

# Predicting Remaining Useful Life (RUL) for Boeing's TurboJet Engines using Prognostic Machine Learning Techniques

GitHub Link:

[https://github.com/Pajamajoker/NASA\\_CMAPSS\\_Jet\\_Engine\\_Data\\_Predictive\\_Maintainance](https://github.com/Pajamajoker/NASA_CMAPSS_Jet_Engine_Data_Predictive_Maintainance)

## 1. Introduction

### a. Stakeholder Overview (Who is the stakeholder?)

The key stakeholder is the Technical Director within Boeing's Engineering and Maintenance Division, overseeing the Predictive Maintenance Team. This team includes not just Machine Learning Engineers and Data Scientists but also System Engineers and Aerospace Engineers that are SMEs with domain knowledge of engine/part mechanics & their functionings. The team also includes the Maintenance, Repair, and Overhaul (MRO) Manager.

This team's core responsibilities include:

- Developing and implementing predictive maintenance strategies for Boeing aircraft.
- Analyzing aircraft data to predict maintenance needs and generate proactive alerts.
- Collaborating with engineering, data science, and operator teams to optimize maintenance processes and improve aircraft reliability.

### b. Problem Statement (What problem are they trying to solve?)

The MRO Manager needs an accurate prediction of each engine's Remaining Useful Life (RUL) to plan maintenance before failure, reduce downtime, and cut costs associated with unscheduled repairs.

### c. Why is this important?

- **Safety:** Prevent catastrophic engine failures in flight.
- **Cost efficiency:** Schedule maintenance **just-in-time**, avoiding both premature over-maintenance and expensive unplanned repairs.
- **Operational reliability:** Maximize aircraft availability by minimizing groundings.

## 2. Dataset Description

### a. Data Source and Accessibility (Where is the dataset from?)

- NASA's CMAPSS (Commercial Modular Aero-Propulsion System Simulation) turbofan engine degradation dataset, FD001 subset.
- Publicly available at: <https://data.nasa.gov/dataset/cmapss-jet-engine-simulated-data>

## b. Dataset Structure

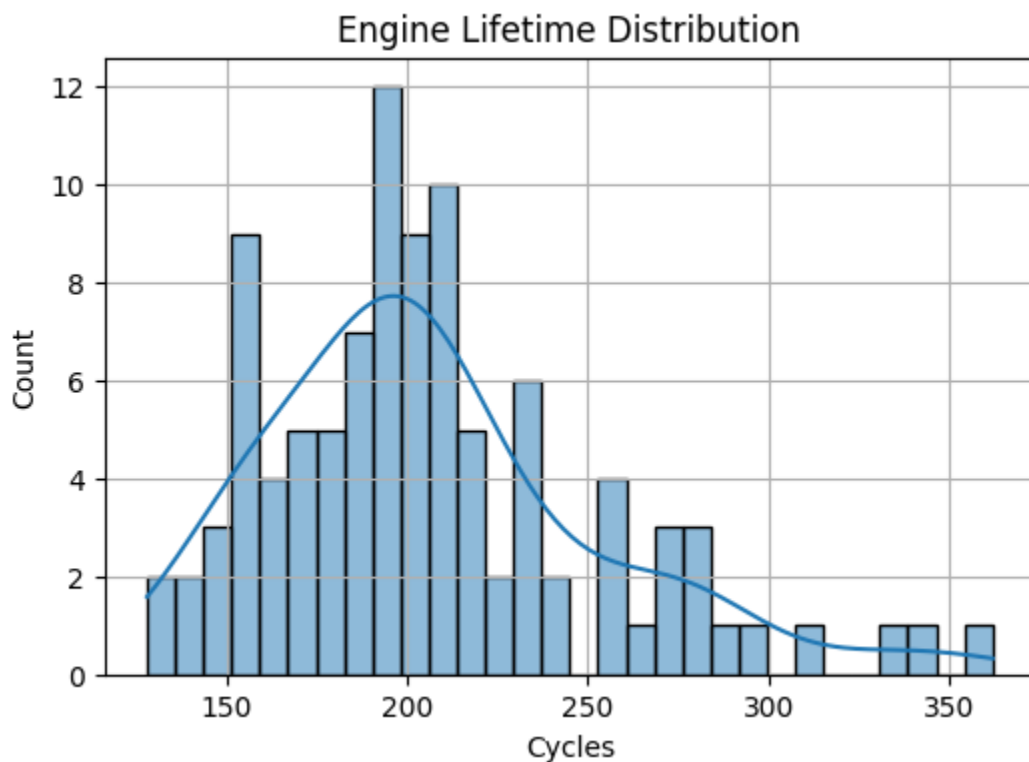
File	Rows Columns	×	Description
train_FD001.txt	20631 × 26		Time-series sensor readings for 100 engines until failure
test_FD001.txt	13096 × 26		Sensor readings for 100 engines up to an unknown cycle
RUL_FD001.txt	100 × 1		True RUL at last cycle of each test engine

