xgb

xgb_model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=10,
 learning_rate=0.1, max_depth=32, random_state=42)
```

```
xgb_model.fit(X_train_scaled, y_train)

# Test set
y_pred = xgb_model.predict(X_test_scaled)

print_metrics('XGBoost', y_test, y_pred)

```

Mean Squared Error	Root Mean Squared Error (days)	R^2 score	Mean Absolute Error (days)
104.328	10.214	0.638	7.922

Table 4.5: XGBoost Model Test Results

Even though XGBoost performs better than Gradient Boosting, Random Forest is still more accurate. It is expected to have similar vulnerabilities to Gradient Boosting, but it was worth a shot to check if it could outperform Random Forest.

4.2.7 K-Nearest Neighbors Regressor

K-Nearest Neighbors (KNN) is a simple Supervised Learning algorithm. It belongs to the category of instance-based (or lazy learning) algorithms. In this category, the Model does not learn a mapping or pattern from input features to output labels during Training. Instead, Predictions are made based on the similarity of unseen data points to the Training instances.

The idea behind KNN is to Predict the target variable of a new data point by considering the labels of its nearest neighbors in the feature space, assuming that samples with similar feature values are likely to have similar target values. A severe drwaback is that it requires the entire Training set to make predictions, which makes it computationally expensive for large datasets.

```
from sklearn.neighbors import KNeighborsRegressor

knn_model = KNeighborsRegressor(n_neighbors=5)
knn_model.fit(X_train_scaled, y_train)

# Test set
y_pred = knn_model.predict(X_test_scaled)

print_metrics('KNN', y_test, y_pred)

```

Mean Squared Error	Root Mean Squared Error (days)	R^2 score	Mean Absolute Error (days)
84.823	9.21	0.705	5.103

Table 4.6: K-Nearest Neighbors Model Test Results

KNN performed close to Random Forest. It would be interesting to see its performance with a higher number of neighbors but, as mentioned in this Chapter's introduction, Computational Resources are limited at the time of this study.

4.2.8 Result Summarisation

This are the results as they derived from the experiments above:

Model	Mean Squared Error	Root Mean Squared Error (days)	R^2 score	Mean Absolute Error (days)
Linear Regression	256.05	16.002	0.11	13.608
Decision Tree	118.852	10.902	0.587	4.983
Random Forest	74.356	8.623	0.742	4.794
Gradient Boosting	153.284	12.381	0.467	9.334
XGBoost	104.328	10.214	0.638	7.922
K-Nearest Neighbors	84.823	9.21	0.705	5.103

Table 4.7: Regression Models Test Results Summarisation

Random Forest proved to be the most accurate algorithm. However, it is still away from the 90% threshold (R^2 score), were we can consider a Model somewhat reliable. Gradient Boosting and XG-Boost underperformance implies noise or outliers lurking in the dataset, something that should have been explored extensively. The nice surprise is K-Nearest Neighbors algorithm, which performs closely to Random Forest, but unfortunately cannot be further explored, due to it being highly computationally demanding during prediction.

So, to predict RUL for ”AirLeakFault2” and ”OilLeakFault”, Random Forest algorithm with the current configuration will be used. The same preparation stage (see Subsection 4.2.1: Preparation) will be followed for both cases, except that now the target values will be ”RUL_event_2” and ”RUL_event_3” respectively.

4.3 RUL estimation for AirLeakFault2 and OilLeakFault

Applying Random Forest Model for both cases, the following results are produced, along with the earlier case:

Fault	Mean Squared Error	Root Mean Squared Error (days)	R^2 score	Mean Absolute Error (days)
RUL_event_1	74.356	8.623	0.742	4.794
RUL_event_2	166.349	12.898	0.698	7.132
RUL_event_3	184.727	13.591	0.902	8.207

Table 4.8: All RUL estimations

4.4 RUL estimation Conclusions

It can be observed that RMSE and MAE metrics are increased between each case. It is somewhat expected because the Faults occurred in different point in time, meaning that there are more samples for the latest Faults than the first event. More variance is introduced, increasing the calculated Means. On the other hand, R_2 score is in a satisfying level regarding the ”OilLeakFault” event. This leads us to the conclusion that this algorithm demonstrates improved generalisation capabilities when more Training data are available. For the first 2 Faults, it would be interesting to gain access to even earlier data, in order to test this hypothesis. Increase volume of data could cost in prediction accuracy, but it needs to be taken care of not being overly distanced from reality.

Chapter 5

STATE CLASSIFICATION

Machine State Classification is a crucial task in various industrial and manufacturing settings where monitoring and controlling the operational states of machines are of the utmost importance. The objective of State Classification is to accurately categorise the state of a machine based on sensor data, signals, or other relevant features. This classification enables real-time monitoring of machine health, identification of malfunctions, and timely intervention to prevent costly downtime. It can encompass different kinds of states, such as normal operation or specific Fault conditions, depending on the requirements of the application and the complexity of the machinery involved. To sum up, it facilitates proactive maintenance strategies, enhances operational efficiency, and improves the overall reliability and performance of industrial systems.

In this chapter, Classification models will be employed to determine between 3 states, namely ”NoFault”, ”AirLeakFault” and ”OilLeakFault”. The respective one-hot encoded target variables are already engineered during Chapter ”Feature Engineering”.

The same approach as RUL estimation models will be followed. Additionally, random ”NoFault” samples will be dropped, in order to balance the dataset.

5.1 Useful Tools

Again, **”sci-kit learn”** library will provide the classes for the Classification models, data scaling and metrics. The **”seaborn”** library will be used to visualise the confusion matrix for each model.

The following code imports a function to split the data, some metric functions, a Scaling function and the Seaborn library.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns

```

Four metrics were used to evaluate model performance and will be explained in the next few paragraphs.

