Prescriptive Algorithm Report

Vishal Sehgal (2109374)

May 09, 2025

1 Introduction

This report addresses a predictive maintenance scheduling problem for airplane engines, focusing on estimating the Remaining Useful Lifetime (RUL) of engines to optimize maintenance schedules and avoid operational failures. The RUL represents the number of operational cycles an engine can perform before it becomes unsafe for use. Using provided datasets, we developed a predictive model to estimate RUL for engines currently in use, compared our predictions with consultancy and analyzed the differences. The datasets include:

- **DataTrain.csv**: Run-to-failure data for 100 engines, containing engine IDs, cycle counts, three operational settings, and 21 sensor measurements.
- **DataSchedule.csv**: Temporal data for 100 engines currently in use, with the same variables as Dataset 1 but without known RUL values.
- **RUL consultancy predictions A3.csv**: Consultancy-provided RUL predictions for the engines in Dataset 2.

The goal was to construct a Random Forest model using DataTrain.csv, predict RUL for Dataschedule.csv, and evaluate our predictions against the consultancy predictions.

2 Modeling

2.1 Data Analysis and Feature Construction

To develop an effective predictive model, we analyzed DataTrain.csv to understand the data structure and identify features relevant to RUL prediction. The dataset includes 26 columns: engine ID, cycle count, three operational settings (set1, set2, set3), and 21 sensor measurements (sensor_val1 to sensor_val21). 1 .

Table 1: DataTrain.csv

engine_id	cycle	set1	set2	set3	sensor_val1	sensor_val2	···	sensor_val21
1	1	100.0	-0.0007	-0.0004	9046.19	521.66	···	23.4190
1	2	100.0	0.0019	-0.0003	9044.07	522.28	···	23.4236
1	3	100.0	-0.0043	0.0003	9052.94	522.42	···	23.3442
1	4	100.0	0.0007	0.0000	9049.48	522.86	···	23.3739
1	5	100.0	-0.0019	-0.0002	9055.15	522.19	···	23.4044

The RUL for each engine was calculated by determining the maximum cycle count for each engine ID and subtracting the current cycle number. For example, an engine with a maximum cycle of 192 has an RUL of 1 at cycle 191 and 2 at cycle 190. This process created a new 'RUL' column for modeling.

To select relevant features, we conducted a var analysis to identify columns with significant variation across engine cycles, as constant or low-var features are unlikely to correlate with RUL. We grouped the data by engine ID and computed the minimum, maximum, and mean for each numerical column. Columns with identical minimum, maximum, and mean across all engines were classified as low-var and excluded. The analysis identified the following low-var columns:

low_var_cols: set1, sensor_val7, sensor_val8, sensor_val11, sensor_val14, sensor_val16, sensor_val20

The remaining columns, excluding engine ID and cycle, were selected as key features for modeling due to their var, suggesting potential correlation with RUL. These features include:

key_features: cycle, set2, set3, sensor_val1, sensor_val2, sensor_val3, sensor_val4, sensor_val5, sensor_val6, sensor_val9, sensor_val10, sensor_val12, sensor_val13, sensor_val15, sensor_val17, sensor_val18, sensor_val19, sensor_val21, Max_cycle, RUL

To assess feature suitability, we visualized the distributions of numerical columns using graphs, confirming the var of selected features and revealing their skewness and distribution patterns. 1 2