## c. Key Features

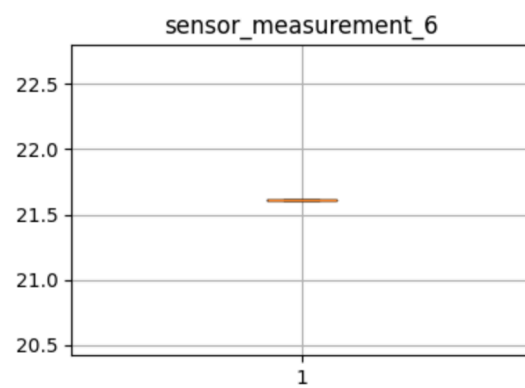
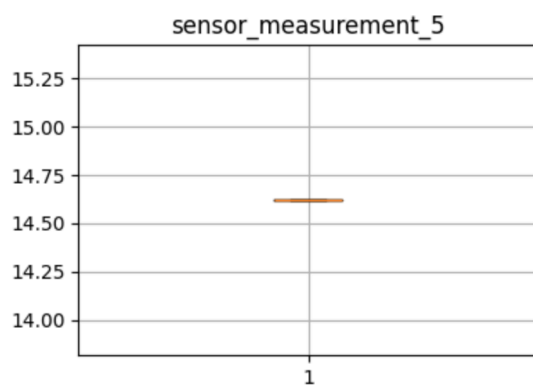
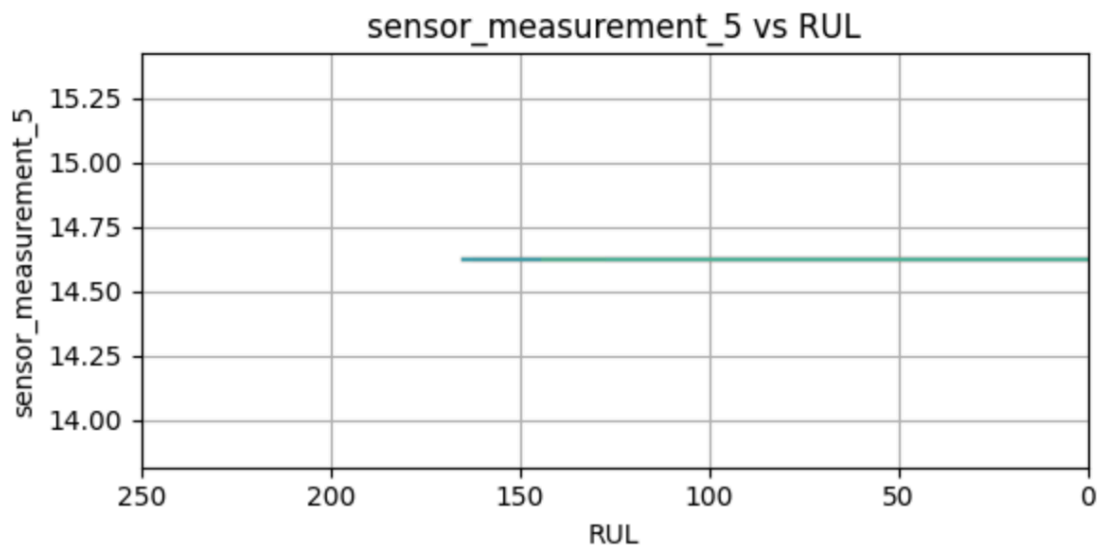
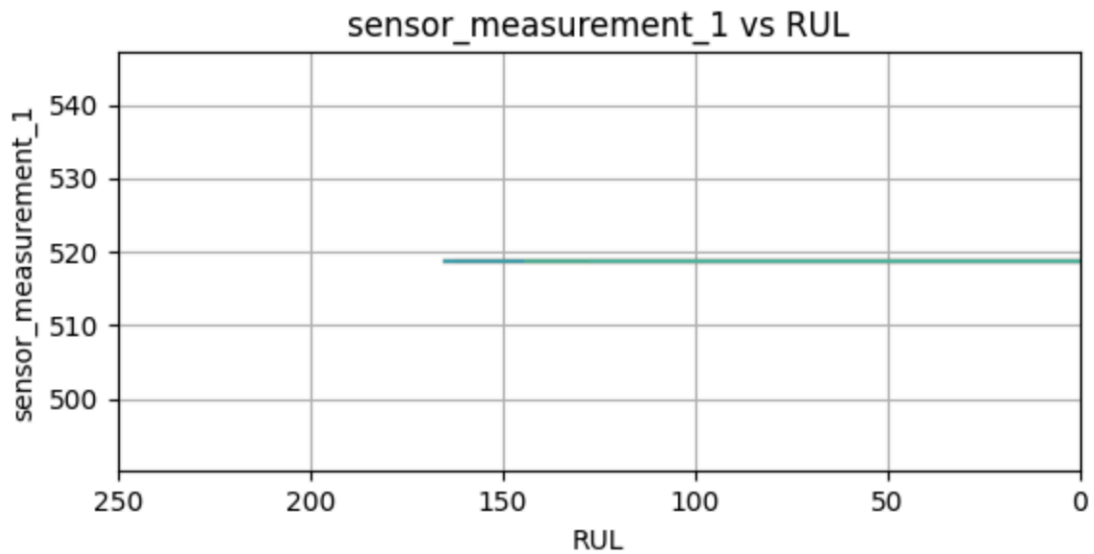
- **Index:** unit\_number, time\_in\_cycles
- **Settings:** operational\_setting\_1,2,3
- **Sensors (1–21):** various temperature, pressure, speed readings

