National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Bruna Rafaela Alves Garcia
Student ID: x21245835

School of Computing
National College of Ireland

Supervisor: Arjun Chikkankod

National College of IrelandProject Submission Sheet School of Computing

Student Name:	Bruna Rafaela Alves Garcia
Student ID:	X21245835
Programme:	Data Analytics
Year:	2025
Module:	MSc. Research Project
Supervisor:	Arjun Chikkankod
Submission Due Date:	03/01/2025
Project Title:	Configuration Manual
Word Count:	1050
Page Count:	8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Bruna Garcia
Date:	3rd January 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keepa copy on computer.	□

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Bruna Rafaela Alves Garcia

x21245835

1 Introduction

This document provides a comprehensive overview of the hardware and software configurations essential for the successful implementation of this research project. It encompasses details on the computational environment setup, the dataset used, the libraries and tools required, and the step-by-step process of model preparation and analysis.

The dataset, sourced from the MetroPT3 system, contains time-series sensor readings critical for identifying machine failures. To prepare the data for analysis, preprocessing steps include cleaning, feature engineering with rolling statistics and handling class imbalance while maintaining the temporal integrity of the data. The project employs Random Forest as the predictive model. SHAP is integrated to enhance interpretability, providing both global and local insights into the model's decision-making process. This comprehensive configuration ensures that the research achieves its dual objectives of delivering actionable insights for predictive maintenance and demonstrating the practical value of Explainable AI techniques.

2 System Configurations

The project was implemented under the following configurations:

2.1 Local Machine

Figure 1: System Configuration on Local Machine.

2.2 GPU Configuration

In this research project, all code development is conducted using Google Colab Pro+, Google's computing platform.

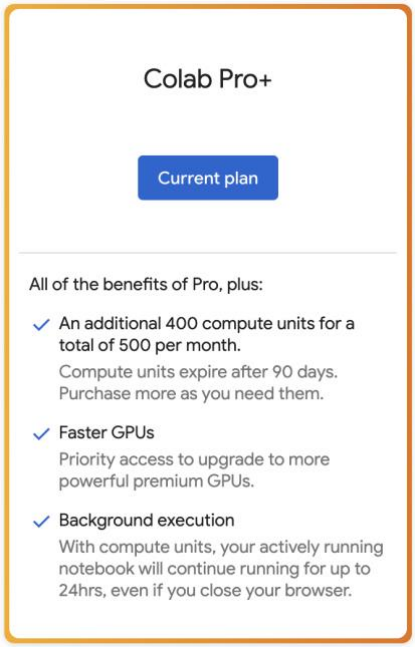

Figure 2: Google Colab Pro+ Specifications.

3 Project Implementation

The project was implemented under the following configurations:

3.1 Data Collection

The dataset, MetroPT3, which is available at UC Irvine Machine Learning Repository, is a time-series dataset collected from the air compressors of Metro do Porto trains. It comprises 15,169,480 data points recorded at a 1Hz frequency from February to August 2020, characterized by 15 features derived from 7 analog and 8 digital sensors. The analog sensors monitor critical metrics such as compressor pressure (TP2), pneumatic panel pressure (TP3), pressure drop during cyclonic separator discharge (H1), and motor current, among others, providing detailed insights into the system's physical operations. The digital sensors capture key operational signals, such as the air intake valve state (COMP), compressor outlet valve status (DV electric), and tower operations for air drying (TOWERS). These combined features comprehensively represent the compressor's functional state and operational environment, enabling effective predictive maintenance analysis.

4 Package Requirements

The following Python libraries were used in the implementation:

- **Pandas**: For data manipulation and preprocessing.
- **NumPy**: For numerical computations.
- **Matplotlib**: For data visualisation.

- **Scikit-learn**: For Random Forest implementation and evaluation.
- **SHAP**: For explainability analysis of the Random Forest model.
- **Google Colab Environment**: For GPU-accelerated model training and SHAP computations.

5 Model Preparation

This section explains the detailed steps involved in preparing the dataset for modelling. The objective is to clean and transform the data to improve the performance of the Random Forest model for predictive maintenance. Each step corresponds to a specific purpose in the pipeline.

Basic data inspection was done as can be seen in Fig. 3.

