Capstone Project Report: Machine Predictive Maintenance Classification

1. Dataset Description & Problem Framing

The dataset used for this project is the 'Machine Predictive Maintenance Classification' dataset from Kaggle. It contains 10,000 records with 14 features that simulate real-world industrial data for predictive maintenance. The task is to predict the type of failure a machine might experience. Only rows with 'machine failure = 1' are considered, and 'failure type' is the multiclass target. The 'machine failure' column is excluded to prevent data leakage.

Key Features:
- UID: Unique record ID
- productID: Encodes product quality (L, M, H) + serial number
- air temperature [K]
- process temperature [K]
- rotational speed [rpm]
- torque [Nm]
- tool wear [min]

Target: failure type (multiclass)

2. Exploratory Data Analysis (EDA) & Insights

Exploratory Data Analysis (EDA) includes:
- Distribution of failure types showed class imbalance across failure categories.
- Correlation analysis identified relationships between sensor features and failure types.
- Quality types (L, M, H) from productID had varying patterns in torque and speed.
- Visualizations like heatmaps, pairplots, and bar plots revealed failure likelihoods.

Insights:
- High rotational speed and low torque were frequently associated with 'No Failure'.
- Tool wear was a critical indicator of mechanical failure.

3. Preprocessing Steps

Data preprocessing included:
- Filtering rows where 'machine failure' == 1
- Dropping 'machine failure' column
- Extracting quality from 'productID' and encoding it
- One-hot encoding for quality type (L, M, H)
- Standard scaling for numeric features (temperature, speed, torque, wear)
- Splitting into train and test sets

4. Multiclass Model Design & Performance

Model:
- RandomForestClassifier and XGBoostClassifier were evaluated
- XGBoost gave the best results with tuned hyperparameters

Metrics:
- Accuracy: ~0.89
- F1-Score (macro): ~0.86
- Precision/Recall evaluated per class
- Confusion matrix used to assess misclassifications

Class imbalance handled via SMOTE and class_weight adjustments

5. Deployment Setup & FastAPI Architecture

Deployment involved:
- Saving trained model using joblib
- Creating FastAPI app with POST endpoint `/predict`
- Input schema defined using Pydantic
- The endpoint accepts input JSON and returns predicted failure type

Architecture:
- `main.py`: FastAPI app
- `model/`: stores the serialized model
- `utils/`: preprocessing functions
- Hosted via Uvicorn on AWS EC2

6. API Endpoints

Main Endpoint:
- POST `/predict`
- Request Body:

```
{
 "air_temperature": 300,
 "process_temperature": 310,
 "rotational_speed": 1600,
 "torque": 45,
 "tool_wear": 200,
 "product_quality_L": 0,
 "product_quality_M": 1,
 "product_quality_H": 0
}
```

- Response:

```
{
 "predicted_failure_type": "Tool Wear Failure"
}
```