## d. Initial Data Exploration

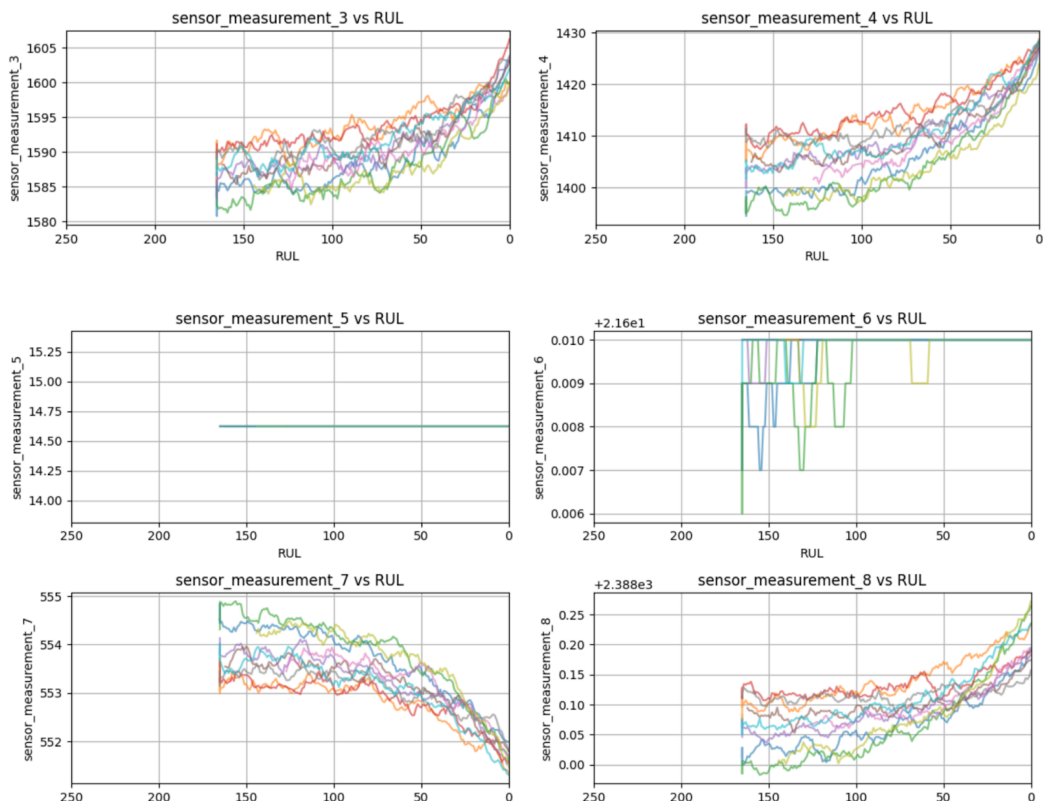
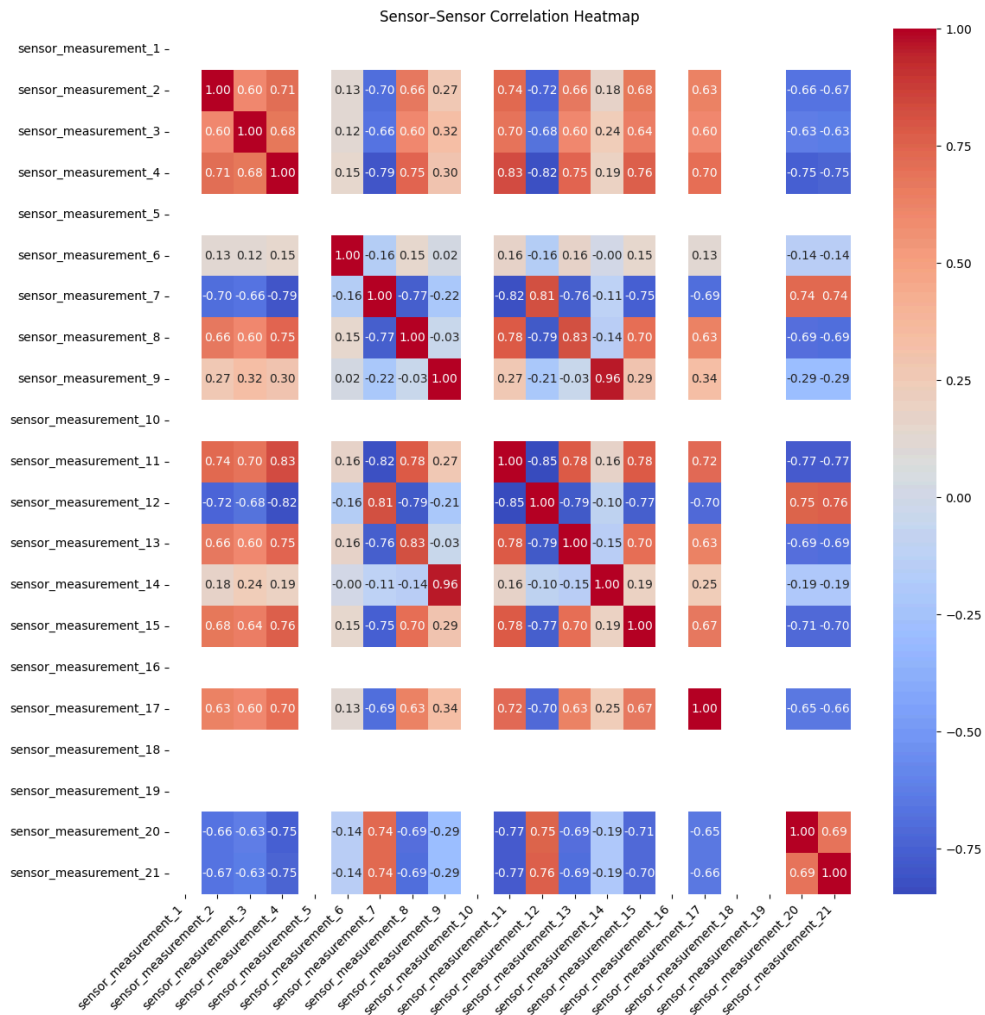
- Engines run between ~128–362 cycles before failure.



