Mini Project: Predictive maintenance Model for an Industrial Pump

Project Delivered date: **30th August 2024**

Done by:

M Fazil Ahamed

Senior Manager – Data Science
Shriram Finance Limited,
Chennai, Tamil Nadu, India

Email: fazilahamed@gmail.com
Mobile: +91 77089 83487

Index

Introduction 03

Data Pre-processing 03

Statistical properties

Exploratory Data Analysis 04

Feature Distributions
Correlation Analysis
Box plot by failure status

Feature Engineering 05

Timestamp Decomposition
Rolling Averages
Trend Indicators
Handling missing values
Correlation Analysis

Model Development 07

Splitting the data
Data scaling and train-test split
Addressing imbalanced classes with SMOTE
Model selection

Model Evaluation 09

Classification report analysis
Feature Importance
Jaccard Score
ROC AUC Curve analysis

Model Deployment Simulation 12

FastAPI Application Overview
Testing the API
Deployment Considerations

Introduction

This report presents the solution for the mini project titled "Predictive Maintenance Model for an Industrial Pump." The primary goal of this project was to develop, evaluate, and simulate the deployment of a machine learning model to predict potential failures in an industrial pump based on sensor data.

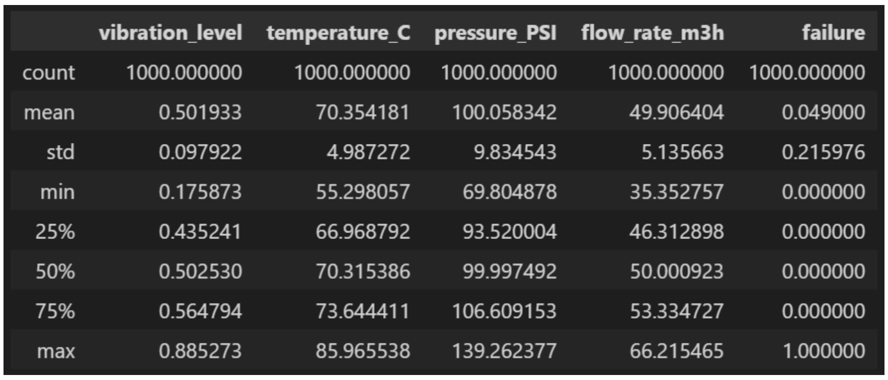
Predictive maintenance is crucial for minimizing downtime and extending the lifespan of industrial machinery. By utilizing sensor data, such as vibration levels, temperature, pressure, and flow rate, along with a binary failure indicator, this project aimed to build a robust predictive model that can forecast pump malfunctions with high accuracy.

Data Pre-processing

The data pre-processing phase is essential for preparing the dataset for modelling. This section outlines the steps taken to load, inspect, and clean the dataset before further analysis.

The given sensor data is loaded into a pandas data frame for easier manipulation and analysis after which, it is checked for presence of null values across the variety of datatypes present in. After confirming that there are no null values, we proceed exploring the data's statistical properties.

Statistical properties



	vibration_level	temperature_C	pressure_PSI	flow_rate_m3h	failure
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.501933	70.354181	100.058342	49.906404	0.049000
std	0.097922	4.987272	9.834543	5.135663	0.215976
min	0.175873	55.298057	69.804878	35.352757	0.000000
25%	0.435241	66.968792	93.520004	46.312898	0.000000
50%	0.502530	70.315386	99.997492	50.000923	0.000000
75%	0.564794	73.644411	106.609153	53.334727	0.000000
max	0.885273	85.965538	139.262377	66.215465	1.000000

Key Insights:

- **Feature Range**: Most sensor readings have a reasonable spread, with no extreme outliers observed in the summary statistics.
- **Imbalance in Failure Data**: The failure variable is significantly skewed towards non-failure, which could pose challenges for model training and require strategies such as balancing techniques to improve predictive accuracy.
- **Potential Relationships**: The variability in features like 'pressure_PSI' and 'temperature_C' suggests that these could be influential factors in predicting failures.

