National College of Ireland

Configuration Manual

MSc Research Project
MSc Data Analytics

Pradnesh Amar Shevantikar
Student ID: x23426845

School of Computing
National College of Ireland

Supervisor: Cristina Hava Muntean

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	**Pradnesh Amar Shevantikar**
Student ID:	**23426845**
Programme:	**MSc Data Analytics** **Year: 2025**
Module:	**Research Practicum**
Supervisor:	**Cristina Hava Muntean**
Submission Due Date:	**11th Dec 2025**
Project Title:	**Predictive Maintenance for Industrial Water Treatment Systems Using Real-Time Sensor Data and Machine Learning**
Word Count:	**718** **Page Count**: 9

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: 

Date: 11th Dec 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 keep a 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

Pradnesh Amar Shevantikar
x23426845

1 System Overview and Requirements

1.1 Project Overview:

The following configuration manual describes the environment configuration, dependencies and execution steps to execute the following code artefact to run the MSc Research Project:

The data of this project is not publicly available as it was acquired in a direct acquisition through one of the industrial water-treatment facilities under the condition of confidentiality. Hence, users of this code should take care that the dataset file:

Marcuras Water Treatment.csv

is placed manually at the working directory and followed by the execution of the notebook.

The code to import the dataset is the following one to be executed in the Jupyter notebook:

```
data_path = "Marcuras Water Treatment.csv"

df = pd.read_csv(data_path)

print("Original columns:")
print(df.columns.tolist())
```

The code implements:

- Import, cleaning and engineering of Simulated industrial water-treatment data (40,000+ rows, 52 multistage sensor variables, including pressure, flow, vibration, UV intensity, mineral dosing, temperature, humidity, etc.).
- Feature engineering like Moving averages, rolling windows, slopes and anomaly Sensitive variables
- The supervised predictive-maintenance models:
    1. Logistic Regression
    2. Random Forest
    3. XGBoost (model optimised best)
- Isolation Forest-based anomaly detection without supervision.
- SHAP-based explainable AI to determine major predictors of failure events.
- Export of live-like outputs of risk scores, flags of anomalies, and SHAP values to a Power BI dashboard.

The entire code artefacts are saved in the Jupyter notebook named Thesis_code.ipynb.

1.2 Hardware Requirement:

Component	Minimum Requirement	Actual Environment Used (Project)	Recommended Requirement

Processor (CPU)	4-core CPU (Intel i5 / AMD equivalent)	Local Python Environment	6–8 core CPU
Memory (RAM)	8 GB	16 GB ram used during Processing	16 GB+
Storage	10 GB free space	15 GB Project footprint	25+ GB
Graphics (GPU)	Not required	CPU-based Environment	GPU optional for SHAP acceleration
Operating System	Windows 10/11, macOS, or Linux	Windows 10	Ubuntu 20.04 and above or windows 10 and above
Internet Connection	Required for installing libraries	Stable broadband used	Required

1.3 Software Requirement:

	Requirement	Version / Notes
Programming Language	Python	Version 3.9+
Jupyter Environment	Jupyter Notebook / JupyterLab	Needed to run Thesis code.ipynb
Core Python Libraries	pandas, numpy, matplotlib, seaborn, scikit-learn	Latest stable versions
ML Libraries	XGBoost, SHAP	Used for predictive models and their expalanations
Anomaly Detection	sklearn Isolation Forest	Built-in
Visualization	Matplotlib, seaborn	For EDA and model graphs

1.4 Python Environment Setup

Below is the step of how to import libraries and configure them

pip install pandas numpy scikit-learn xgboost shap matplotlib seaborn

1. Imports and Configuration

[1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import TimeSeriesSplit, RandomizedSearchCV
from sklearn.preprocessing import RobustScaler
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    roc_auc_score,
    precision_score,
    recall_score,
    f1_score,
    RocCurveDisplay,
    PrecisionRecallDisplay
)