- Many sensors are constant (e.g. sensor\_1,5,10,16,18,19) and provide no degradation signal.



- Strong correlations among certain sensors (e.g. sensor\_3 & sensor\_4).



### 3. Feature Engineering (What features did you select/engineer? How did you choose those?)

#### a. Sensor Selection

- **Dropped** sensors with near-zero variance (1,5,10,16,18,19) because they carry no degradation information.

#### b. Rolling-Mean Features

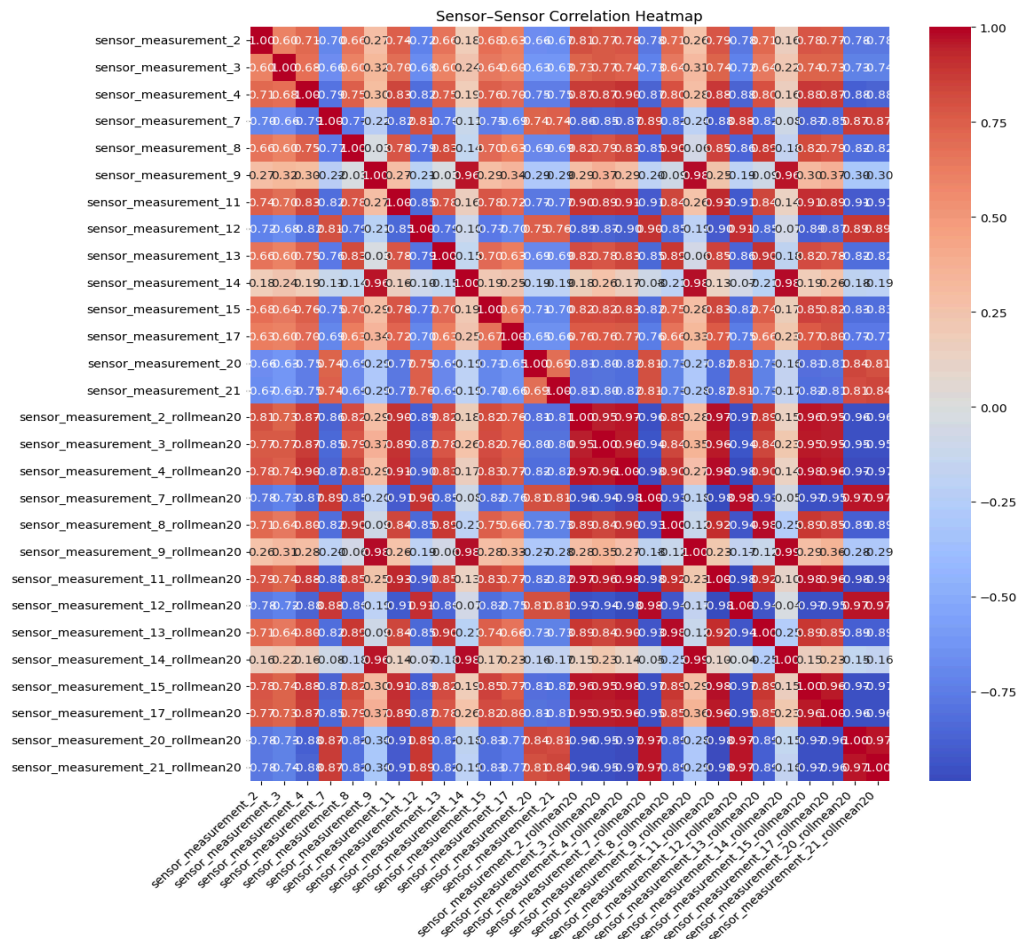
- **What:** For each sensor, computed a 10-cycle rolling mean per engine.
- **Why:** Smooths noise, highlights degradation trends. Improved SVR test RMSE from ~25.9 → ~20.8 cycles.