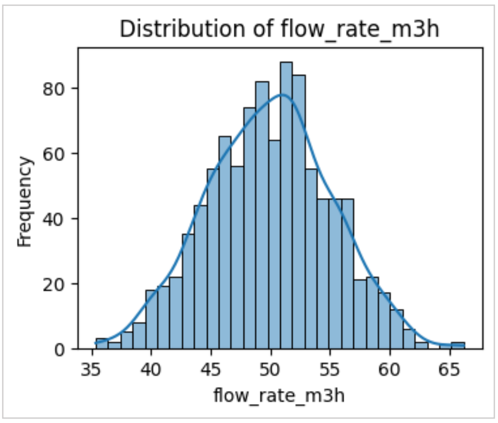
These statistical insights provide a foundation for feature engineering and modelling, emphasizing the need to carefully handle class imbalance and explore relationships among sensor readings to improve the predictive maintenance model.

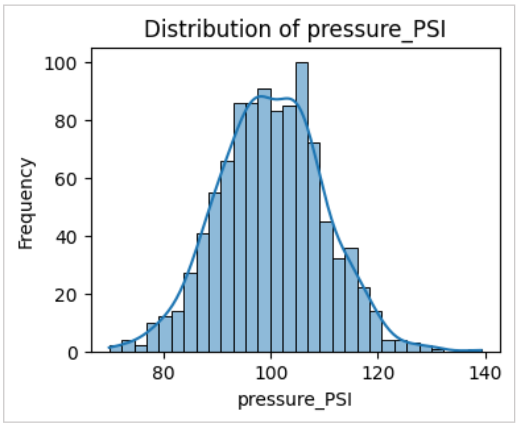
Exploratory Data Analysis (EDA)

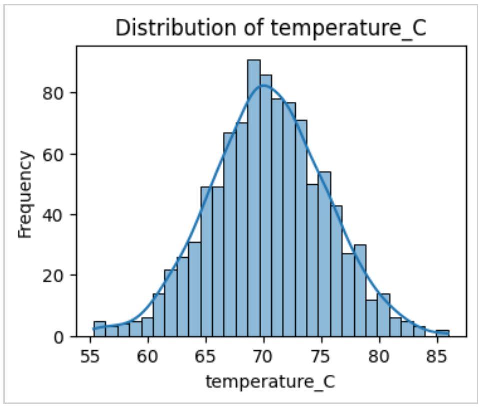
Exploratory Data Analysis (EDA) was conducted to gain insights into the dataset and guide the feature engineering process:

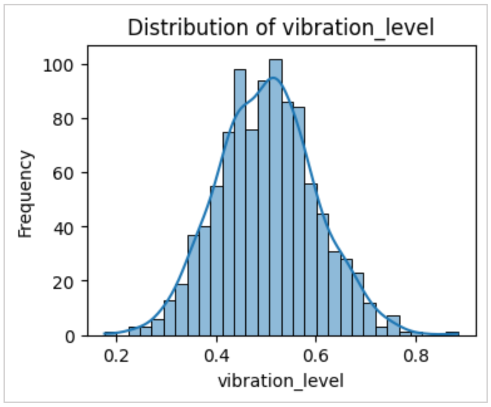
Feature Distributions:

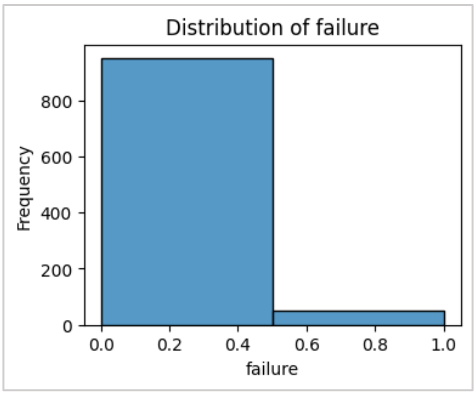
Histograms were plotted for each feature to visualize their distributions. For features like vibration level, temperature, pressure, and flow rate, we used histograms with KDE lines to observe the spread and density. For the binary failure indicator, a simple histogram with two bins was used