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, IsolationForest
from sklearn.cluster import DBSCAN

from xgboost import XGBClassifier

from sklearn.utils.class_weight import compute_class_weight
from imblearn.over_sampling import SMOTE

import shap
import joblib
```

2 Project Structure and Data Dependencies

The entire implementation of this research project was performed in Jupyter Notebook and all the data sets and results files were saved in Local Machine.

The subsections that follow outline the actual folder organization and data interdependence in the project.

2.1 Project Folder Structure

2.1.1 Input Data

Folder / File	Location	Description / Contents
thesis/	thesis/	Main project directory containing all datasets, outputs, and processed files.
Marcuras Water Treatment.csv	thesis/	Raw water treatment dataset obtained from Marcuras Water Systems.

2.1.2 Output Data

Folder / File	Location in Drive	Description / Contents
outputs/	thesis/outputs/	All final processed outputs
dashboard-ready/	thesis/dashboard-ready/	Dashboard-ready export saved to: marcuras_powerbi_data.csv
models/	thesis/models/	Models and scaler saved: rf_tuned_model.pkl,

		xgb_tuned_model.pkl, iso_forest_model.pkl, robust_scaler.pkl

2.1.3 Figures (plots & visualizations)

Folder / File	Description / Contents
Corr_heatmap	Correlation matrix of all features
Feature_importance_rf	Feature importance of Random Forest
ROC comparison	ROC curves of all supervised models
SHAP_plot	SHAP feature contribution plot

2.2 Execution notes for local machine environment

- All files are stored locally and no runtime resets occur
- The jupyter notebook requires relative path of dataset
- No authentication files are required
-

3 Execution Steps and Reproducibility

Here, this section gives the detailed guidelines that will be used in carrying out the project code to achieve all the results.

3.1 Initial Setup

1. Install python 3.9+
2. Install all required libraries
3. Place dataset in the correct location
4. Open the notebook Thesis code.ipynb

3.2 Loading and Preprocessing (Water treatment Plant)

Load dataset

```
data_path = "Marcuras Water Treatment.csv"

df = pd.read_csv(data_path)
```

Feature selection and Preprocessing operations performed
Removed constant and low variance features manually

```
column_map = {
    'RO_Pressure_bar': 'RO Pressure (bar)',
    'Membrane_Pressure': 'Membrane Pressure',
    'UV_Intensity_mW_cm2': 'UV Intensity (mW/cm²)',
    'Mineral_Dose_mg_L': 'Mineral Dose (mg/L)',
    'Discharge_pH': 'Discharge pH',
    'TDS_mg_L': 'Conductivity (µS/cm)',
    'TSS_mg_L': 'TSS (mg/L)',
    'Turbidity': 'Turbidity (NTU)',
    'Pump_Vibration': 'Pump Vibration',
    'Vibration_mm_s': 'Vibration (mm/s)',
    'Motor_Load_Percent': 'Motor Load (%)',
    'Airflow_CFM': 'Airflow (CFM)',
    'Pressure_Drop_bar': 'Pressure Drop (bar)',
    'Reactor_Pressure_bar': 'Reactor Pressure (bar)',
    'Surge_Tank_Level_Percent': 'Surge Tank Level (%)',
    'Temperature_C': 'Temperature (°C)',
    'Ambient_Temperature_C': 'Ambient Temp (°C)',
    'Humidity_Percent': 'Humidity (%)',
    'CO2_ppm': 'CO2 Level (ppm)',
    'O2_Percent': 'O2 Level (%)',
    'Energy_kWh': 'Energy Usage (kWh)',
    'Power_Factor': 'Power Factor'
}

df = df.rename(columns=column_map)
print("\nColumns after renaming:")
print(df.columns.tolist())
```

```
# Select the 22 core sensor features
selected_features = list(column_map.values())

# Keep only timestamp, failure label and selected features
df_model = df[["Timestamp", "Failure"] + selected_features].copy()

# Set time index
df_model = df_model.set_index("Timestamp").sort_index()

print("\nShape of df_model:", df_model.shape)
```

```
Shape of df_model: (40000, 23)
```

Handle missing values

7]:
```
# Time-based interpolation + median fill for any remaining gaps
for col in selected_features:
    df_model[col] = df_model[col].interpolate(method="time")
    df_model[col] = df_model[col].fillna(df_model[col].median())

# Check missing values
print("\nMissing values after interpolation/fill:")
print(df_model[selected_features].isna().sum())
```

