Project Title: IoT Predictive Maintenance Engine (Time-Series Classification)

Product Brand Name: "FactoryGuard AI"

Use Case (Production): A critical manufacturing plant floor contains 500 robotic arms with vibration, temperature, and pressure sensors. The objective is to predict a catastrophic failure 24 hours before it occurs to allow for scheduled, preemptive maintenance, avoiding millions in unscheduled downtime.

Dataset: NASA Turbofan Engine Degradation Dataset (C-MAPSS)

The NASA Turbofan Engine Degradation Simulation Dataset (also known as C-MAPSS) is a synthetic, run-to-failure time-series dataset developed by NASA for predictive maintenance and remaining useful life (RUL) modeling.

Dataset Entity	Manufacturing Analogy
Turbofan engine	Robotic arm / Machine
Engine cycle	Operating hour
Sensor readings	Vibration, temperature, pressure
Failure point	Machine breakdown
RUL	Time left before failure

>>Files in the Downloaded Folder

1.Training Files:

- Contain full run-to-failure data
- Each engine runs until failure
- Used to train models

2.Test Files:

- Engines do NOT reach failure
- Used for model evaluation
- Corresponding RUL provided separately

3.RUL Files

- Each value = Remaining cycles after last test cycle
- Used to calculate true RUL for test engines

In this Project we will use Dataset Variant-FD001 (single condition, single fault)

Data Structure: Each row represents: One engine at one operating cycle

Total Columns: 26

Column Type	Count
Engine ID	1
Cycle number	1
Operational settings	3
Sensor measurements	21

Column Description

- Identification Columns

Column	Description
engine_id	Unique engine identifier
cycle	Time step / operating hour

- Operational Settings

Column	Meaning
op_setting_1	Environmental / operating condition
op_setting_2	Environmental / operating condition
op_setting_3	Environmental / operating condition

- Sensor Measurements

Sensor Type	Example Meaning
sensor_1	Fan inlet temperature
sensor_2	Pressure
sensor_3	Rotational speed
sensor_4	Fuel flow
sensor_7	Vibration-related
sensor_11	Temperature
sensor_15	Mechanical efficiency

Not all sensors degrade — some are constant or noisy.

Time-Series Nature

- Each engine has different lifespan
- Sensor readings show:
    - Stable behavior initially
    - Gradual degradation
    - Rapid failure close to end

Label Availability

- The dataset does NOT contain: Failure flag or Binary classification label
- What Is Provided:
  - Run-to-failure cycles (training)
  - Remaining Useful Life (RUL) for test set

Why this is ideal?

- Designed exactly for predictive maintenance
- Contains time-series sensor readings
- Includes failure labels

What it contains?

- Multiple engines (similar to robotic arms)
- Sensor readings over time
- Engine gradually degrades until failure
- Can be converted to: Failure in next 24 hours → Yes/No

Task 1: Feature Engineering

Production Requirements & Features: Advanced Rolling Window Statistics: Creating time-series features such as Rolling Mean, Exponential Moving Average, Standard Deviation of sensor readings over the last 1, 6, and 12 hours. Lag Features (t-1, t-2) are essential.

Implementation Details: Focus on Pandas for feature creation; efficient serialization with joblib.

Feature Type	What It Captures
Lag (t-1)	Immediate past behavior
Lag (t-2)	Short-term trend
Rolling Mean	Smoothed behavior
EMA	Recent degradation
Rolling Std	Instability

Efficient Serialization Using joblib

- Saving trained data (features, labels, models) to disk so they can be reused later without recomputation.
- This is critical for production ML systems.

joblib is a Python library used for:

- Fast serialization of large numerical data
- Saving ML models, features, pipelines
- Efficient memory handling

It is optimized for NumPy and Pandas objects.

Task 2: Modeling

Model Selection: Start with a simple Logistic Regression or Scikit-Learn Random Forest as a baseline. The production model must be XGBoost or LightGBM for maximum predictive power.

- **Imbalance Handling**

Class Imbalance is Extreme: Failures are rare (typically <1% of the data). Do NOT use Accuracy. Must use Precision-Recall Area Under Curve (PR-AUC). Handle imbalance using SMOTE or, preferably, by adjusting Class Weights in the model.

Use the imbalanced-learn library. Prioritize High Precision (avoiding false alarms).

- **Hyperparameter Tuning** via GridSearchCV or the advanced Optuna library.

Production Model: XGBoost

Why approach using scale_pos_weight (NO SMOTE) approach is better than SMOTE in production

SMOTE	`scale_pos_weight`
Creates fake samples ✗	Uses real data only ☑
Risk of leakage ✗	Safe for deployment ☑
Memory heavy ✗	Lightweight ☑
Offline training only ✗	Online retraining friendly ☑

Key Observations from Precision–Recall Curve

- PR-AUC = 0.994. This is near-perfect performance

- High Precision Across Most Recall Range

Precision stays ~1.0 for recall up to ~85–90%

This means: Very few false alarms

- Sharp Precision Drop Near Recall ≈ 1.0

Key Observations from Evaluation Metrics (Confusion matrix)

Class 0 (Negative / Majority Class)

Metric	Value	Meaning
Precision	**0.9959**	Almost all predicted negatives are correct
Recall	**0.9926**	Very few false positives
F1-score	**0.9942**	Excellent balance

- The model rarely raises false alarms
- Stable performance on the majority class

Class 1 (Positive / Minority Class – Most Important)

Metric	Value	Meaning
Precision	**0.9511**	~95% of predicted positives are correct
Recall	**0.9720**	**97% of actual positives are detected**
F1-score	**0.9614**	Strong balance between precision & recall

- The model captures almost all minority cases while keeping false positives low