Accuracy is a fundamental metric to evaluate the performance of Classification models in Machine Learning. It represents the proportion of correctly classified samples among all the samples in the dataset. Simply said, Accuracy measures the model's ability to correctly predict the class labels of the data points.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{5.1}$$

Precision is a metric used to evaluate the quality of a Classification models. It measures the proportion of correctly predicted positive samples (true positives) among all samples predicted as positive by the model. Therefore, Precision represents the model's ability to avoid samples that were incorrectly classified as positive (false positives).

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{5.2}$$

Recall is a metric used to evaluate the ability of a Classification model to correctly identify positive samples from the entire set of actual positive samples in the dataset. It measures the proportion of correctly predicted positive samples (true positives) among all instances that are actually positive.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{5.3}$$

The F1 score metric combines both Precision and Recall into a single metric, providing a balanced measure of a model's performance. The harmonic mean is used instead of a simple arithmetic mean to ensure that the F1 score gives equal weight to Precision and Recall. This means that the F1 score will be high only if both Precision and Recall are high.

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.4}$$

5.2 Machine State Classification

5.2.1 Preparation

First, a list with the State names is created. This is used to select the appropriate features for target data and assign a name to the prediction labels (0: ”NoFault”, 1:”AirLeakFault”, 2:”OilLeakFault”). After loading the dataset, the samples are splitted between Training and Test data, with a 80%-20% ratio. Then, Training data are Scaled in range [0,1) and Test data are flattened, to be fed into the models.

```
target_names = ['NoFault', 'AirLeakFault', 'OilLeakFault']
X = df[features]
y = df[target_names]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

y_train_flat = y_train.values.argmax(axis=1)
y_test_flat = y_test.values.argmax(axis=1)

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

```

A function to quickly print the metrics for each model is created in this case too.

```
def print_metrics(model, target_names, y_test, y_pred):

  print(f'\n{model} Test Metrics:')
  print(classification_report(y_test, y_pred, target_names=target_names))

  cm = confusion_matrix(y_test, y_pred)
  sns.set(rc={'figure.figsize':(15,7)})
  sns.heatmap(cm, fmt='d', annot=True, xticklabels=target_names,
yticklabels=target_names)
  plt.xlabel('Predicted Label')
  plt.ylabel('True Label')
  plt.title('Confusion Matrix')
  plt.show()

```

5.2.2 Logistic Regression

Logistic Regression is a statistical method used for binary Classification tasks. It models the probability that a given input belongs to a class by fitting a logistic function (sigmoid) to the observed data. To support the current case, it will be extended to execute multiclass Classification using the Multinomial Logistic Regression technique.

```
from sklearn.linear_model import LogisticRegression

linear_model = LogisticRegression(max_iter=1000, multi_class='multinomial')
linear_model.fit(X_train_scaled, y_train_flat)

```

```
    # Test set predictions
    y_pred = linear_model.predict(X_test_scaled)

    print_metrics('Logistic Regression', target_names, y_test_flat, y_pred)

```

Label	Accuracy	Precision	Recall	F1-score
NoFault	0.77	0.86	0.64	0.74
AirLeakFault		0.95	0.81	0.87
OilLeakFault		0.70	0.90	0.79

Table 5.1: Logistic Regression Model Results

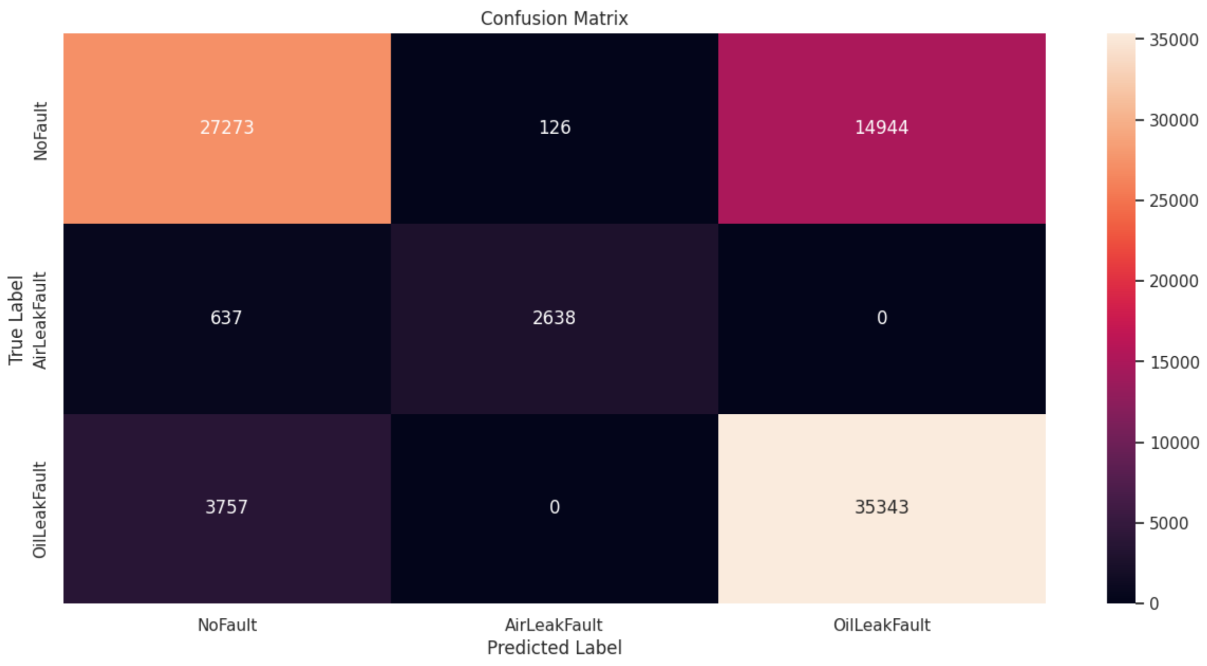


Figure 5.2.1: Logistic Regression Confusion Matrix

The accuracy of 77% indicates that the model predicts the labels relatively well. Let's see the rest of the metrics for each labels, to better explore how the model behaves for each of them.

For "NoFault" label, the model performs very well in avoiding False Positives but it does even better when it comes to "AirLeakFault" label, with a Precision of 95%. "OilLeakFault" predictions are the least precise.

