FALCON INFORMATICS

Remaining Useful Life Prediction (RUL)

A CASE STUDY

by Khursheed Fatima

Table of Contents

Case Study Structure

This case study covers the complete development lifecycle: from exploratory analysis and feature engineering, through model training with MLflow experiment tracking, to a modular production-ready pipeline and REST API deployment.

01

Introduction

Background and motivation

Background

In modern industrial settings, unplanned equipment failures are among the most costly operational disruptions. Aircraft engines, industrial turbines, and manufacturing machinery all degrade over time yet traditional maintenance schedules are either reactive (fix after failure) or time-based (replace on a fixed schedule regardless of actual condition). Both approaches are inefficient.

Predictive Maintenance (PdM) addresses this by using real-time sensor data and machine learning to estimate when a machine will fail — enabling maintenance to be scheduled at exactly the right time. The key metric in PdM is the **Remaining Useful Life (RUL)**: how many operational cycles remain before a component requires maintenance or replacement.

Why RUL Prediction Matters

Cost Reduction

- Eliminates unnecessary preventive maintenance
- Avoids expensive unplanned breakdowns
- Optimises spare parts inventory

Operational Efficiency

- Maximises asset utilisation
- Reduces unplanned downtime by up to 30-50%
- Improves fleet-wide scheduling

Safety

- Prevents catastrophic failure in critical systems
- Enables risk-based maintenance prioritisation

Data-Driven Decisions

- Moves from gut-feel to evidence-based planning
- Enables continuous model improvement

About This Project

This project implements a full machine learning pipeline to predict RUL for aircraft engines using the NASA C-MAPSS (Commercial Modular Aero-Propulsion System Simulation) dataset. The work spans the complete development lifecycle — from exploratory data analysis in Jupyter Notebook to a modularised, production-ready codebase with REST API deployment and MLflow experiment tracking.

Technology Stack

Component	Technology	Purpose
Data Analysis	Pandas, NumPy, Jupyter	EDA, feature computation, visualisation
ML Models	Scikit-learn, XGBoost	Regression model training and evaluation
Pipeline	Scikit-learn Pipeline + Pickle	Serialised preprocessing + inference
Experiment Tracking	MLflow	Parameter/metric logging, model registry
Deployment	REST API (Flask/FastAPI)	Real-time RUL prediction endpoint

02

Problem Statement

Dataset, challenge definition and success metrics

> **Goal:** Given multivariate time-series sensor readings from aircraft engines, predict the number of operational cycles remaining before each engine requires maintenance — framed as a supervised regression problem.

The NASA C-MAPSS Dataset

The C-MAPSS (Commercial Modular Aero-Propulsion System Simulation) dataset was generated by NASA using a simulation of a turbofan engine degradation process. Each engine starts in a healthy state with some initial wear variation, and degrades progressively until failure.

Property	Detail
Data Type	Multivariate time-series sensor readings
Input Features	21 sensor measurements + 3 operational settings per cycle
Target Variable	RUL: Remaining Useful Life (cycles to failure)
Train Split	Full run-to-failure trajectories per engine
Test Split	Truncated trajectories - predict RUL at cut-off point
Avg Engine Lifespan	~206 cycles (range: ~130 to ~360 cycles)
Sensors Removed	Constant-value sensors removed (using nunique > 1 check)

Challenges

Data Challenges

- High-frequency sensor noise obscures true degradation signal
- Sensors have varying scales and units requiring normalisation
- Some sensors are near-constant and carry no useful information
- Train/test split structure requires careful preprocessing alignment

Modelling Challenges

- Non-linear relationship between sensor readings and RUL
- Degradation rate accelerates near end-of-life (non-stationary)
- NASA Score metric penalises late predictions more than early
- Need for temporal feature capture (lag features, rolling stats)

Success Metrics

RMSE
Primary metric lower is better

MAE
Mean Absolute Error
Interpretable in cycles

NASA
Penalises late predictions more

R^2
Variance Explained
Baseline: 0.72 (Linear Reg)

03 Pipeline Architecture

End-to-end ML workflow

The project follows a modular architecture where each stage is independently implemented and testable. The same preprocessing pipeline is serialised and reused at inference time preventing data leakage and ensuring production consistency.