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1516948 entries, 0 to 1516947
Data columns (total 17 columns):
 #   Column           Non-Null Count    Dtype
---  ------           --------------    -----
 0   Unnamed: 0       1516948 non-null  int64
 1   timestamp        1516948 non-null  object
 2   TP2              1516948 non-null  float64
 3   TP3              1516948 non-null  float64
 4   H1               1516948 non-null  float64
 5   DV_pressure      1516948 non-null  float64
 6   Reservoirs       1516948 non-null  float64
 7   Oil_temperature  1516948 non-null  float64
 8   Motor_current    1516948 non-null  float64
 9   COMP             1516948 non-null  float64
 10  DV_eletric       1516948 non-null  float64
 11  Towers           1516948 non-null  float64
 12  MPG              1516948 non-null  float64
 13  LPS              1516948 non-null  float64
 14  Pressure_switch  1516948 non-null  float64
 15  Oil_level        1516948 non-null  float64
 16  Caudal_impulses  1516948 non-null  float64
dtypes: float64(15), int64(1), object(1)
memory usage: 196.7+ MB
None
```

Figure 3: Dataset information.

The dataset was checked for missing values (Fig. 4) and irrelevant columns were dropped (Unnamed) to ensure relevance.

```
Missing Values Count:
timestamp          0
TP2                0
TP3                0
H1                 0
DV_pressure        0
Reservoirs         0
Oil_temperature    0
Motor_current      0
COMP               0
DV_eletric         0
Towers             0
MPG                0
LPS                0
Pressure_switch    0
Oil_level          0
Caudal_impulses    0
dtype: int64
```

Figure 4: Missing values count.

Based on the failure report provided by the company, four distinct failure events were identified and incorporated into the dataset. A new target feature, labelled as “status,” was introduced to indicate the operational state of the system. For each data point, the "status" was assigned a value of 1 if it corresponded to a failure event, and 0 otherwise, representing normal operation. This binary classification enabled the dataset to be structured for supervised learning, facilitating the prediction of failure events and enhancing the accuracy of the predictive maintenance model. The process ensured that the labelling accurately reflected the real-world scenarios described in the failure report, making the dataset a robust foundation for modeling (Fig. 5).

```
[44] # Defining failure periods according to failure report provided in the dataset documentation file
     failure_periods = [
         {"start": "2020-04-18 00:00:00", "end": "2020-04-18 23:59:59"},
         {"start": "2020-05-29 23:30:00", "end": "2020-05-30 06:00:00"},
         {"start": "2020-06-05 10:00:00", "end": "2020-06-07 14:30:00"},
         {"start": "2020-07-15 14:30:00", "end": "2020-07-15 19:00:00"},
     ]

     # Adding the target variable 'status' column initialized to 0
     data['status'] = 0

     # Marking the rows within failure periods as 1
     for period in failure_periods:
         start_time = pd.to_datetime(period["start"])
         end_time = pd.to_datetime(period["end"])
         data.loc[(data['timestamp'] >= start_time) & (data['timestamp'] <= end_time), 'status'] = 1

     # Verifying the results
     print(data[['timestamp', 'status']].head(10))
     print(data['status'].value_counts())
```

Figure 5: Labelling the dataset.

The class distribution was analysed (Fig. 6) to assess the degree of imbalance in the dataset, highlighting potential challenges in achieving a clean and balanced representation of the target variable.

```
Class Distribution:
status
0    1486988
1      29960
Name: count, dtype: int64
```

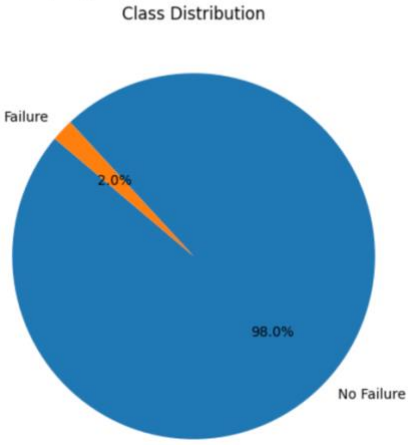

Figure 6: Class Imbalance.

Subsequently, a detailed correlation analysis was conducted to identify features with the strongest relationship to the target feature "status" (Fig. 7).

```
# Separating analog and digital sensors
analog_features = ['TP2', 'TP3', 'H1', 'DV_pressure', 'Reservoirs', 'Oil_temperature', 'Motor_current']
digital_features = ['COMP', 'DV_eletric', 'Towers', 'MPG', 'LPS', 'Pressure_switch', 'Oil_level', 'Caudal_impulses']

# Calculating correlations for analog features
analog_corr = data[analog_features + ['status']].corr()
print("\nAnalog Features Correlation Matrix with Target:")
print(analog_corr['status'])

# Checking correlation among digital sensors
digital_corr = data[digital_features + ['status']].corr()
print("\nDigital Features Correlation Matrix with Target:")
print(digital_corr['status'])

# Combining digital and analog correlations for inspection
combined_features = analog_features + digital_features
combined_corr = data[combined_features + ['status']].corr()
print("\nCombined Features Correlation Matrix with Target:")
print(combined_corr['status'])
```

Figure 7: Calculating correlations.

Features with weak or negligible correlations were removed to reduce noise and improve model efficiency (Fig. 8).