Even though False Positives are avoided relatively well for "NoFault" and very well for "AirLeakFault", it does not succeed in predicting True Positives in an acceptable manner. On the other hand, "OilLeakFault" True positives are predicted successfully 90% of the time.

F1-score indicates that there is a good balance between Precision and Recall for all state labels.

Overall, the model performs well in predicting "AirLeakFault" and relatively well when it comes to "NoFault" and "OilLeakFault" states.

5.2.3 Decision Tree Classifier

Decision Tree will be used in this case as a Classifier. (see Section 4.2.3: Decision Tree Regressor)

```
from sklearn.tree import DecisionTreeClassifier

tree_model = DecisionTreeClassifier(max_depth=64, random_state=42)
tree_model.fit(X_train_scaled, y_train_flat)

# Test set predictions
y_pred = tree_model.predict(X_test_scaled)

print_metrics('Decision Tree', target_names, y_test_flat, y_pred)

```

Label	Accuracy	Precision	Recall	F1-score
NoFault	0.88	0.89	0.87	0.88
AirLeakFault		0.98	0.98	0.98
OilLeakFault		0.86	0.88	0.87

Table 5.2: Decision Tree Classifier Model Results

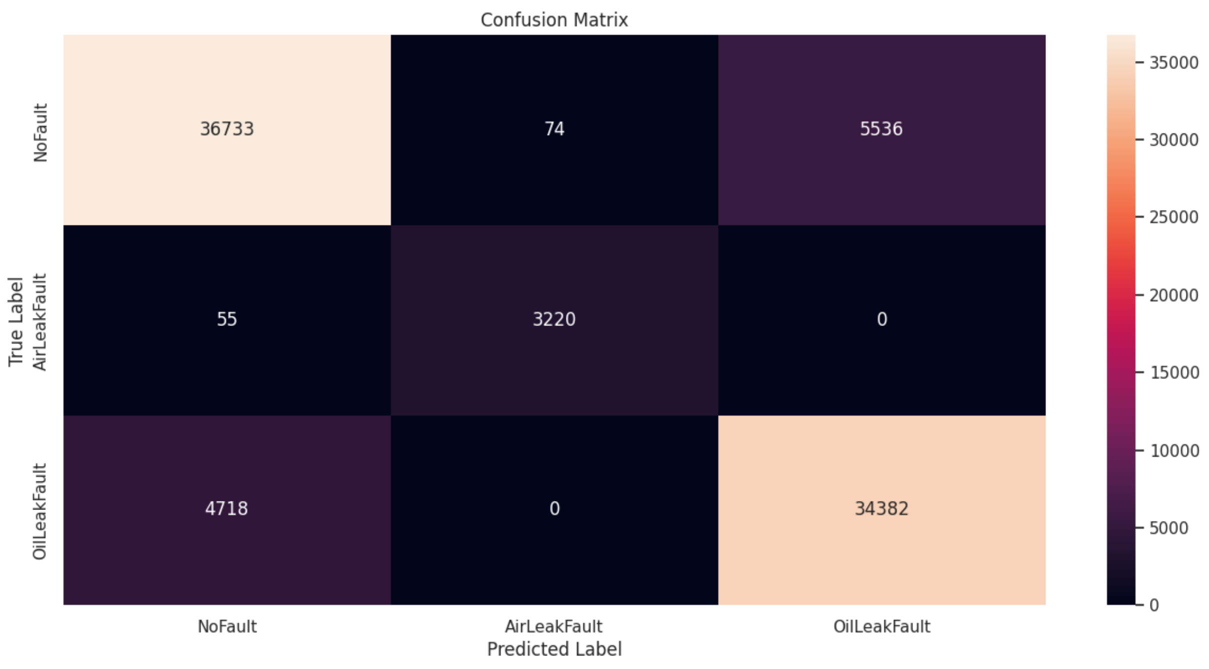


Figure 5.2.2: Decision Tree Confusion Matrix

It is immediately apparent that Decision Tree performs quite better than Logistic Regression, with a difference of 11%. This is reflected in the individual metrics.

The model avoids False Positives in a quite high level, for all kinds of Faults, especially when it comes to ”AirLeakFault” (98%).

Recall's results are at almost the same level as Precision. This means that True Positives are properly detected most of the time for all labels.

Overall, the Decision Tree model performs very well across all metrics. It achieves high quality results for all three classes. "AirLeakFault" prediction performance is quite impressive too.

5.2.4 Random Forest Classifier

Random Forest will be used in this case as a Classifier. (see Section 4.2.4: Random Forest Regressor)

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=100, max_depth=64,
random_state=42)
rf_model.fit(X_train_scaled, y_train_flat)