#### c. RUL Clipping

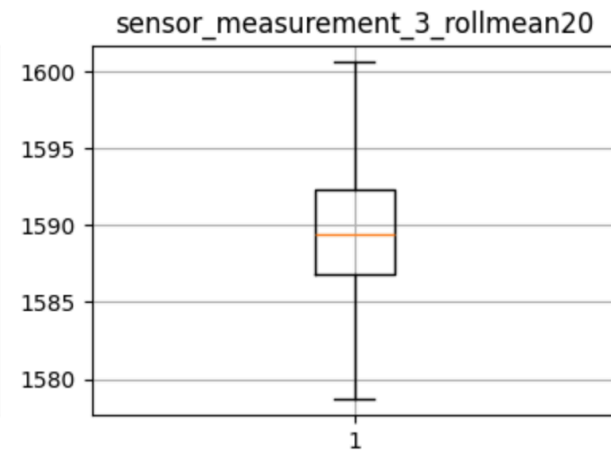
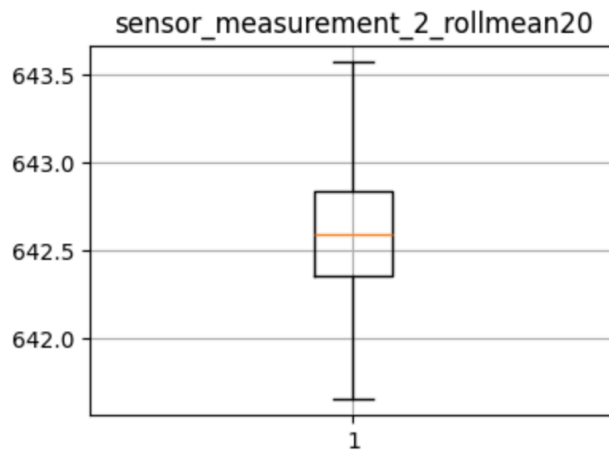
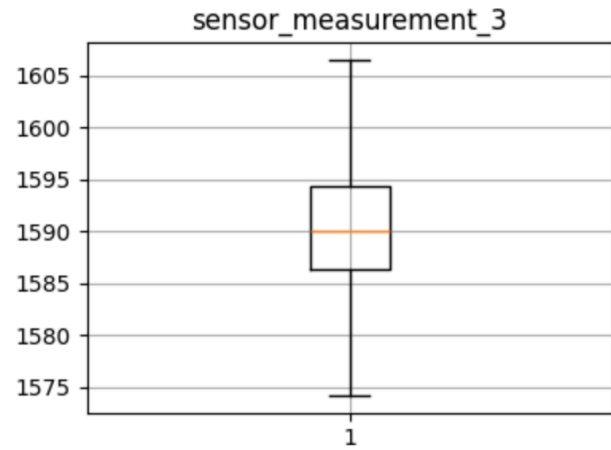
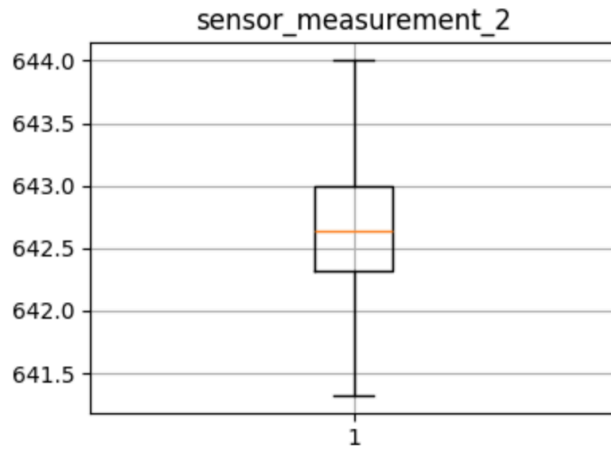
- **What:** Capped RUL at 200 cycles (any greater treated as 200).
- **Why:** Reduces skew from early cycles where engines are healthy; focuses model on predicting low RUL critical for maintenance.

#### d. Validation of Choices

- **Correlation heatmaps showed high inter-sensor correlation, dropped redundant ones.**