01	**Raw Sensor Data**	NASA C-MAPSS train/test CSVs - 21 sensors, operational settings, cycle index
02	**EDA & Cleaning**	Missing value checks, constant sensor removal (nunique>1), train-test analysis
03	**Feature Engineering**	RUL computation, rolling mean smoothing, lag features, consistent scaling
04	**Model Training**	Config-driven - accepts model type + hyperparams, trains, evaluates, saves .pkl
05	**MLflow Tracking**	Logs params (lags, max_depth, rolling_window), metrics (RMSE, MAE, NASA Score)
06	**API Deployment**	REST endpoint loads serialised pipeline, validates input, returns RUL prediction

Module Details

Module	Inputs	Outputs	Key Validation
Preprocessing	Raw CSV data	Scaled + engineered featur	Neso null values, correct dtypes
Training	Processed features		R^2, RMSE, MAE evaluation
Inference	New sensor readings	prediction (cycles)	Column consistency + shape check

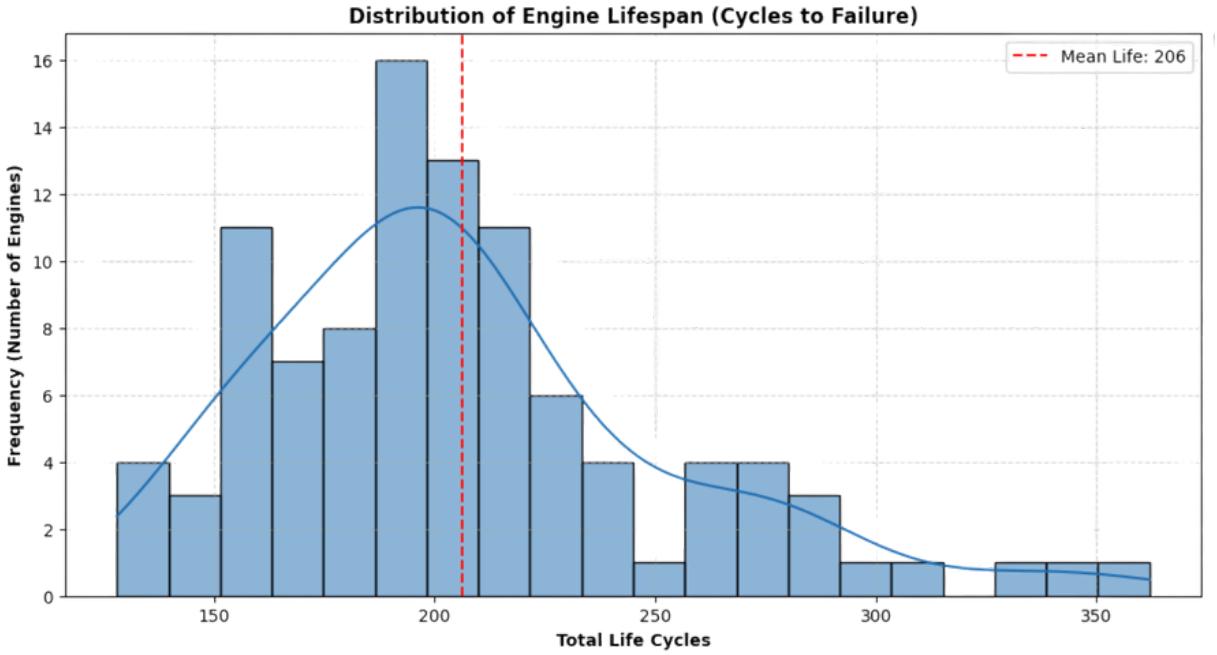
Key design decision: Swapping the underlying model requires only a config change. The same API, preprocessing, and tracking infrastructure remain intact — enabling rapid A/B testing of new models without code changes.