# Test set predictions
y_pred = rf_model.predict(X_test_scaled)

print_metrics('Random Forest', target_names, y_test_flat, y_pred)

```

Label	Accuracy	Precision	Recall	F1-score
NoFault	0.91	0.94	0.87	0.90
AirLeakFault		0.99	0.98	0.99
OilLeakFault		0.87	0.94	0.90

Table 5.3: Random Forest Classifier Model Results

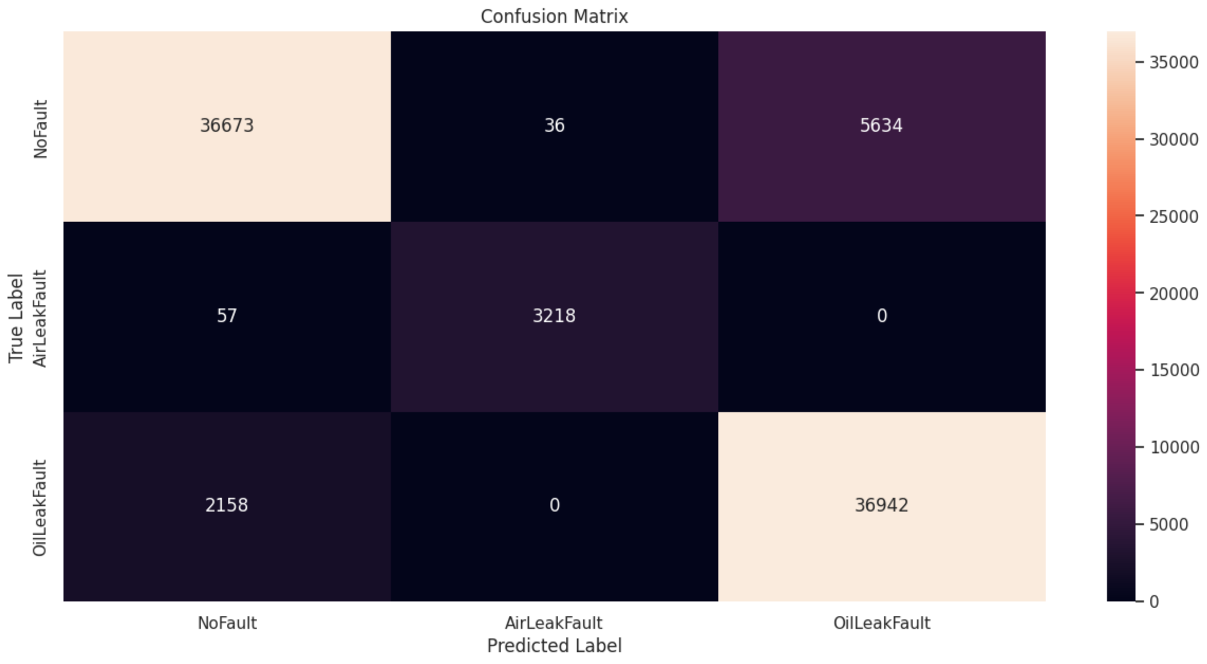


Figure 5.2.3: Random Forest Confusion Matrix

As expected, Random Forest leverages the great performance of Decision Tree and achieves even better results, with an Accuracy of 91%.

False Positives are perfectly avoided for ”AirLeakFault” and the model does so for ”NoFault” too. ”OilLeakFault” Precision is slightly lower compared to the other classes but it is still in an satisfactory percentage.

True Positive predictions are very well predicted across all classes. Interestingly, the scores are ”symmetrical” around the ”AirLeakFault” label. High Precision leads to lower Recall and vice-versa for the 2 remaining states.

Regarding F1-score, it all comes together to a very good Precision-Balance balance for ”NoFault” and ”OilLeakFault” states and an excellent balance for ”AirLeakFault” predictions.

The Random Forest classifier is a robust performer for this problem and outperforms the previous models. So far, it predicts the ”AirLeakFault” class more accurately than the others.

5.2.5 Gradient Boosting Classifier

Gradient Boosting will be used in this case as a Classifier. (see Section 4.2.5: Gradient Boosting Regressor)

```
from sklearn.ensemble import GradientBoostingClassifier

gb_model = GradientBoostingClassifier(n_estimators=10, learning_rate=0.05,
max_depth=5, random_state=42)
gb_model.fit(X_train_scaled, y_train_flat)

# Test set predictions
y_pred = gb_model.predict(X_test_scaled)

print_metrics('Gradient Boosting', target_names, y_test_flat, y_pred)

```

Label	Accuracy	Precision	Recall	F1-score
NoFault	0.79	0.84	0.72	0.78
AirLeakFault		0.99	0.73	0.84
OilLeakFault		0.74	0.88	0.81

Table 5.4: Gradient Boosting Classifier Model Results

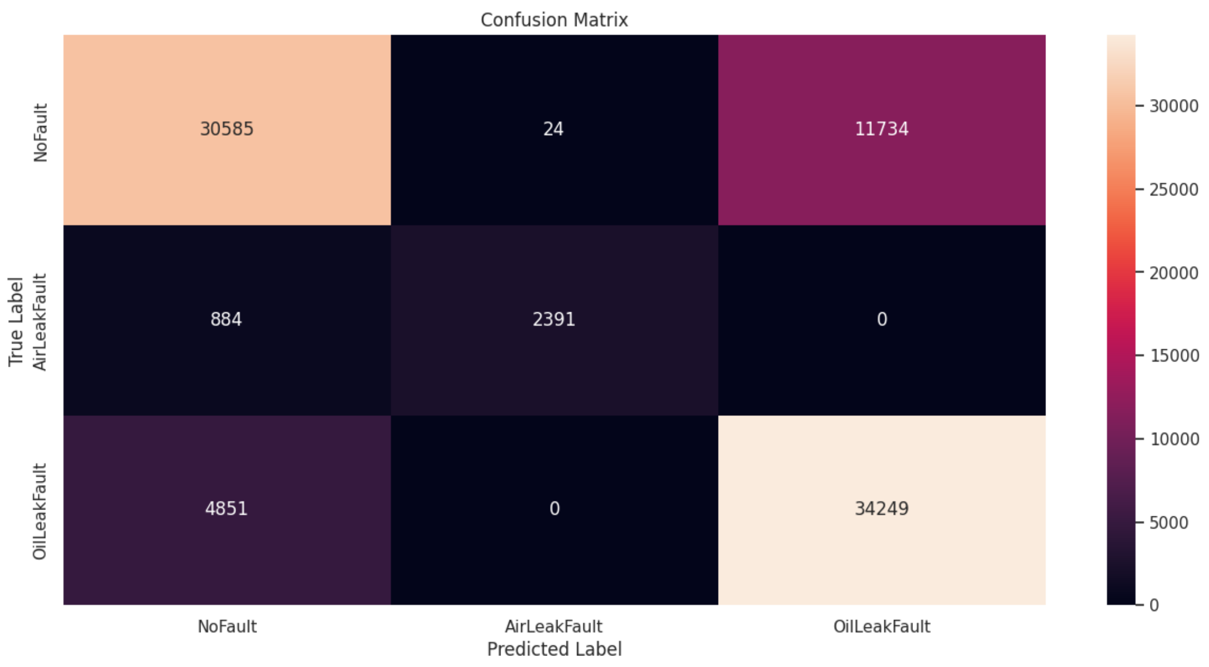


Figure 5.2.4: Gradient Boosting Confusion Matrix

Even though Gradient Boosting is using Decision Trees at its core, it is performing worse than them. In fact, its results are similar to Logistic Regression.

Once again, the ”AirLeakFault” predictions are more precise than in any class. Regarding ”OilLeakFault” class, False Positive rate is not that much satisfactory.

Recall’s low percentages indicate a not so good performance because True Positives are not as high as expected, especially when compared to the rest of the models.

About F1-score, for ”NoFault” state, the 74% score represents a somewhat reasonable balance between Precision and Recall, but it is not acceptable when we take into account the standards that are set from previous models.

Overall, Gradient Boosting could not be the algorithm of choice for this problem, but let’s see how its ”enhanced version”, XGBoost, performs.

5.2.6 XGBoost Classifier

XGBoost will be used in this case as a Classifier. (see Section 4.2.6: XGBoost Regressor)