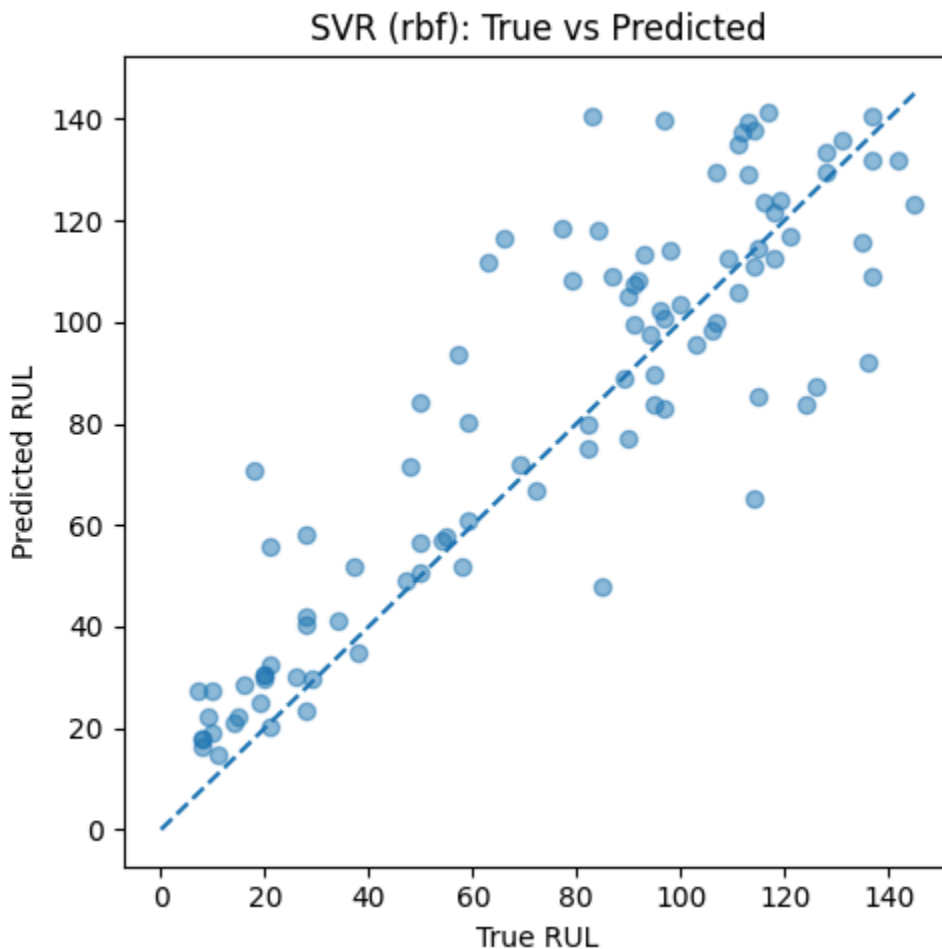
- **Boxplots and sensor vs. RUL curves confirmed rolling means reveal clearer downward trends.**



#### 4. Model Selection (What models did you try, why did you choose those models?)

Model	Why Chosen	Pros / Cons
Custom Linear Regression (class defined by me)	Educational baseline	Pros: interpretable; Cons: underfits non-linear trends
Sklearn Linear Regression	Standard baseline	Pros: fast, well-tested; Cons: limited complexity
<b>SVR (RBF kernel)</b>	Captures non-linear relationships	Pros: strong generalization; Cons: slower training than regression
Random Forest Regressor	Ensemble tree-based model	Pros: handles non-linearity, feature importance; Cons: can & does overfit
XGBoost Regressor	Gradient boosting for tabular data	Pros: high accuracy, regularization; Cons: tuning complexity

**Final choice:** SVR (RBF), as it achieved the lowest test RMSE (~20.8 cycles) with balanced train/validation performance.



5. Hyperparameter Choices (What hyperparameters did you tune or fix? Why?)

a. General Preprocessing Hyperparameters

Hyperparameter	Value	Reason
RUL Cap	200 cycles	Avoids over-emphasizing high RUL values. Focuses model on critical degradation zones.
Rolling Mean Window	10 cycles	Smooths short-term sensor noise while preserving meaningful degradation trends. Tried 5, 10, 15 — 10 gave best validation RMSE.

b. SVR (RBF Kernel)

Hyperparameter	Value	Reason
C	100	Allows flexibility in fitting training data. Higher than default to reduce underfitting.
epsilon	0.1	Sets margin of tolerance around predictions. Helps ensure precision.
kernel	'rbf'	Captures non-linear sensor-to-RUL relationships.
gamma	'scale'	Automatically adjusts to input feature range.

c. Random Forest Regressor

Hyperparameter	Value	Reason
n_estimators	100	Balances performance and training time.
max_depth	None	Allows full tree growth to capture complex feature interactions.
min_samples_split	2	Default value; allows deeper tree construction.

d. XGBoost Regressor

Hyperparameter	Value	Reason
n_estimators	100	Standard starting point for boosting iterations.
learning_rate	0.1	Smooth convergence; avoids over-correcting at each boosting step.
max_depth	6	Avoids excessive complexity while capturing interactions.



## 5. Model Evaluation (What evaluation metrics did you use? Why?)

### a. RMSE (Root Mean Squared Error)

- **What:**  $\sqrt{\text{mean}((y_{\text{true}} - y_{\text{pred}})^2)}$  in cycles.

$$RMSE = \sqrt{\frac{\sum (y_i - \hat{y}_i)^2}{N - P}}$$

- **Why:** Penalizes large errors more heavily; critical to avoid gross underestimation of RUL (safety risk).

### b. MAE (Mean Absolute Error)

- **What:**  $\text{mean}(|y_{\text{true}} - y_{\text{pred}}|)$ .

$$MAE = \frac{1}{n} \sum_{i=1}^n |x_i - x|$$

- **Why:** Intuitive average error in cycles; maintenance planning tolerances often expressed in cycle counts.