```
Missing values after interpolation/fill:
RO Pressure (bar)          0
Membrane Pressure          0
UV Intensity (mW/cm²)      0
Mineral Dose (mg/L)        0
Discharge pH               0
Conductivity (µS/cm)       0
TSS (mg/L)                 0
Turbidity (NTU)            0
Pump Vibration             0
Vibration (mm/s)           0
Motor Load (%)             0
Airflow (CFM)              0
Pressure Drop (bar)        0
Reactor Pressure (bar)     0
Surge Tank Level (%)       0
Temperature (°C)           0
Ambient Temp (°C)          0
Humidity (%)               0
CO2 Level (ppm)            0
O2 Level (%)               0
Energy Usage (kWh)         0
Power Factor               0
dtype: int64
```

Generate rolling window features

4.3 Rolling-Window Features

In [9]:
```
# Define base features and windows for rolling stats
rolling_windows = [30, 60, 180]

base_for_rolling = [
    "Vibration (mm/s)",
    "Pressure Drop (bar)",
    "RO Pressure (bar)",
    "UV Intensity (mW/cm²)",
    "Energy Usage (kWh)"
]

rolling_features = []

for col in base_for_rolling:
    if col in df_model.columns:
        for w in rolling_windows:
            mean_col = f"{col} - Mean({w})"
            std_col = f"{col} - Std({w})"
            slope_col = f"{col} - Slope({w})"

            df_model[mean_col] = df_model[col].rolling(w, min_periods=1).mean()
            df_model[std_col] = df_model[col].rolling(w, min_periods=1).std()
            df_model[slope_col] = df_model[col].diff().rolling(w, min_periods=1).mean()

            rolling_features.extend([mean_col, std_col, slope_col])

# Fill any remaining NaNs in rolling features
df_model[rolling_features] = df_model[rolling_features].fillna(df_model[rolling_features].median())

# Reset index for modelling
df_model = df_model.reset_index()

print("\nTotal features (original + rolling):", len(selected_features) + len(rolling_features))
```

```
Total features (original + rolling): 67
```

3.3 Model Training and evaluation

3.3.1. Supervised Models

XGBoost(Primary Model)
Trained using:

```
# Compute class weights for XGBoost
classes = np.unique(y_train)
class_weights = compute_class_weight(
    class_weight="balanced",
    classes=classes,
    y=y_train
)
class_weight_dict = {cls: w for cls, w in zip(classes, class_weights)}
scale_pos_weight = class_weight_dict.get(1, 1.0) / class_weight_dict.get(0, 1.0)

xgb = XGBClassifier(
    objective="binary:logistic",
    random_state=RANDOM_STATE,
    eval_metric="logloss",
    use_label_encoder=False
)

xgb_param_dist = {
    "n_estimators": [200, 300, 400],
    "max_depth": [3, 5, 7],
    "learning_rate": [0.01, 0.05, 0.1],
    "subsample": [0.7, 0.9, 1.0],
    "colsample_bytree": [0.7, 0.9, 1.0],
    "gamma": [0, 0.1, 0.2]
}
```

Performance metrics:

```
{'Model': 'XGBoost (tuned, class-weight)',
 'Precision': 0.5733333333333334,
 'Recall': 1.0,
 'F1': 0.7288135593220338,
 'ROC_AUC': np.float64(0.4877167741642442)}
```

3.3.2. Isolation forest (Anomaly Detection)

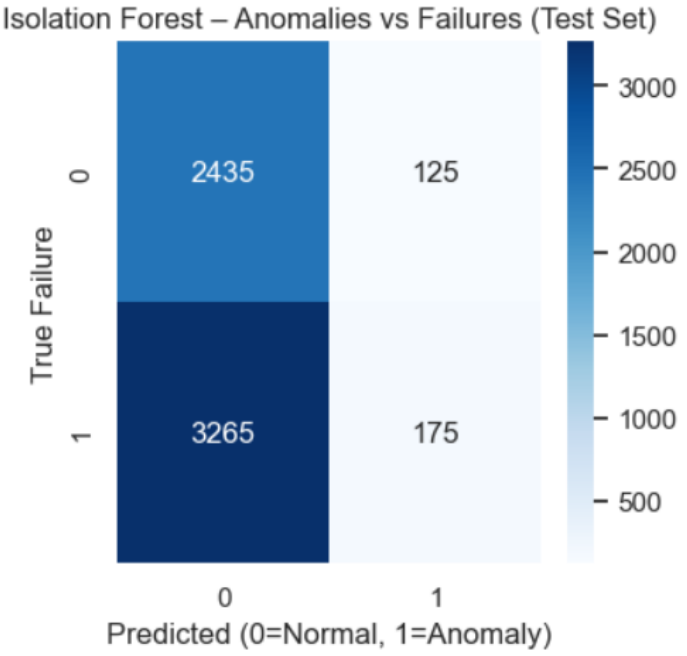
Used to detect early fluctuations

10.1 Isolation Forest