```
    import xgboost as xgb

    xgb_model = xgb.XGBClassifier(objective='reg:squarederror', n_estimators
    =80, learning_rate=0.1, max_depth=64, random_state=42)
    xgb_model.fit(X_train_scaled, y_train_flat)

    # Test set predictions
    y_pred = xgb_model.predict(X_test_scaled)

    print_metrics('XGBoost', target_names, y_test_flat, y_pred)

```

Label	Accuracy	Precision	Recall	F1-score
NoFault	0.91	0.94	0.88	0.91
AirLeakFault		0.99	0.99	0.99
OilLeakFault		0.88	0.94	0.91

Table 5.5: XGBoost Classifier Model Results

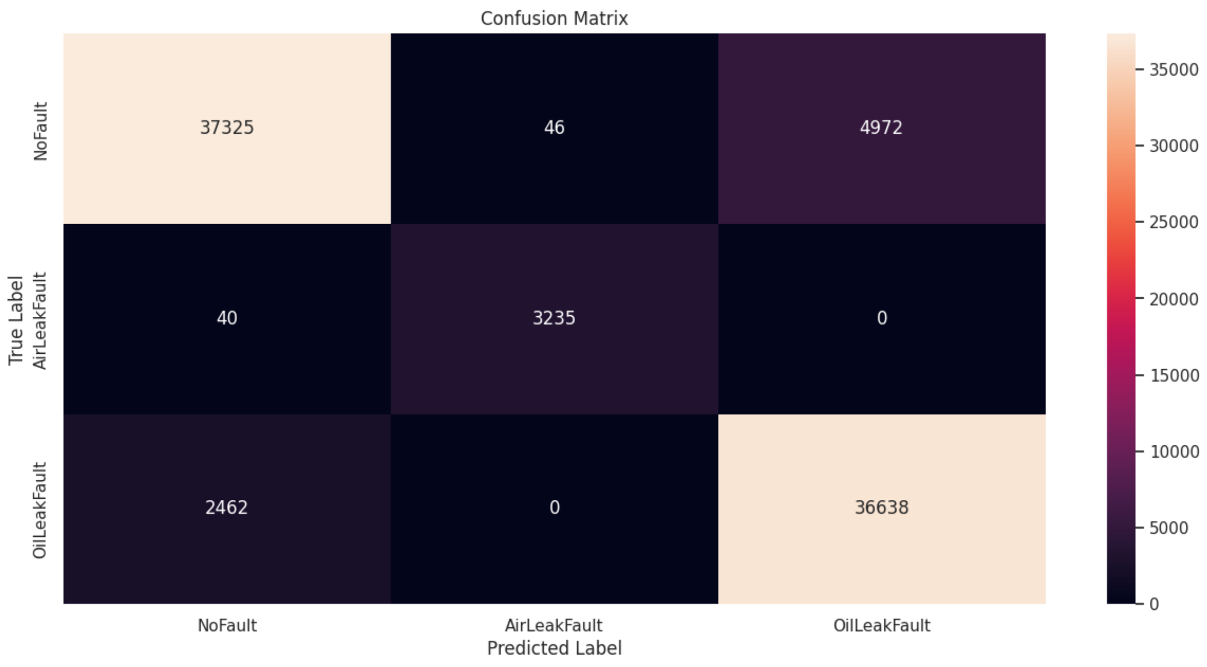


Figure 5.2.5: XGBoost Confusion Matrix

XGBoost performs exceptionally well with an Accuracy of 91%, reaching the same level as Random Forest. ”AirLeakFault” is perfectly precise once again, while the remaining 2 classes are greatly predicted as well, in terms of avoiding False Positives.

True Positives of ”AirLeakFault” are at a perfect percentage. The recall percentage is high for the rest of the states.

XGBoost slightly outperforms Random Forest when it comes to F1-score. Even though the difference is miniscule, we can say that it does the best job so far.

5.2.7 Deep Neural Network

Dense Deep Neural Networks are a type of Artificial Neural Network architecture commonly used for multiclass Classification tasks. These networks consist of multiple layers of neurons organized sequentially, where each neuron in one layer is connected to every neuron in the subsequent layer, as the term ”Dense” indicates.

In a Sequential Dense DNN, the input layer receives the features of the input data, and subsequent hidden layers process and transform this information through a series of non-linear transformations. Each hidden layer applies an activation function to the weighted sum of inputs from the previous layer, allowing the network to learn complex patterns from the data. Finally, the output layer produces the predictions for each class.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization

nn_model = Sequential([
Dense(32, activation='relu', input_shape=(len(features),)),
Dense(16, activation='relu'),
Dense(len(target_names), activation='softmax'),
])

nn_model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])

history = nn_model.fit(X_train_scaled, y_train, epochs=50, batch_size=32,
shuffle=True)

loss = history.history['loss']
accuracy = history.history['accuracy']
epochs = range(1, len(loss) + 1)

plt.figure(figsize=(10, 6))
plt.plot(epochs, loss, label='Training Loss')
plt.plot(epochs, accuracy, label='Training Accuracy')

plt.title('Training Loss and Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss / Accuracy')
plt.legend()

plt.grid(True)
plt.show()

```

Training Loss function was decreased to 0.38 and Training Accuracy was 81%.