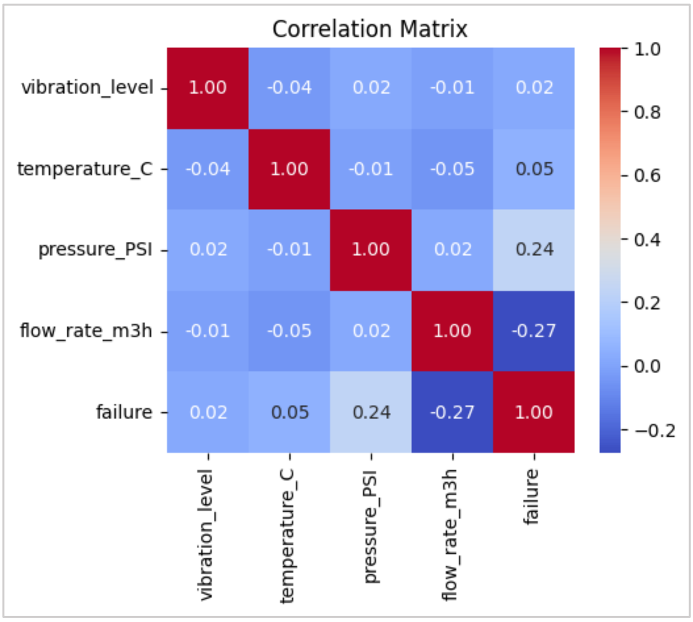






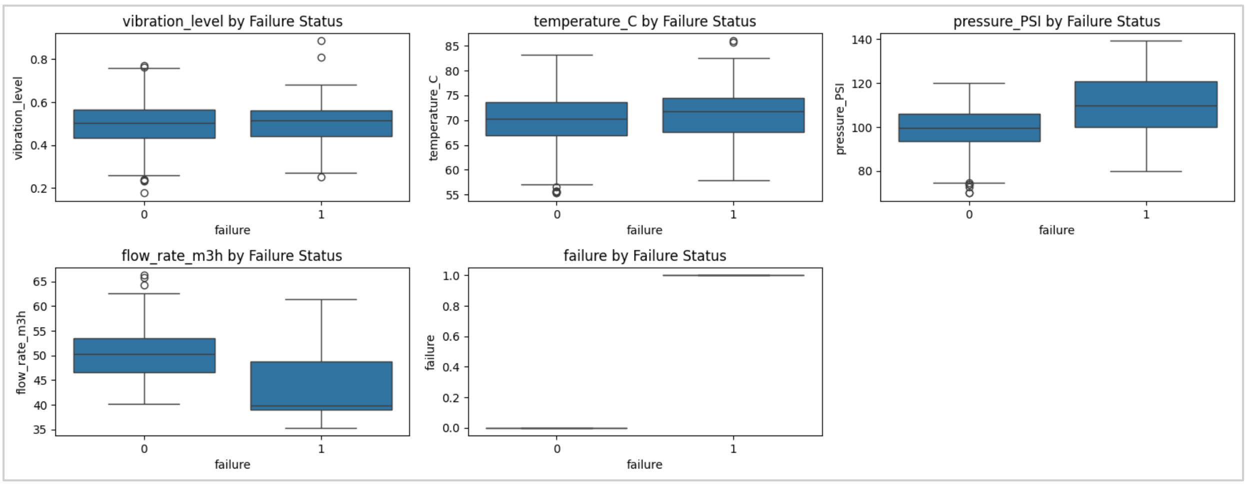
Correlation Analysis:

A correlation matrix was created to assess the relationships between the features. This helps in identifying any multicollinearity and understanding the dependencies between different variables.



Box Plots by Failure Status:

Box plots were generated for each feature to compare their distributions across different failure statuses. This visualization aids in understanding how each feature varies with respect to pump failure.



Key insight:

- **Skewness in the relationship between flow rate and failure**: From the box plots above, it is evident that failure is more common at lower flow rate scenarios.

Feature Engineering

Feature engineering plays a critical role in enhancing the predictive power of machine learning models by creating new variables that better represent the underlying patterns in the data. For this project, several advanced feature engineering techniques were applied to the dataset to capture temporal patterns and trends that could improve the model's ability to predict pump failures.