04 Exploratory Data Analysis

Key findings from sensor and lifecycle data

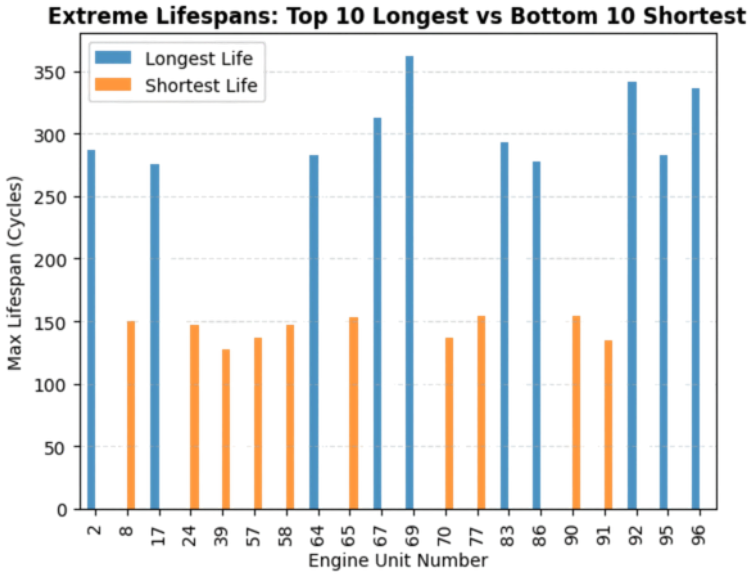
EDA was conducted in Jupyter Notebook to understand data characteristics, validate quality, and guide feature engineering decisions. Key analyses included survival analysis, correlation profiling, and sensor trend visualisation.

Graph 1 - Engine Lifetime Distribution (Survival Analysis)



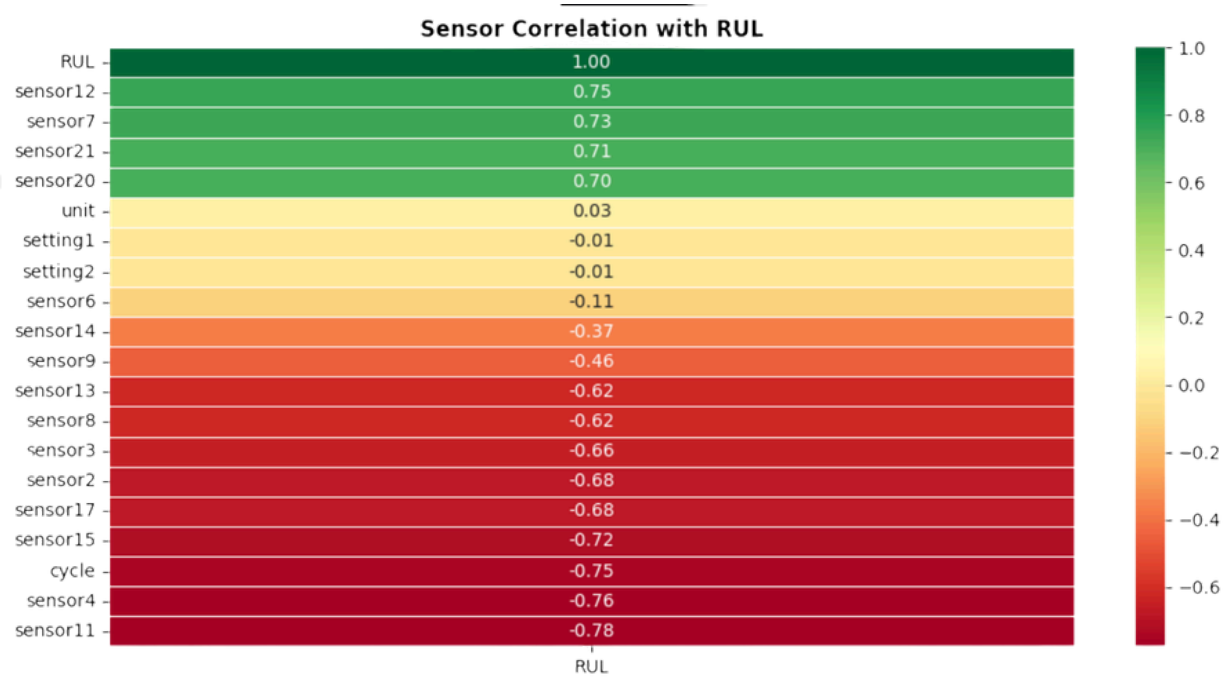
Distribution of total engine cycles before failure. Mean ≈ 206 cycles. Right-skewed — a few engines last significantly longer. Informs the expected RUL output range.