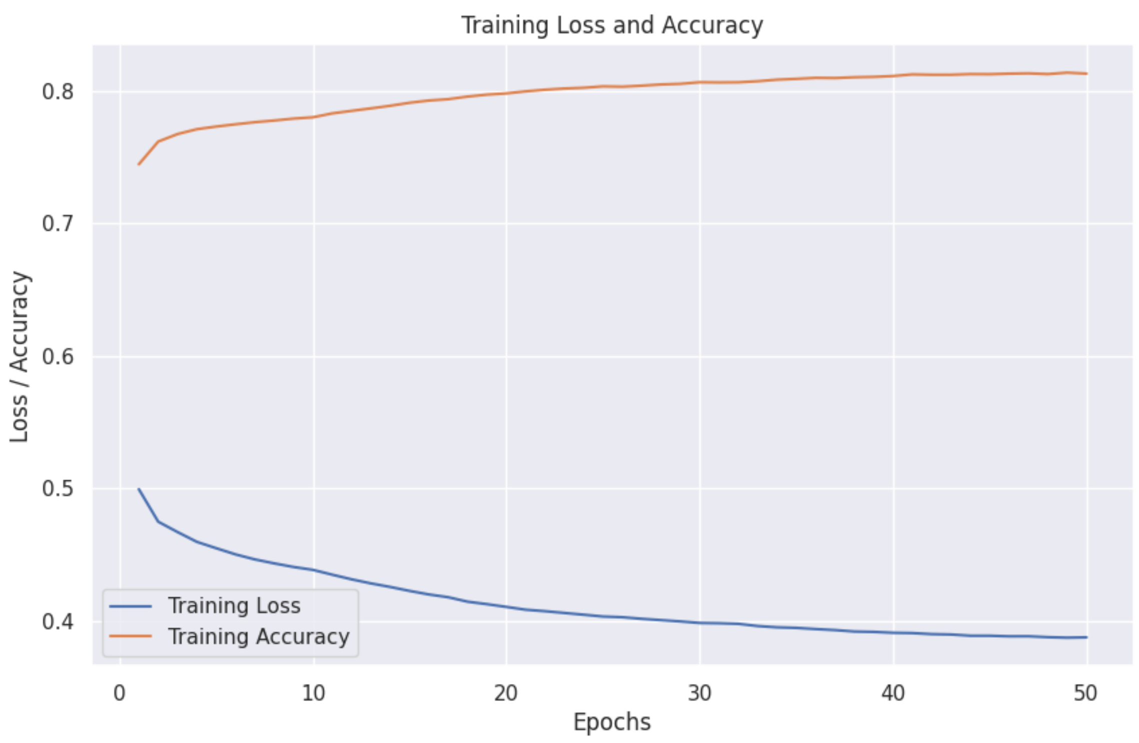


Figure 5.2.6: Training Loss and Accuracy plot

Now to perform predictions and check the metrics and Confusion Matrix.

```
y_pred = nn_model.predict(X_test_scaled)

print_metrics('Deep Neural Network', target_names, y_test_flat, y_pred.
argmax(axis=1))

```

Label	Accuracy	Precision	Recall	F1-score
NoFault	0.81	0.93	0.66	0.77
AirLeakFault		0.88	0.95	0.92
OilLeakFault		0.73	0.95	0.82

Table 5.6: Deep Neural Network Classifier Model Results

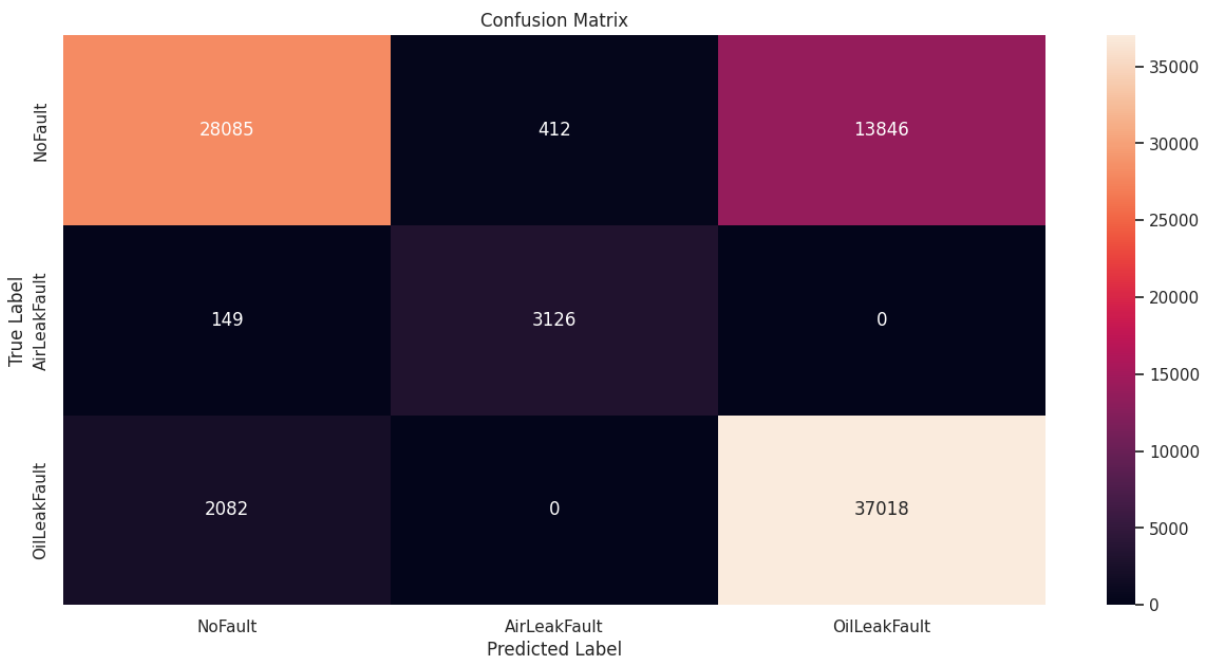


Figure 5.2.7: Deep Neural Network Confusion Matrix

Neural network did not perform that well. It managed to achieve very high Recall percentages for Fault states but it failed in the ”NoFault” state. Its performance is generally reflected in the F1-score. Notice that it performs in a mediocre way for ”NoFault” and ”OilLeakFault” but once again, ”AirLeakFault” is the most accurately predicted class.