Key Steps in Feature Engineering:-

Timestamp Decomposition:

The timestamp column, initially a datetime string, was converted into a datetime format to enable the extraction of temporal features that can significantly influence pump performance.

Extracted Features:

- **Hour of Day** (hour): Captures the specific hour when the reading was recorded, allowing the model to learn any time-dependent patterns, such as peak operation times or maintenance periods.
- **Day of the Week** (day_of_week): Indicates the day on which the sensor reading was taken, accounting for weekly operational patterns, such as differences between weekdays and weekends.
- **Week of the Month** (week_of_month): Reflects the week within the month, which can help identify monthly operational cycles or maintenance schedules.

Rolling Averages:

Rolling averages smooth the data and capture recent trends, making them particularly useful for time-series data like sensor readings.

Features Created:

- **Rolling Means**: For the variables vibration_level, temperature_C, pressure_PSI, and flow_rate_m3h, rolling mean values with a window size of 5 were computed. These features represent the average of the last five readings, providing a short-term trend that can highlight gradual changes in pump behaviour.

Trend Indicators:

Trend features were generated by calculating the difference between the current reading and the reading from five steps before (shift(5)). This helps detect sudden changes or anomalies in the sensor values.

Features Created:

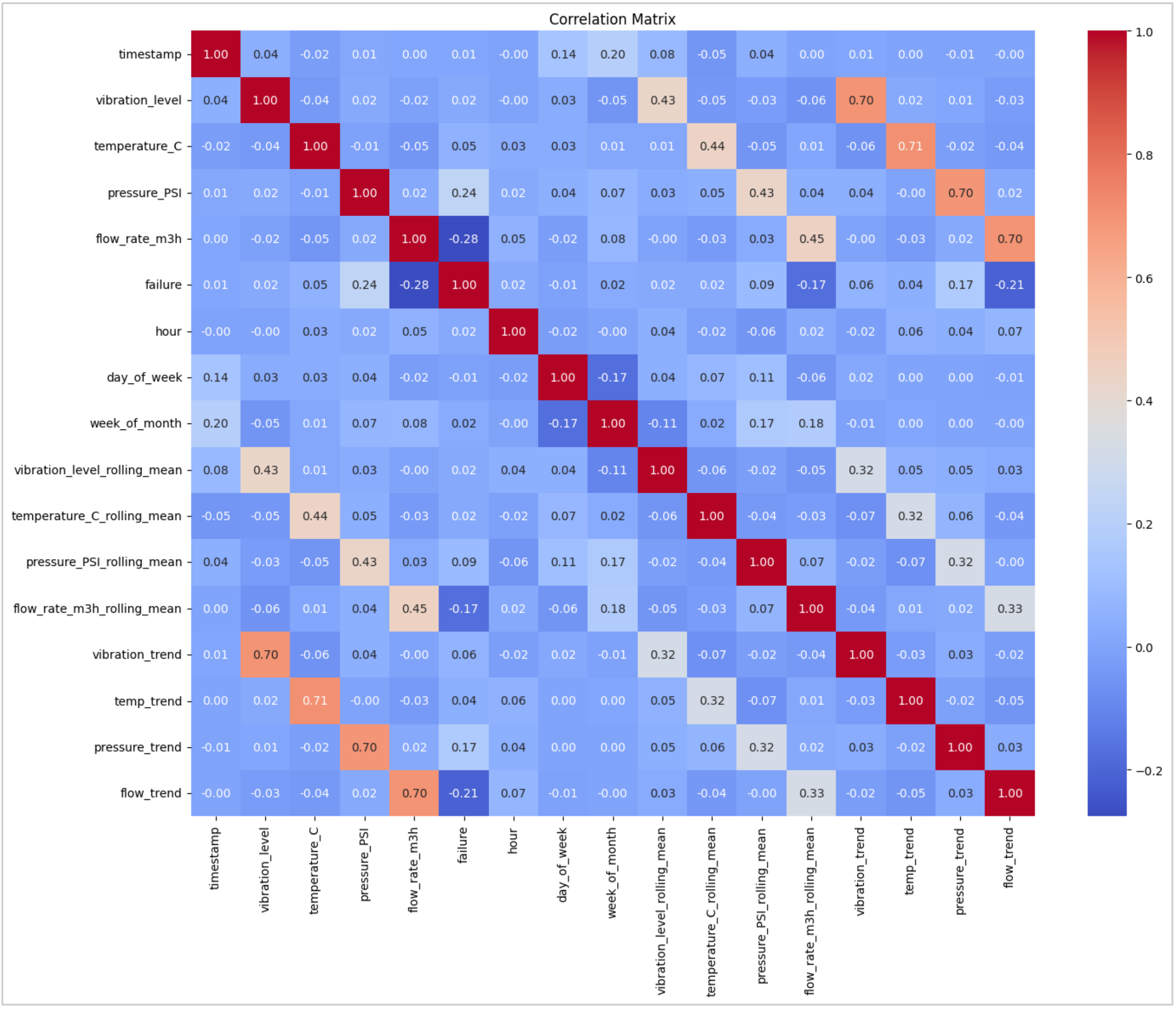
- **Vibration Trend (vibration_trend)**: Measures the change in vibration levels over time, highlighting sudden spikes or drops that could indicate potential failures.
- **Temperature Trend (temp_trend)**: Tracks changes in temperature, which are critical in understanding thermal stresses that may lead to pump degradation.
- **Pressure Trend (pressure_trend)**: Captures the variation in pressure readings, helping identify irregular pressure conditions that could possibly affect pump performance.
- **Flow Rate Trend (flow_trend)**: Reflects changes in the flow rate, which is essential for detecting blockages or leaks.