Graph 2 - Shortest vs Longest Engine Lifespans



Engines range from ~130 to ~360 cycles. The same unit ID can appear in both extremes due to cycle-based grouping this is expected behaviour, not a data error.

Graph 3 - Sensor Correlation with RUL (Heatmap)



Sensors s11, s4 show the strongest negative correlation with RUL — rising sensor values signal approaching failure. Dimension reduction was not applied given the limited sensor count and large dataset size.

Graph 4 - Top 4 Sensor Trends with Rolling Mean

ean (window=5) applied per engine unit. Sensors degrade monotonically over cycles, validating their predictive ignal. Raw values show measurement noise, justifying the benefit of smoothing — critical for reliable model training.

Key EDA Finding: Drifts, documents and pdf posts are not bootstrap — similarly, constant sensors (identified via nunique check) provide no signal and were removed before modelling. Active, variable sensors were retained regardless of correlation magnitude to preserve the full degradation pattern.

05

Feature Engineering

Temporal smoothing, lag features and preprocessing decisions

Rolling Mean (Moving Average)

Raw sensor readings are noisy due to operational variation. Rolling mean smoothing was applied per engine unit to capture the underlying degradation trend while suppressing cycle-to-cycle noise.

Decision	Rationale
Window = 5 cycles	Tuned via MLflow experiments — balances smoothing vs. lag
Applied per engine unit	Prevents cross-engine contamination of trends
Applied before scaling	Preserves temporal order; scaling after prevents leakage
rolling_window=[5, 10] tested	Marginal improvement for RF; [5] optimal for XGBoost

Lag Features

Lag features capture temporal dependencies by incorporating past cycle readings as input features. Multiple lag configurations were tested via MLflow experiments.

Lag Config	Model	RMSE	Observation
[1]	XGBoost	20.30	Single lag — limited temporal context
[1]	Random Forest	18.84 – 19.02	Consistent RF results with single lag
[2]	Random Forest	17.72	Slight improvement with 2-cycle lookback
[1, 2, 3, 5]	Random Forest	18.50	Multi-lag RF — more features, marginal gain
[2, 5]	XGBoost	16.85 – 17.85	Best config — two-step lookback captures degradation rhythm

Constant Sensor Removal

Standard approach (std == 0)

- Only removes sensors with exact zero standard deviation
- Misses sensors with fractional constant values
- std can be non-zero for effectively constant sensors

Approach used (nunique > 1)

- Checks number of unique values — catches all constants
- More robust for fractional and near-constant sensors
- Removed sensors visualised post-removal for verification

06

Model Training & Results

MLflow-tracked experiments across XGBoost and Random Forest

21.0	18.50	16.85	20%
Linear Reg (baseline)	Best Random Forest	Best XGBoost ✦	vs LR baseline