More Training data could lead the model to yield better results. This would require to include more sample from the Normal operation state and then employ some short of data augmentation technique for the Fault states’ samples. This approach was not followed in order to fairly explore how all algorithms types perform.

5.3 State Classification Conclusions

Model	**Accuracy**	**Precision (0 / 1 / 2)**	**Recall (0 / 1 / 2)**	**F1-score (0 / 1 / 2)**
Logistic Regression	0.77	0.86 / 0.95 /0.70	0.66 / 0.81 / 0.90	0.74 / 0.87 / 0.79
Decision Tree	0.88	0.89 / 0.98 / 0.86	0.87 / 0.98 / 0.88	0.88 / 0.98 / 0.87
Random Forest	0.91	0.94 / 0.99 / 0.88	0.87 / 0.98 / 0.94	0.90 / 0.99 / 0.90
Gradient Boosting	0.79	0.84 / 0.99 / 0.74	0.72 / 0.73 / 0.88	0.78 / 0.84 / 0.81
XGBoost	0.91	0.94 / 0.99 / 0.88	0.88 / 0.99 / 0.94	0.91 / 0.99 / 0.91
Neural Network	0.81	0.93 / 0.88 / 0.73	0.66 / 0.95 / 0.95	0.77 / 0.92 / 0.82

Table 5.7: Classifier Model Results Summarisation

In this multiclass Classification problem, six (6) different models were evaluated:

- Logistic Regression
- Decision Tree
- Random Forest
- Gradient Boosting
- XGBoost
- Deep Neural Network

Each model was assessed based on its Accuracy, Precision, Recall, and F1-score. The table above summarises the performance of all model's that were tested.

The Logistic Regression model, being a simple algorithm, demonstrated decent performance across all metrics. It provided a reasonable balance between Precision and Recall for each class label. However, its performance was not satisfactory and it was surpassed by more complex models.

The Decision Tree model showed strong performance with high Accuracy, Precision, Recall, and F1-score for all class labels. Decision Trees are known for being able to handle non-linear relationships in the data more effectively, which likely contributed to its performance in this task.

The Random Forest model, being an Ensemble technique, leveraged the good aspects of Decision Trees and performed even better. Accuracy and F1-score reached (or exceeded) the 90% mark for all equipment operation states. It hit the perfect F1-score for ”AirLeakFault” too!

The Gradient Boosting model, another Ensemble learning technique, provided results with good Accuracy and Precision. However, it did not manage to reach the levels of the earlier models with the exception of Logistic Regression.

Then, we can see that the XGBoost model emerged as the top performer, even by a slight margin compared to its ”competitor” (Random Forest). It showcased exceptional Accuracy, Precision, Recall, and F1-score for all class labels. XGBoost is known for its scalability, efficiency, and robustness, making it well-suited for complex classification tasks like the one at hand.

Finally, Deep Neural Network model also demonstrated reasonable performance it could not stand out, especially when Decision Tree more accurately. Deep Neural Networks are capable of capturing intricate patterns in data through multiple layers of abstraction, making them effective for such complex tasks. Therefore, it may offer further improvements with additional Training data, hyper-parameter tuning and optimisation.

The results are further proving that Ensemble techniques tend to perform better than ”simple” models. They are harnessing the collective ”wisdom” of multiple models to achieve superior performance, making them a popular choice. This way, they manage to reduce the variance of predictions, improve generalisation and tend to better capture complex relationships between the samples.

To sum up, each model demonstrated its strengths and weaknesses in tackling this Classification problem. While traditional Machine Learning models offer strong performance, Deep Neural Networks provide an alternative approach with potential for further improvement and exploration. Ultimately, the model of choice is dependent on the requirements of the task, the available computational resources and the desired level of performance.

Chapter 6

CONCLUSIONS

After conducting an extensive batch of experiments of various regression algorithms for estimating Remaining Useful Life (RUL) and classification algorithms for state classification, several notable insights have emerged. Firstly, in the realm of RUL estimation, it's evident that different regression algorithms exhibit varying degrees of effectiveness. While some algorithms may excel in certain contexts, others might outperform them in different scenarios. This highlights the importance of selecting the most suitable algorithm based on the specific characteristics and complexities of the dataset under consideration.

The results suggest that the performance of regression algorithms heavily depends on the quality and relevance of the features used for prediction. Algorithms such as Linear Regression are unable to demonstrate robust performance when dealing with complex, non-linear data. Conversely, more complex algorithms like Random Forest offer superior predictive capabilities when faced with intricate relationships within the data. Feature Engineering and selection play a crucial role in enhancing the overall performance of regression models for RUL estimation tasks.

In contrast, the analysis of classification algorithms for state classification unveils a distinct set of findings. While regression algorithms focus on predicting a continuous numerical value (e.g., RUL), classification algorithms are geared towards categorizing data into predefined states. The results indicate that certain algorithms, such as Random Forest or XGBoost, demonstrate high performance in accurately classifying the states of equipment based on the selected features.

Furthermore, the interpretability of classification algorithms emerges as a notable advantage, particularly in scenarios where explainability is important. Unlike some regression models that might be viewed as ”black boxes,” certain classification algorithms offer transparent decision-making processes, enabling to comprehend the logical path followed behind each classification outcome. This interpretability aspect not only facilitates confidence in the model's predictions but also enables the observer to extract actionable insights from the outputs.