Handling Missing Values:

After creating these features, rows containing any null values (especially from trends and rolling means) were removed to ensure clean data for modelling. The absence of missing data helps maintain the integrity of trend and rolling mean calculations, which rely on consecutive data points.

Correlation Analysis:

A correlation matrix was generated to visualize the relationships between the newly engineered features and the target variable (failure). The heat map provides insights into which features are most strongly associated with pump failures, guiding feature selection and model tuning decisions.



Key insight:

The Correlation heatmap including the newly engineered parameters does not seem to depict any significantly abnormal correlation against the target variable ‘failure’. Hence, the newly engineered parameters can be retained for further steps.

Model Development

Splitting the Data

The first step in model development was to split the dataset into independent features (X) and the target variable (y). The target variable, failure, indicates pump failure (1 = failure, 0 = no failure), which we aimed to predict based on sensor data and engineered features.

Data Scaling and Train-Test Split

To prepare the data for training, the dataset was split into training and testing sets using an 80-20 split. Feature scaling was considered but not implemented directly in this code to maintain feature interpretability for this initial evaluation phase.

Addressing Imbalanced Classes with SMOTE

One of the main challenges with predictive maintenance data is class imbalance, as failures are relatively rare. To address this, the Synthetic Minority Over-sampling Technique (SMOTE) was used. SMOTE helps balance the class distribution by generating synthetic samples for the minority class, improving the model's ability to detect failures.