```
# Dropping features with very weak correlations
irrelevant_features = ['Pressure_switch', 'Oil_level', 'Caudal_impulses']
data = data.drop(columns=irrelevant_features)

# Verifying the remaining features
print("Remaining Features after Dropping Irrelevant Ones:")
print(data.columns)
```

```
Remaining Features after Dropping Irrelevant Ones:
Index(['timestamp', 'TP2', 'TP3', 'H1', 'DV_pressure', 'Reservoirs',
       'Oil_temperature', 'Motor_current', 'COMP', 'DV_eletric', 'Towers',
       'MPG', 'LPS', 'status'],
      dtype='object')
```

Figure 8: Dropping features.

To enhance the dataset further, rolling mean and standard deviation values were computed and added for key analog and digital sensor features (Fig. 9), capturing temporal trends and fluctuations critical for time-series analysis. Any missing values introduced during this process were systematically handled using forward and backward fill methods to ensure data integrity.

```
# Define rolling window size
rolling_window = 180

# Adding rolling mean and standard deviation for key analog features
for feature in ['DV_pressure', 'Motor_current', 'Oil_temperature', 'TP2', 'H1']:
    data[f'{feature}_rolling_mean'] = data[feature].rolling(window=rolling_window, min_periods=1).mean()
    data[f'{feature}_rolling_std'] = data[feature].rolling(window=rolling_window, min_periods=1).std()

# Adding rolling mean for key digital features
for feature in ['DV_eletric', 'COMP', 'MPG']:
    data[f'{feature}_rolling_mean'] = data[feature].rolling(window=rolling_window, min_periods=1).mean()

# Verifying the new features
print(data.head())
```

Figure 9: Rolling mean and standard deviation.

The dataset was then split into training and testing subsets, maintaining the sequential nature of the time-series data to preserve temporal dependencies (Fig. 10).

```
# Define a timestamp-based split point
split_timestamp = "2020-06-07 00:00:00"  # Adjust based on the failure report

# Split the dataset into training and testing sets
train = data[data['timestamp'] < split_timestamp]
test = data[data['timestamp'] >= split_timestamp]

# Verify the split
print("Training Set Class Distribution:")
print(train['status'].value_counts())
print("\nTesting Set Class Distribution:")
print(test['status'].value_counts())

# Output dataset shapes
print(f"Training set size: {train.shape}")
print(f"Testing set size: {test.shape}")
```

```
Training Set Class Distribution:
status
0    879787
1     23450
Name: count, dtype: int64

Testing Set Class Distribution:
status
0    607201
1      6510
Name: count, dtype: int64
Training set size: (903237, 27)
Testing set size: (613711, 27)
```

Figure 10: Splitting the dataset.

To address the significant class imbalance, a weighted approach using class weights was implemented during model training (Fig. 11), ensuring the model could effectively learn from the underrepresented class while maintaining overall prediction reliability.

```python
from sklearn.utils.class_weight import compute_class_weight
import numpy as np

# Convert classes to a NumPy array
classes = np.array([0, 1])

# Calculate class weights based on the `status` column
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=classes,
    y=data['status']
)

# Create a dictionary for the weights
class_weights_dict = {cls: weight for cls, weight in zip(classes, class_weights)}

# Display the computed class weights
print("Computed Class Weights:")
print(class_weights_dict)
```

```
Computed Class Weights:
{0: 0.5100740557422118, 1: 25.316221628838452}
```

Figure 11: Class Weights.

5.1 Regression Model

The Random Forest model was employed (Fig. 12) to classify failure events based on the engineered dataset. The model was configured with 100 estimators, a maximum tree depth of 10 to prevent overfitting, and parallel processing to optimize computation. The dataset was split into training and testing subsets, excluding non-informative columns like timestamps. After training, the model's performance was evaluated using precision, recall, and F1-score, which were calculated based on the predictions against the testing set. The model underwent extensive evaluation, including threshold optimisation for improved F1-scores and a grid search for hyperparameter tuning, to handle the challenges of the imbalanced dataset.

```
Random Forest Model Training Completed.