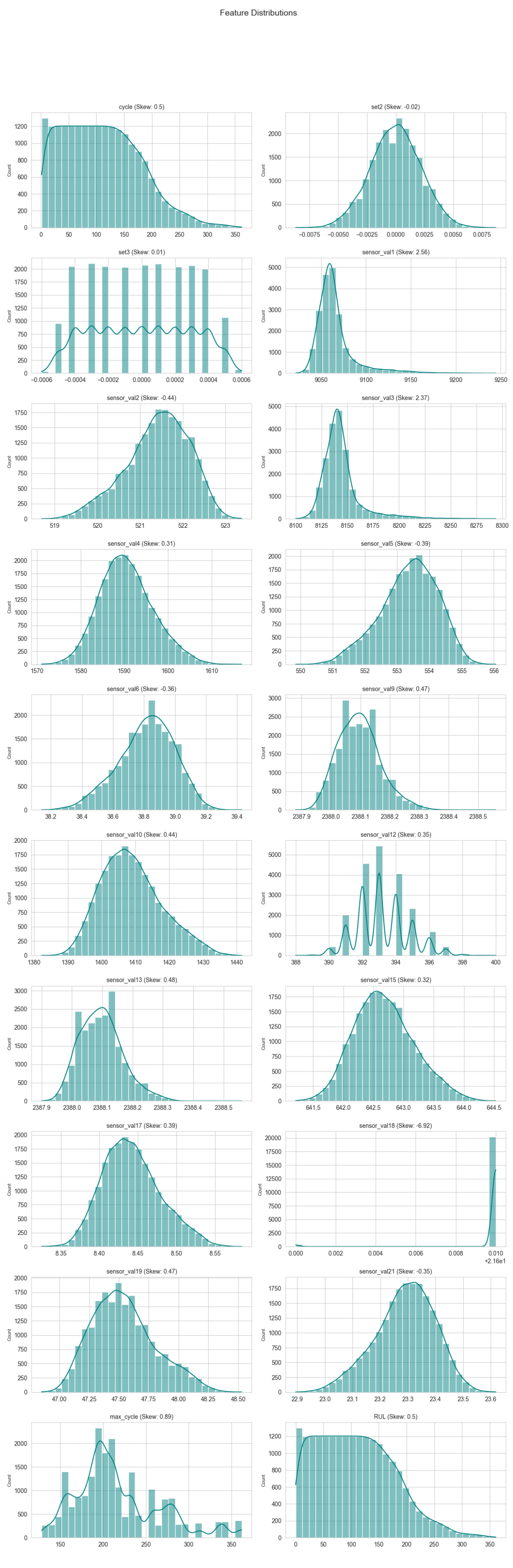


Figure 1

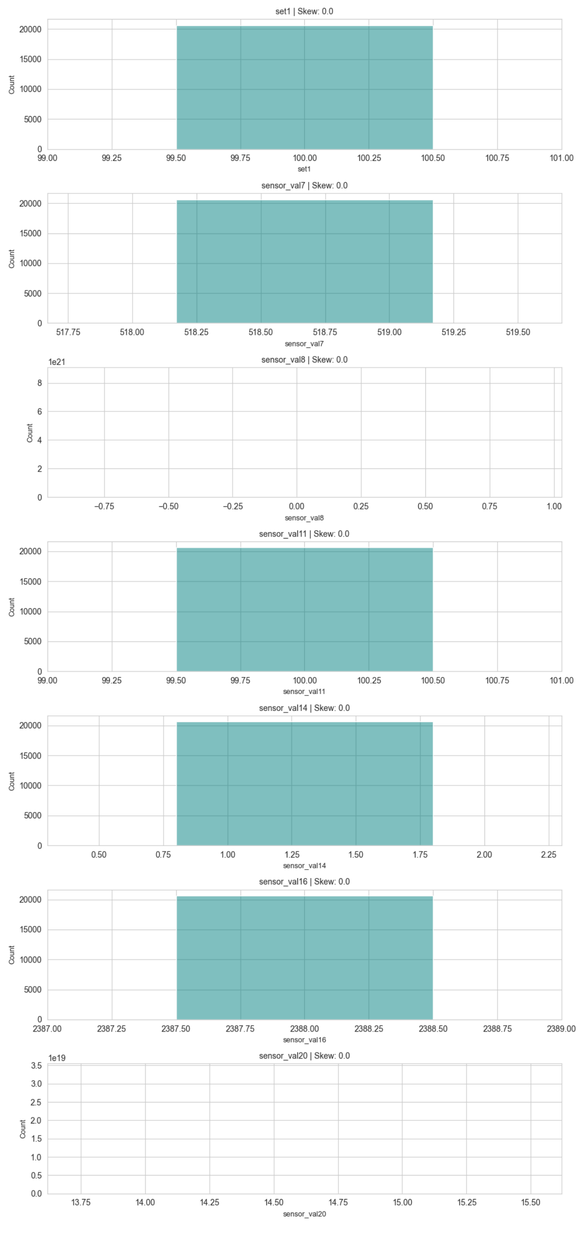


Figure 2

2.2 Random Forest Model Development

I developed a Random Forest regression model to predict RUL. The model was trained on Dat-Train.csv, using the selected features as inputs and RUL as the target variable.
Hyperparameters were tuned using a grid search within a 5-fold cross-validation framework, testing:

- Number of trees: 50, 100, 200
- Maximum depth: None, 10, 20, 30
- Minimum samples to split: 2, 5, 10
- Minimum samples at leaf: 1, 2, 4
- Max Feature: [sqrt],[log2]

The best combination was selected based on the lowest Mean Absolute Error, suitable for RUL prediction due to its linear penalty on errors. The dataset was split into 80% training and 20% testing sets, with 5-fold cross-validation on the training set to ensure robust tuning. MAE was chosen as the performance metric because it directly measures average absolute deviation, critical for maintenance scheduling, and is less sensitive to outliers than Mean Squared Error .
The model showed a reasonable accuracy in a test .

3 Prediction

3.1 RUL Prediction for DataSchedule.csv

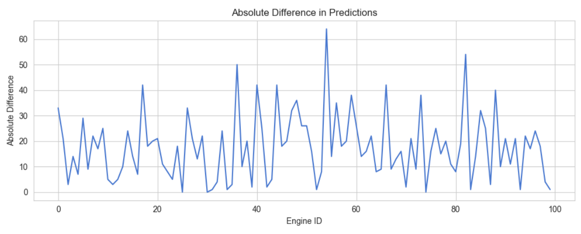
The trained Random Forest model was applied to DataSchedule.csv to predict RUL for the 100 engines currently in use, using the same features as in training. Predictions were rounded to integer values for maintenance scheduling.