```
# Train IF only on normal training points
X_train_normal = X_train_scaled[y_train == 0]

iso_forest = IsolationForest(
    n_estimators=300,
    contamination=0.05,
    random_state=RANDOM_STATE
)
iso_forest.fit(X_train_normal)
```

```
Isolation Forest anomaly label counts (0=normal, 1=anomaly)
[5700  300]
```



3.3.3. SHAP Explainability

SHAP identifies top predictors:

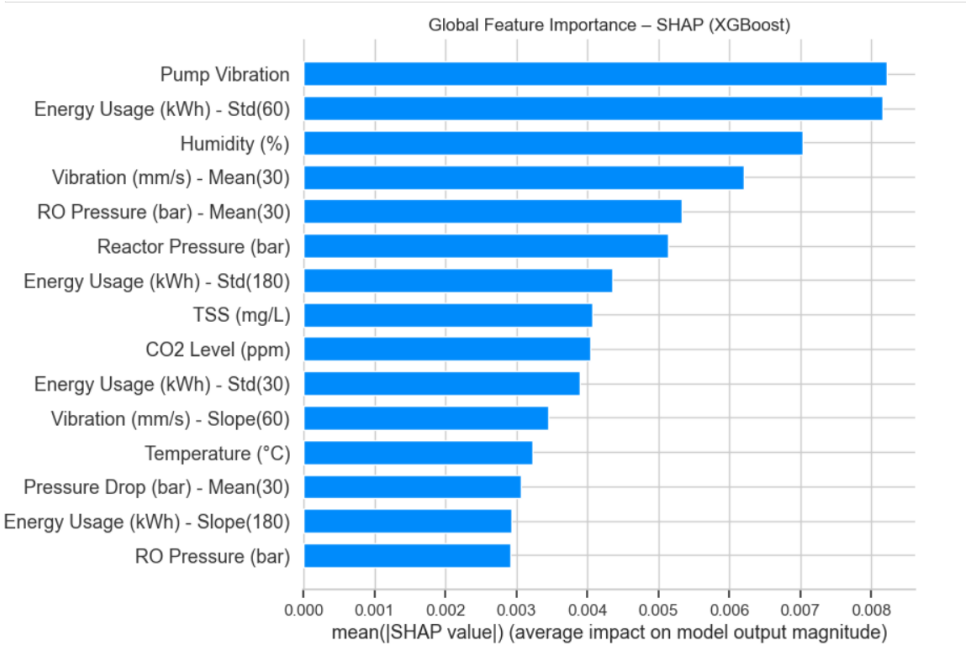
- Vibration
- RO pressure
- UV sensitivity

Implementation:

```python
# Use XGBoost model for SHAP (tree-based and efficient)
explainer = shap.TreeExplainer(xgb_best)

# Sample from training data for speed
sample_size = min(2000, X_train_scaled.shape[0])
sample_idx = np.random.choice(X_train_scaled.shape[0], size=sample_size, replace=False)
shap_sample = X_train_scaled[sample_idx]

shap_values = explainer.shap_values(shap_sample)
```



3.4 Dashboard Output Generation

The Notebook then exports:

- Risk Scores
- Failure probabilities
- Anomaly indicators
- Feature contribution values

These files are later used by PowerBI.

References

1. Python Software Foundation (2024).
   Python Programming Language. https://www.python.org/
   pandas: Python Data Analysis Library. https://pandas.pydata.org/
   NumPy: Fundamental package for scientific computing with Python https://numpy.org/
2. Scikit-learn Documentation (Pedregosa et al., 2011). https://scikit-learn.org/
3. XGBoost Documentation https://xgboost.readthedocs.io/
4. Explainability Library (SHAP) https://shap.readthedocs.io/
5. Industrial Water Treatment Sensor Dataset https://www.marcuras.com/
6. Matplotlib https://matplotlib.org/
7. Seaborn https://seaborn.pydata.org/