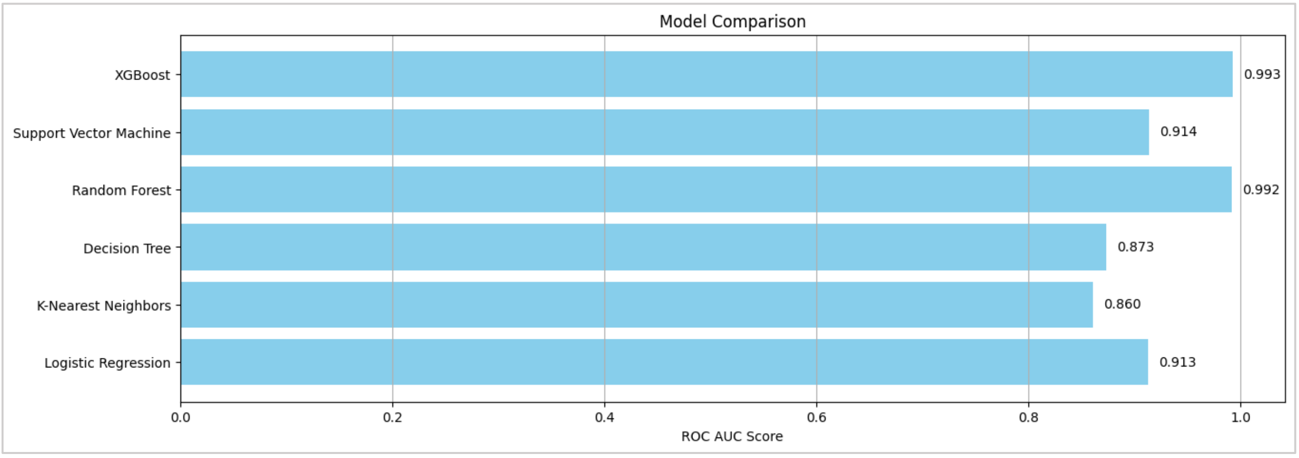
Model Selection

Several classification models were evaluated to determine which performed best on the dataset:

- **Logistic Regression**: A linear model used for binary classification.
- **K-Nearest Neighbours (KNN)**: A non-parametric algorithm that classifies based on the closest training examples in feature space.
- **Decision Tree**: A non-linear model that splits data into subsets based on feature values.
- **Random Forest**: An ensemble of decision trees that improves robustness and reduces overfitting.
- **Support Vector Machine (SVM)**: A model that finds the optimal hyperplane to separate classes.
- **XGBoost**: A powerful gradient boosting model known for its performance in classification tasks.

Each model was trained using the resampled training data from SMOTE and evaluated on the test set using several metrics:

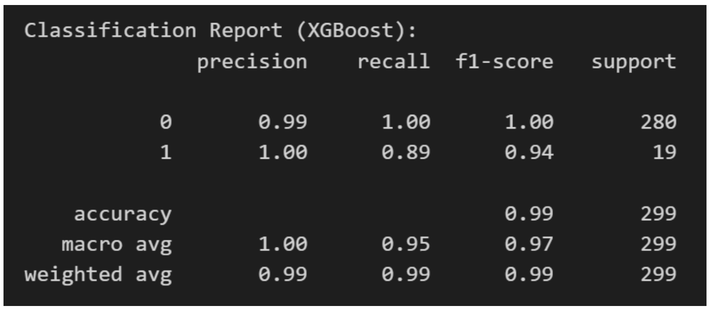
- **ROC AUC Score**: Measures the model's ability to distinguish between classes; higher scores indicate better performance.
- **Precision and Recall**: Evaluated for both classes to understand how well the model identifies failures vs. non-failures.
- **F1 Score**: Combines precision and recall into a single metric, particularly useful for imbalanced datasets.
- **Accuracy**: Overall correctness of the model's predictions.



From this model comparison with respect to the ROC AUC Score, we could converge to the fact that XGBoost has the best performance for the given scenario when compared with other models. Random Forest comes very close, but still, as we aim to build the best possible model, we go for the best case. Hence, we save the model file of XGBoost for further API building and deployment.

Model Evaluation

Classification report analysis:



```
Classification Report (XGBoost):
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       280
           1       1.00      0.89      0.94        19

    accuracy                           0.99       299
   macro avg       1.00      0.95      0.97       299
weighted avg       0.99      0.99      0.99       299
```

The classification report provides a detailed breakdown of the model's performance in distinguishing between the two classes (0 = no failure, 1 = failure). Here's a deeper analysis:

Precision:

Class 0 (No Failure): The precision for class 0 is 0.99, indicating that out of all the instances predicted as 'No Failure', 99% were correctly classified.

Class 1 (Failure): The precision for class 1 is perfect at 1.00, meaning every instance predicted as a failure was indeed a failure. This is crucial for predictive maintenance, where false positives (predicting a failure when there isn't one) can be costly.

Recall:

Class 0 (No Failure): The recall is 1.00, showing that the model identified all the instances of 'No Failure' correctly.

Class 1 (Failure): The recall for class 1 is 0.89, meaning that 89% of the actual failures were correctly identified. While this is strong, there is a 11% miss rate, which could be critical depending on the application.

F1-Score:

Class 0 (No Failure): The F1 score is 1.00, indicating a perfect balance between precision and recall.