Testing Classification Report:
              precision    recall  f1-score   support

  No Failure       1.00      1.00      1.00    607201
     Failure       0.95      0.73      0.83      6510

    accuracy                           1.00    613711
   macro avg       0.98      0.86      0.91    613711
weighted avg       1.00      1.00      1.00    613711


Top 10 Features by Importance:
                        Feature  Importance
3                  DV_pressure    0.256445
12     DV_pressure_rolling_mean    0.213807
18           TP2_rolling_mean    0.135276
23          COMP_rolling_mean    0.095199
14   Motor_current_rolling_mean    0.075563
24           MPG_rolling_mean    0.057954
16  Oil_temperature_rolling_mean    0.046475
20           H1_rolling_mean    0.028450
22      DV_eletric_rolling_mean    0.027783
6               Motor_current    0.016363
```

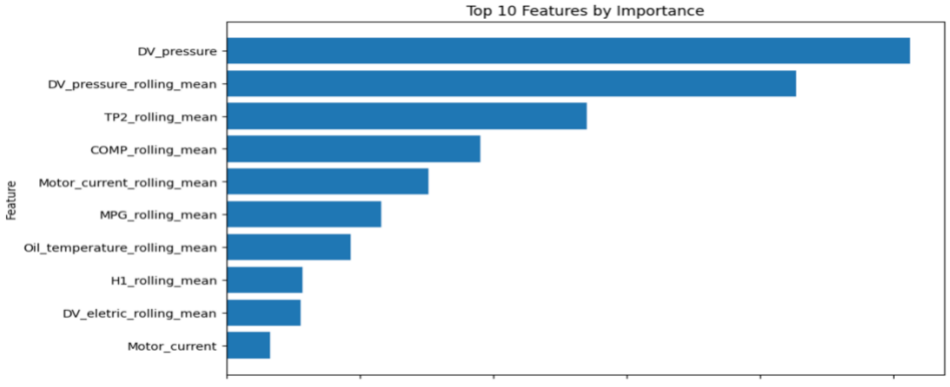

Figure 12: First Random Forest model.

5.2 SHAP

To interpret the predictions made by the Random Forest model, SHAP (SHapley Additive exPlanations) was utilized (fig. 13). SHAP values provided insights into how individual features contributed to specific predictions, enabling a transparent understanding of the model's behavior. The analysis focused on true positive failure cases, where borderline predictions (close to the decision threshold) were selected for detailed examination. The SHAP explainer was applied to these samples, and visualizations such as waterfall plots highlighted the contributions of key features to the predicted outcomes. Furthermore, high-impact features, including sensor readings like TP2, TP3, and DV_pressure, were analysed through density plots and correlation studies to understand their distribution in failure and non-failure scenarios. This interpretability framework ensured the model's decisions were explainable and aligned with domain expectations, enhancing its credibility for deployment in predictive maintenance.

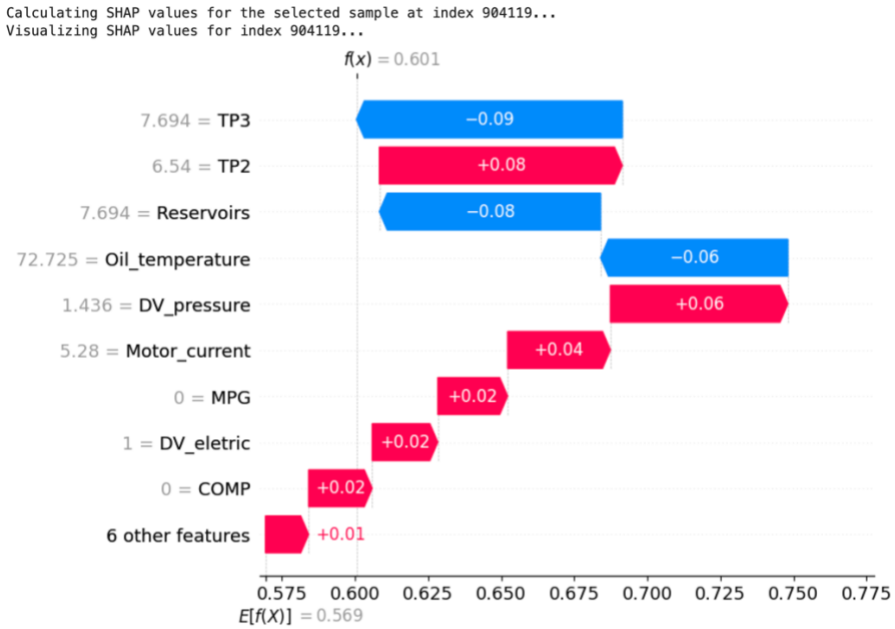

Figure 13: SHAP analysis.