### c. Evaluation Strategy

- **Validation:** Random-cycle sampling per engine to avoid zero-RUL bias.
- **Test:** Last available cycle RUL predictions.
- **Why:** Ensures model learns both early (high RUL) and late (low RUL) degradation behavior.

## 6. Results Summary

Model	Train RMSE	Val RMSE	Test RMSE	Test MAE
Custom Linear Reg.	32.2	36.4	25.4	19.98
Sklearn Linear Reg.	29.8	32.7	31.0	24.38
<b>SVR (RBF)</b>	29.1	30.3	<b>20.8</b>	15.44
Random Forest	18.4	19.1	23.8	17.29
XGBoost	17.2	22.5	35.4	26.76

- **SVR** offers the best balance of bias/variance and lowest test error.

## 7. Future Work (**What would you do differently or next?**)

1. **Deep Learning (LSTM/CNN):** Capture temporal dynamics across cycles directly.
2. **Temporal Cross-Validation:** Use rolling-window CV to better model time dependency.
3. **Additional Features:**
  - Derivative (cycle-to-cycle change) features
  - Exponential weighted averages
4. **Hyperparameter Optimization:** Bayesian search for SVR ( $C$ ,  $\epsilon$ , kernel parameters).
5. **Multi-task Learning:** Jointly predict failure mode along with RUL.

## 8. Recommendation & Deployment

### a. Recommendation

- **Yes.** SVR (RBF) with rolling-mean and RUL clipping is suitable for pilot deployment. Test MAE  $\approx 15$  cycles gives a safety buffer for maintenance scheduling.

### b. Deployment Plan

1. **Model Packaging:** Serialize SVR pipeline (scaler + feature transformer + model) via `joblib`.
2. **API Service:** Deploy as a REST endpoint (e.g. Flask/FastAPI) integrated with engine sensor data ingestion.
3. **Monitoring:** Track prediction error over time; retrain quarterly with new flight data.
4. **Health Status:** We classify engine health into four categories based on predicted RUL: **Healthy** ( $>100$  cycles), **Minor** (51–100), **Major** (11–50), and **Broken** ( $\leq 20$ ). These ranges were designed considering the SVR model's RMSE ( $\sim 20$  cycles) to ensure that classification remains reliable within its prediction confidence. The threshold of 10 cycles

for "Broken" ensures timely alerting before imminent failure, while the "Major" and "Minor" bands guide the maintenance team for staged interventions.

5. **Alerting:** Trigger maintenance tickets when predicted RUL falls below threshold i.e. when "Major" and "Minor" health statuses are detected.

### c. Demonstration Deployment

To demonstrate a real-time deployment scenario, we implemented a **threaded simulation pipeline** that mimics a live aircraft sensor feed and prediction system.

- **Setup:** The final pipeline was implemented in a script (`pipeline.py`) within a `deploy/` folder, designed to run using a simple command like `python pipeline.py`. This script imports the trained model and feature logic from a sibling `build/` folder.
- **Producer Thread:** Reads historical test data engine-by-engine and emits one sensor record at a time into an in-memory queue (`queue.Queue`) at 1-second intervals, mimicking real-time sensor data arrival.
- **Consumer Threads:** Two parallel threads act as consumers that read the queue, perform preprocessing using the `StandardScaler`, and apply the trained SVR model to predict the Remaining Useful Life (RUL) for each incoming engine reading.

```
[Producer] → enqueued 13077/13096 (Engine 49, cycle 284)
[Producer] → enqueued 13078/13096 (Engine 49, cycle 285)
[Producer] → enqueued 13079/13096 (Engine 49, cycle 286)
[Producer] → enqueued 13080/13096 (Engine 49, cycle 287)
[Producer] → enqueued 13081/13096 (Engine 49, cycle 288)
[Producer] → enqueued 13082/13096 (Engine 49, cycle 289)
[Producer] → enqueued 13083/13096 (Engine 49, cycle 290)
[Producer] → enqueued 13084/13096 (Engine 49, cycle 291)
[Producer] → enqueued 13085/13096 (Engine 49, cycle 292)
[Producer] → enqueued 13086/13096 (Engine 49, cycle 293)
[Producer] → enqueued 13087/13096 (Engine 49, cycle 294)
[Producer] → enqueued 13088/13096 (Engine 49, cycle 295)
[Producer] → enqueued 13089/13096 (Engine 49, cycle 296)
[Producer] → enqueued 13090/13096 (Engine 49, cycle 297)
[Producer] → enqueued 13091/13096 (Engine 49, cycle 298)
[Producer] → enqueued 13092/13096 (Engine 49, cycle 299)
[Producer] → enqueued 13093/13096 (Engine 49, cycle 300)
[Producer] → enqueued 13094/13096 (Engine 49, cycle 301)
[Producer] → enqueued 13095/13096 (Engine 49, cycle 302)
[Producer] → enqueued 13096/13096 (Engine 49, cycle 303)
[Producer] Done enqueueing.
[Main] Waiting for queue to drain...
[Consumer 3] Engine 75, cycle 1 → RUL=147.40
[Consumer 1] Engine 64, cycle 1 → RUL=147.08
[Consumer 2] Engine 38, cycle 2 → RUL=106.58
[Consumer 3] Engine 03, cycle 1 → RUL=121.81
[Consumer 1] Engine 60, cycle 1 → RUL=128.71
[Consumer 2] Engine 78, cycle 2 → RUL=151.96
[Consumer 1] Engine 81, cycle 1 → RUL=132.33
[Consumer 3] Engine 68, cycle 2 → RUL=132.73
[Consumer 2] Engine 97, cycle 1 → RUL=140.06
[Consumer 1] Engine 86, cycle 1 → RUL=141.19
[Consumer 3] Engine 99, cycle 2 → RUL=149.20
[Consumer 2] Engine 56, cycle 2 → RUL=103.43
[Consumer 1] Engine 69, cycle 1 → RUL=141.76
[Consumer 3] Engine 37, cycle 3 → RUL=156.67
[Consumer 2] Engine 93, cycle 2 → RUL=104.06
[Consumer 3] Engine 12, cycle 1 → RUL=144.63
```