Class 1 (Failure): The F1 score is 0.94, reflecting a strong performance in predicting failures but highlighting the slight trade-off between precision and recall.

Accuracy:

The model's overall accuracy is 0.99, meaning that 99% of all predictions (both failures and non-failures) were correct. This high accuracy suggests that the model is reliable in generalizing across different data points.

Macro and Weighted Averages:

Both macro and weighted averages show that the model maintains high performance across both classes, with minor variation, indicating good balance and no significant bias towards the majority class.

Key insights:

- The XGBoost model is highly effective in predicting both normal and failure conditions, with particularly strong performance in recognizing normal states and a slightly lower, yet very much acceptable, recall for failures.

- Given the critical nature of failure detection in predictive maintenance, further optimization or model tuning could focus on enhancing recall for the failure class, possibly by adjusting the classification threshold or incorporating additional features.
- Overall, the model's performance suggests it is well-suited for deployment in predictive maintenance scenarios, balancing the need to minimize false alarms with a high success rate in identifying actual pump failures.

Feature Importance Analysis:

Feature	Importance	Rank
flow_rate_m3h	0.406109	1
pressure_PSI	0.300663	2
temperature_C	0.053488	3
temp_trend	0.050268	4
vibration_trend	0.040062	5
flow_trend	0.032805	6
vibration_level_rolling_mean	0.028657	7
flow_rate_m3h_rolling_mean	0.024077	8
week_of_month	0.015291	9
hour	0.014670	10
vibration_level	0.013261	11
day_of_week	0.006944	12
pressure_trend	0.005793	13
pressure_PSI_rolling_mean	0.004256	14
temperature_C_rolling_mean	0.003658	15

Understanding which features most influence the model's predictions is key to both model interpretability and future improvements:

- **Flow Rate (flow_rate_m3h)**: With the highest importance score of 0.406, this feature plays a crucial role in predicting pump failures. Variations in flow rate are likely strong indicators of potential issues.
- **Pressure (pressure_PSI)**: The second most important feature at 0.301 suggests that pressure levels are also critical to the model's predictions. Sudden changes or anomalies in pressure might correlate strongly with pump failures.
- **Temperature (temperature_C)**: While less influential than flow rate and pressure, temperature still plays a role with an importance of 0.053. This might indicate that extreme temperatures, though not as common, can contribute to pump failures.
- **Trend Features (temp_trend, vibration_trend, flow_trend)**: These engineered features capture the rate of change over time and show moderate importance. This indicates that not just absolute values, but how these values evolve over time, can predict failures.
- **Rolling Means**: Features like vibration_level_rolling_mean and flow_rate_m3h_rolling_mean capture smoothed trends over time, and while they contribute less individually, they collectively enhance the model's ability to detect patterns that might precede a failure.

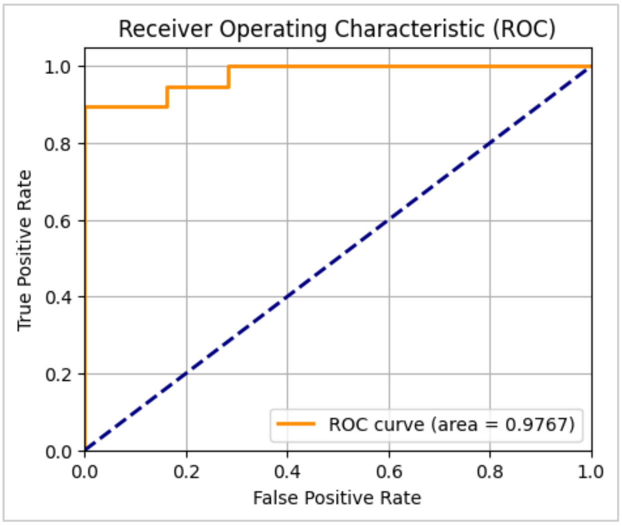
Jaccard Score

The Jaccard Score of **0.8947** reflects a high level of similarity between the predicted and actual labels, particularly for the 'Failure' class. This score indicates that the model does well in predicting true positives without being overly conservative.