3.2 Comparison with Consultancy Predictions

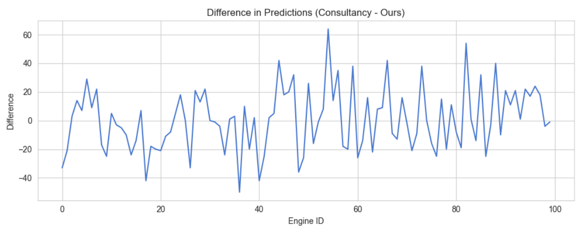
We compared our RUL predictions with those in `RUL consultancy predictions A3.csv` using the following metrics:
The results were:

- MAE: 17.61
- MSE: 481.31
- Mean Absolute Difference: 17.61
- Mean Difference: 0.63
- Standard Deviation of Absolute Differences: 13.08

We visualized the differences using two plots:



(a)



(b)

Figure 3

4 OPTIMIZATION TASK

4.1 Genetic Algorithm

4.1.1 Approach Explanation

The Genetic Algorithm is developed to allocate four maintenance teams (two of type A: T1, T3; two of type B: T2, T4) to 100 engines over a 30-day planning horizon, starting at day t=1. The objective is to minimize penalty costs based on safety due dates derived from RUL predictions. Each engine must be maintained exactly once and teams can only work on one engine at a time without overlap on the same engine.

Individual Representation: An individual represents a complete maintenance schedule. It consists of four teams, each assigned a list of tasks. Each task includes the engine ID, RUL, start day, end day (computed using team-specific durations uAj for type A, uBj for type B), days late (if maintenance ends after the safety due date), and penalty cost. The individual also tracks the total penalty cost across all tasks and the maximum duration of the schedule (up to 30 days).

Selection Strategy: Tournament selection is used to select parents. A random subset of 5 individuals (tournament size) is chosen from the population, and the two with the lowest total penalty costs are selected. This method ensures that better schedules have a higher chance of reproduction while maintaining some diversity.

Crossover Method: Crossover involves selecting one team from each parent and swapping task segments at a random point. For example, if Team T1's tasks from Parent 1 are swapped with Team T2's tasks from Parent 2, the child inherits these swapped segments, while the remaining teams (T2, T3, T4 from Parent 1; T1, T3, T4 from Parent 2) are copied directly. This preserves partial schedules while introducing new combinations.

Mutation Method: No explicit mutation is implemented. The diversity introduced by crossover, combined with a repair strategy for infeasible solutions, is sufficient to explore the solution space. Adding mutation (e.g., swapping tasks between teams) was considered unnecessary given the problem size and constraints, as the repair mechanism already adjusts schedules effectively.

Handling Infeasible Solutions: After crossover, duplicates (an engine assigned to multiple teams) or missing engines may occur. A repair process identifies unassigned engines and assigns them to teams with the least workload, removing duplicates and ensuring all 100 engines are scheduled exactly once. Placeholder tasks (if any) are replaced, and workloads are balanced within a maximum imbalance of 5 days. **Design Choices and Hyperparameters**:

1. Population Size: 100 individuals
2. Crossover Probability: 0.9
3. Tournament Size: 5
4. Elitism Size: 10 (preserves the top 10 solutions each generation)
5. Number of Iterations: 100
6. Random Insertion Maximum Imbalance: 5 days (for workload balancing)
7. Maximum Daily Cost: 250 (per problem specification)
8. Planning Horizon: 30 days

These parameters ensure a balance between exploration (via crossover and population size) and exploitation (via elitism and tournament selection), while keeping runtime manageable.

4.2 Optimization

The GA was executed for 100 iterations using the predicted RULs as input, with a total runtime under 5 minutes. The best solution was identified and saved.

4.2.1 Performance Over 30 Runs

The average best fitness (total penalty cost) across these runs was computed and plotted. The results also note how often a specific low cost (e.g., 30) was achieved, indicating consistency. The GA reliably produces feasible schedules within the 30-day horizon. Elitism preserves high-quality solutions, while crossover and repair ensure diversity and feasibility. The average best fitness likely improves over iterations, stabilizing as the algorithm converges. Performance depends on RUL accuracy, as tighter safety due dates increase penalty risks.

4.3 Comparision

The GA was run 30 times using the consultancy's RUL predictions each within 5 minutes. Differences between the predicted and consultancy RULs may lead to varying penalty costs. If consultancy RULs are shorter, safety due dates occur earlier, potentially increasing penalties unless schedules adapt. Conversely, longer RULs could reduce costs. The plot comparison reveals which dataset yields better outcomes, reflecting the impact of prediction accuracy on scheduling efficiency.