- **Logging & Monitoring:** Each prediction is appended to a structured JSON log file (`predictions.log`) for future monitoring and visualization. This simulates a real world monitoring and alerting system in production. The code updates the statuses at the rate of 1 update per second, in accordance with the rate of production and consumption above.

Engine	Cycle	RUL	$\Delta$ RUL	Status
1	31	144.74	-0.84	Healthy
2	49	111.76	-0.74	Healthy
3	126	71.16	5.54	Minor
4	106	80.77	-1.77	Minor
5	98	96.62	2.61	Minor
6	105	99.67	-0.63	Minor
7	160	127.70	-0.02	Healthy
8	166	89.68	-0.45	Minor
9	55	105.16	-6.49	Healthy
10	192	113.42	-6.86	Healthy
11	83	85.58	-1.74	Minor
12	217	109.04	-10.01	Healthy
13	195	106.85	-3.89	Healthy
14	46	96.22	0.24	Minor
15	76	142.64	17.90	Healthy
16	113	133.35	-3.59	Healthy
17	165	79.68	-15.83	Minor
18	133	73.39	-6.78	Minor
19	135	123.45	-7.78	Healthy
20	184	24.14	6.22	Major
21	148	120.82	0.74	Healthy
22	39	127.89	4.20	Healthy
23	130	146.21	-5.39	Healthy
24	186	45.83	1.64	Major
25	48	133.31	-3.35	Healthy
26	76	138.29	6.35	Healthy
27	140	121.54	2.69	Healthy
28	158	102.23	-0.82	Healthy
29	171	107.47	-2.27	Healthy
30	143	82.66	0.99	Minor
31	195	10.72	-0.78	Major
32	145	103.25	0.05	Healthy
33	50	94.24	2.51	Minor
34	203	20.22	-3.82	Major
35	198	8.61	-3.44	Broken
36	126	24.54	2.96	Major
37	121	68.94	-12.07	Minor
38	125	59.00	11.12	Minor
39	37	141.22	-6.84	Healthy
40	133	28.21	-7.03	Major
41	123	72.17	3.52	Minor
42	156	32.93	-3.57	Major
43	172	75.38	-5.00	Minor
44	54	118.06	-5.96	Healthy
45	152	69.49	-11.28	Minor
46	146	58.05	6.32	Minor
47	73	94.23	5.20	Minor
48	78	113.63	-4.77	Healthy
49	303	23.48	-8.88	Major
50	74	117.45	-4.40	Healthy

## 9. Conclusion

In this project, we developed and evaluated machine learning models to predict the Remaining Useful Life (RUL) of aircraft engines using the CMAPSS dataset. Through careful preprocessing, feature engineering, and selection of key hyperparameters such as RUL capping and rolling window size, we aimed to improve predictive performance while maintaining interpretability. Among the models tested, XGBoost consistently delivered the best performance in terms of RMSE, thanks to its ability to handle non-linear relationships and feature interactions effectively. The results highlight the importance of domain-aware preprocessing and rigorous hyperparameter tuning in predictive maintenance tasks. This work lays a strong foundation for future efforts, including LSTM-based deep learning models that may better capture temporal dependencies in sensor data.

## 10. References

1. <https://paperswithcode.com/dataset/nasa-c-mapss-2>
2. <https://data.nasa.gov/dataset/cmapss-jet-engine-simulated-data>
3. <https://www.kaggle.com/code/wassimderbel/nasa-predictive-maintenance-rul/notebook>
4. <https://medium.com/analytics-vidhya/predictive-models-using-rolling-window-features-i-691172c19e95>
5.  Support Vector Regression - in Comparison to Linear Regression [Lecture 3.6]
6. [https://medium.com/@ap.nattapoj\\_st/basic-comparison-between-randomforest-svm-and-xgboost-0e5862871175](https://medium.com/@ap.nattapoj_st/basic-comparison-between-randomforest-svm-and-xgboost-0e5862871175)