ROC AUC Curve Analysis

The ROC (Receiver Operating Characteristic) curve is a graphical representation that illustrates the performance of a binary classification model by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings. The AUC (Area Under the Curve) value represents the model's ability to distinguish between the positive and negative classes.

In our model, the ROC AUC score is **0.9767**, which is close to 1. This high AUC score indicates that our model has a strong ability to separate the classes effectively, meaning it is very good at predicting both pump failures and non-failures. The closer the ROC AUC score is to 1, the better the model's performance, with 0.5 representing a model with no discriminative power (equivalent to random guessing).



The curve itself shows the trade-off between sensitivity (True Positive Rate) and specificity (1 - False Positive Rate). A steep curve that quickly approaches the top left corner of the graph indicates that the model achieves high sensitivity and specificity across most thresholds, which is the case with our model.

Optimal Thresholds

Youden's J Statistic & F1 Score Optimal Threshold: Both optimal thresholds are determined to be **0.7877**. This threshold balances sensitivity (true positive rate) and specificity (true negative rate), as well as precision and recall, ensuring that the model maximizes overall predictive power. Adjusting the threshold to this level could improve the detection of actual failures without significantly increasing false positives.

Model Deployment Simulation

In this section, we demonstrate the deployment of the predictive maintenance model using FastAPI. FastAPI is a modern, fast (high-performance) web framework for building APIs with Python 3.7+ based on standard Python type hints. It is designed to be easy to use and to provide automatic interactive API documentation.

FastAPI Application Overview

The FastAPI application is designed to serve the XGBoost model for real-time predictions. The code for the FastAPI is in the file ‘**api_predict.py**’ for further reference. Here is an overview of the FastAPI application setup:

Model Loading:

The pre-trained XGBoost model is loaded using **joblib.load()**. This model is saved locally as "XGBoost_model.joblib" and is used for making predictions based on incoming data.

API Definition:

Endpoint: /predict

Method: POST

Description: This endpoint accepts sensor data, processes it, and returns predictions along with the probability of failure.

Input Data Model:

Class: InputData

Description: This class, defined using Pydantic, specifies the schema for the input data. It includes various features such as vibration_level, temperature_C, pressure_PSI, and rolling mean values.

Prediction Logic:

Processing: The incoming data is converted into a pandas DataFrame. This DataFrame is then passed to the loaded model to obtain predictions.

Output: The endpoint returns a JSON response containing the predicted class (0 or 1) and the probability of failure.

Testing the API

To test the FastAPI application:

Run the Application:

Start the FastAPI server using uvicorn run command: **uvicorn api_predict:app --reload**, where filename is the name of your Python script.

Access the API:

The API can be accessed locally at http://127.0.0.1:8000 (local host).

FastAPI automatically generates interactive API documentation, which can be accessed at http://127.0.0.1:8000/docs.

Sample Request:

Use tools like Postman or curl to send a POST request to the /predict endpoint with a JSON payload that matches the InputData schema.

The following json payload represents an example of the request to this API:

```
{
 "vibration_level" : 0.25,
 "temperature_C" :  75.0,
 "pressure_PSI" : 120.0,
 "flow_rate_m3h" : 30.0,
 "hour" : 14,
 "day_of_week" : 2,
 "week_of_month" : 3,
 "vibration_level_rolling_mean" : 0.22,
 "temperature_C_rolling_mean" : 74.5,
 "pressure_PSI_rolling_mean" : 119.5,
 "flow_rate_m3h_rolling_mean" : 29.5,
 "vibration_trend" : -0.03,
 "temp_trend" : 0.5,
 "pressure_trend" : -0.5,
 "flow_trend" : 0.2
}
```

Sample Response:

```
{
 "prediction" : 1,
 "prediction_proba" :  0.86,
}
```

The content present in the response json file can be altered as per the business requirement. The above shown content ‘prediction’ and ‘prediction_proba’ is just an example.

Deployment Considerations

Platform Deployment: This FastAPI application can be deployed on various cloud platforms such as AWS, Azure, or Heroku. This will make the model accessible over the internet and integrate it into production environments.

Scalability: Consider implementing scaling strategies and monitoring to handle varying loads and ensure high availability.

Security: Implement security measures such as authentication and authorization to protect the API endpoints.