All MLflow Experiment Runs

Model	Lags	Max Depth	Rolling Win	RMSE	MAE	NASA Score
XGBoost	**[2, 5]**	**6**	**[5]**	**16.85 ✦**	**11.82**	**815**
XGBoost	[2, 5]	6	[5]	17.00	12.37	909
XGBoost	[2, 5]	6	[5]	17.85	12.90	969
Random Forest	[2]	5	[5]	17.72	12.99	1267
Random Forest	[1,2,3,5]	5	[5, 10]	18.50	13.12	1273
Random Forest	[1]	5	[5]	18.71	14.07	1176
Random Forest	[1]	5	[5]	18.83	14.14	1345
Random Forest	[1]	5	[5]	19.02	14.27	1424
XGBoost (outlier)	[2, 5]	6	[5]	58.95 *	51.05	27363
XGBoost (baseline)	[1]	5	[5]	20.30	15.73	1270
Linear Regression	—	—	—	21.00	—	— (baseline)

✦ Best model selected for deployment · * Outlier run misconfigured early experiment, excluded from selection

Key Insights from Experiments

- Lags=[2,5] outperforms lags=[1] for XGBoost — multi-step lookback captures degradation rhythm
- max_depth=6 optimal for XGBoost; deeper trees led to overfitting
- XGBoost consistently outperforms RF on both RMSE and NASA Score
- rolling_window=[5] sufficient; [5,10] adds marginal value only for RF
- NASA Score rewards early predictions — XGBoost scores significantly lower (better)
- RMSE baseline of 21 confirms strong non-linearity in sensor-RUL relationship

07 Experiment Tracking & Deployment

MLflow setup and REST API inference service

MLflow Experiment Tracking

- Logs hyper parameters per run: lags, max_depth, rolling_window, model type
- Tracks RMSE, MAE and NASA Score for every experiment
- Side-by-side run comparison via MLflow UI
- Best model registered in MLflow Model Registry
- Full reproducibility every run is recoverable
- 13+ runs tracked across XGBoost and Random Forest

REST API Inference Service

- Loads serialised pipeline (.pkl) on startup
- Accepts sensor readings as JSON input via POST /predict
- Applies identical preprocessing pipeline to incoming data
- Validates column consistency and data shape before inference
- Returns predicted RUL (cycles to failure) per engine
- Architecture supports model swaps via config change only

> **Production Architecture Principle:** The preprocessing pipeline is serialised alongside the model as a single .pkl artifact. This guarantees that identical transformations are applied at training and inference time — eliminating a major source of production ML bugs.

08 Key Takeaways & Next Steps

Lessons learned and improvement roadmap

Lessons Learned

- nunique>1 check catches constant sensors missed by std==0 (fractional constants)
- Rolling mean is critical raw sensor data is too noisy for reliable training
- Lag features [2,5] significantly improve XGBoost temporal pattern capture
- Modular pipelines reduce inference bugs and dramatically speed up iteration
- MLflow is essential for managing 13+ experiments — without it, runs become untrackable
- Non-linear models (XGBoost) outperform linear baselines by ~20% on this dataset

Planned Improvements

- Explore LSTM/GRU networks for full sequence-aware RUL prediction
- Add Optuna hyperparameter optimisation integrated with MLflow autolog
- Implement model monitoring for prediction drift detection in production
- Test on multi-condition C-MAPSS subsets (FD002, FD003, FD004)
- Add confidence intervals to API responses for uncertainty quantification
- Explore ensemble stacking of XGBoost + RF for further RMSE reduction

Best Model	Best RMSE	Best MAE	Improvement	Runs Tracked
XGBoost	16.85	11.82	20%	13+

FALCON INFORMATICS

TRANSFORMING BUSINESSES WITH ERP SYSTEMS, DATA SCIENCE, AND INTELLIGENCE

Delivering Data-Driven Insights for Smarter Decisions and Sustainable Growth

Contact Us

www.falconinformatics.com