The findings underscore the importance of employing a diverse set of regression and classification algorithms in Predictive Maintenance and system monitoring applications. By leveraging the strengths of different algorithms and tailoring them to the specific requirements of the task at hand, one can harness the full potential of Machine Learning for optimising maintenance schedules and enhancing operational efficiency. Additionally, continued research and experimentation with novel algorithms and techniques hold the promise of further advancing the accuracy and applicability of predictive analytics in industrial settings.

Chapter 7

FUTURE WORK

The study described in the previous chapters can be expanded further. There are more approaches and experiments that could yield even better results in all tasks.

To begin with, it would be interesting to explore the performance of Neural Networks for RUL estimation. It is possible that this algorithm will detect complex patterns more effectively than the rest of the models. Due to technical restrictions, such as limited RAM Memory and GPU not available on-demand (Google Colab was used), this concept was dropped. Therefore, it could be executed in the future with additional resources.

Staying in Neural Network's use, Data Augmentation application could lead to interesting results for the State Classification problem. Usually, when it comes to Neural Networks, the larger the volume of data, the better the performance of it. There are enough samples of Normal operation state, but the Fault state samples could be greatly increased. In the case where Data Augmentation does not improve the models performance, another idea would be to employ Transfer Learning techniques. Choosing a model designed for similar tasks, pre-trained models' knowledge is leveraged, saving a great amount of time to Train and tune the Network from scratch. Also, note that Transfer Learning is observed to perform well in cases where data are limited, as is the case for the Fault states in our dataset.

Regarding the models' hyper-parameters, several attempts and combinations were tested to reach the results presented in the chapters earlier. There is potential for improving though. Testing several values in an excessive and exhaustive manner through a grid-search, could have better tuned models as a result.

Another interesting approach would be to investigate the effect and performance of Ensemble Model Stacking technique. To briefly explain the idea of this method, the first step is to Train a diverse set of Base Models using different algorithms or variations of the same algorithm. Then, each Base Model is trained on the Training set and makes predictions on the Test set. The predictions of all models now serve as Training features along with the original Training data, for a model called Meta-learner. Meta-learner could be any model, such as Logistic Regression, Decision Trees or Neural Networks. Finally, when predictions are needed for unseen data, a similar pipeline is executed. The Base Models make predictions individually which are fed to the Meta-learner, tasked to generate the final prediction. This model stacking takes advantage of the different perspectives of multiple models, each capturing separate of the underlying patterns in the dataset.

Bibliography – References – Online sources

[1] B. Veloso, J. Gama, R. Ribeiro, and P. Pereira, ”MetroPT: A Benchmark dataset for predictive maintenance [Data set],” Zenodo, 2022. [Online]. Available: `https://doi.org/10.5281/zenodo.6854240`

[2] O. Surucu, S. A. Gadsden, and J. Yawney, ”Condition Monitoring using Machine Learning: A Review of Theory, Applications, and Recent Advances,” *Expert Systems with Applications*, vol. 221, p. 119738, 2023. doi: `https://doi.org/10.1016/j.eswa.2023.119738`

[3] A. Theissler, J. Pérez-Velázquez, M. Kettelgerdes, and G. Elger, ”Predictive maintenance enabled by machine learning: Use cases and challenges in the automotive industry,” *Reliability Engineering & System Safety*, vol. 215, p. 107864, 2021. doi: `https://doi.org/10.1016/j.ress.2021.107864`

[4] S. Schwendemann, Z. Amjad, and A. Sikora, ”A survey of machine-learning techniques for condition monitoring and predictive maintenance of bearings in grinding machines,” *Computers in Industry*, vol. 125, p. 103380, 2021. doi: `https://doi.org/10.1016/j.compind.2020.103380`

[5] J. Dalzochio et al., ”Machine learning and reasoning for predictive maintenance in Industry 4.0: Current status and challenges,” *Computers in Industry*, vol. 123, p. 103298, 2020. doi: `https://doi.org/10.1016/j.compind.2020.103298`

[6] T. P. Carvalho, F. A. A. M. N. Soares, R. Vita, R. da P. Francisco, J. P. Basto, and S. G. S. Alcalá, ”A systematic literature review of machine learning methods applied to predictive maintenance,” *Computers & Industrial Engineering*, vol. 137, no. April, p. 106024, 2019. doi: `https://doi.org/10.1016/j.cie.2019.106024`

[7] R. Caruana and A. Niculescu-Mizil, ”An empirical comparison of supervised learning algorithms,” in *Proceedings of the 23rd International Conference on Machine Learning (ICML’06)*, 2006, pp. 161-168.

[8] T. G. Dietterich, ”Ensemble methods in machine learning,” in *Multiple classifier systems*, Springer, Berlin, Heidelberg, 2000, pp. 1-15.

[9] D. H. Wolpert, ”Stacked generalization,” *Neural Networks*, vol. 5, no. 2, pp. 241-259, 1992.

[10] W. Wang and X. Wang, ”A comprehensive survey of data-driven predictive maintenance and machine learning in mechanical systems,” *Mechanical Systems and Signal Processing*, vol. 104, pp. 799-834, 2018.

[11] A. K. Jardine, D. Lin, and D. Banjevic, ”A review on machinery diagnostics and prognostics implementing condition-based maintenance,” *Mechanical Systems and Signal Processing*, vol. 20, no. 7, pp. 1483-1510, 2006.

[12] Y. LeCun, Y. Bengio, and G. Hinton, ”Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436-444, 2015.

[13] J. Schmidhuber, ”Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85-117, 2015.

[14] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[15] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[16] Pandas. ”Documentation.” Retrieved from `https://pandas.pydata.org/docs/`

[17] NumPy. ”Documentation.” Retrieved from `https://numpy.org/doc/stable/`

[18] SciPy. ”Documentation.” Retrieved from `https://docs.scipy.org/doc/scipy/`

[19] Matplotlib. ”Documentation.” Retrieved from `https://matplotlib.org/stable/contents.html`

[20] scikit-learn. ”Documentation.” Retrieved from `https://scikit-learn.org/stable/documentation.html`

[21] TensorFlow. ”Documentation.” Retrieved from `https://www.tensorflow.org/api_docs`