# C964: Computer Science Capstone – Predictive Maintenance with NASA CMAPSS

**Student:** Kelley Buhlig  
**Program:** B.S. Computer Science  
**Date:** August 24, 2025

## Part A: Letter of Transmittal

**Subject:** Proposal for a Predictive Maintenance Application Using NASA CMAPSS Data

**Date:** August 24, 2025  
**To:** Capstone Evaluation Committee, Western Governors University  
**From:** Kelley Buhlig, Student – B.S. Computer Science

Dear Committee Members,

Unplanned equipment failures impose significant operational risk and cost in aerospace and other asset‑intensive industries. This proposal requests approval to develop and evaluate a predictive maintenance application that estimates the Remaining Useful Life (RUL) of turbofan engines using the publicly available NASA CMAPSS dataset. The resulting tool will demonstrate practical business value by enabling earlier maintenance interventions, reducing downtime, and improving safety margins.

**Problem Summary.** Maintenance organizations often rely on fixed schedules or reactive repairs. These approaches cannot fully exploit rich sensor telemetry collected during operations. The organization needs an approach that converts multivariate time‑series data into actionable RUL predictions that can be monitored and explained to decision makers.

Proposed Solution. I will build a browser‑based data science application (implemented as a VS Code notebook and accompanying HTML export) that reads CMAPSS engine telemetry, trains a supervised machine learning model to predict RUL, and presents three decision‑support visualizations: (1) Predicted vs. Actual RUL with error highlighting, (2) Feature Importance ranking, and (3) a Correlation heatmap focusing on the most informative sensors. A concise user interface inside the notebook will let users select a scenario (FD001–FD004), load data, retrain, and obtain predictions for new inputs.

**Benefits to the Organization.** The application translates raw telemetry into time‑to‑failure estimates that support planning and inventory decisions. Benefits include fewer unplanned removals, better part lifecycle planning, and clearer communication between engineering and operations through interpretable charts.

Implementation Plan (Summary). The project will follow a lightweight, industry‑standard lifecycle: requirements and data understanding; modeling and validation; packaging and documentation; and post‑implementation analysis. Development will occur in VS Code to ensure easy review on a standard Windows 10 machine without installs. Source code, data processing steps, and trained models will be included with a user guide.

Costs, Timeline, and Ethics. The solution uses free, open tools (VS Code, Python/NumPy/Pandas/scikit‑learn). The dataset (NASA CMAPSS) is public and appropriate for academic use. Anticipated effort is ~40–60 hours across 4–6 weeks (details in Part B). No personally identifiable information or sensitive customer data is used. The report will document reproducibility and limitations.

**Qualifications.** My coursework includes Data Structures, Algorithms, Operating Systems, and Machine Learning fundamentals; I have hands‑on practice with Python, Pandas, and scikit‑learn and have implemented time‑series models in prior coursework.

Sincerely,

*Signed,*  
Kelley Buhlig  
Student, B.S. Computer Science  
Western Governors University

## Part B: Project Proposal Plan

### Project Summary

**Problem.** The client needs a data‑driven way to predict turbofan engine RUL from multivariate sensor streams to reduce unplanned downtime.

**Client and Needs.** The target stakeholder will be a maintenance planning team that will need: (1) accurate RUL estimates, (2) visibility into which sensors influence predictions, and (3) an auditable workflow for training and evaluating models.

Deliverables. The project will deliver:  
- A functioning VS Code application (notebook + HTML export) that loads CMAPSS data, trains a model, and produces predictions.  
- Three visualizations (Predicted vs. Actual, Feature Importances, Correlation/Top‑10 correlations).  
- Trained model artifacts and feature metadata.  
- A user guide with step‑by‑step instructions.  
- A technical report describing data handling, model selection, validation, and results.

**Benefit Justification.** The application will enable proactive maintenance decisions by forecasting RUL with defensible validation and transparent visualizations. This will illustrate cost avoidance potential (fewer surprises, improved parts staging) and provide an extensible template for other assets.

### Data Summary

Source and Collection. The project will use the NASA CMAPSS turbofan engine simulation dataset (FD001–FD004 scenarios). Data will be provided as text files with operating settings and 26 sensor measurements per cycle and will be uploaded to VS Code for processing.

**Processing and Management.** Data will be parsed using whitespace‑delimited reads, labeled with per‑unit RUL (maximum cycle minus current cycle), and stored as Pandas DataFrames. Processing will include group‑aware splits by engine unit to prevent leakage, optional feature engineering (rolling means and first‑order differences), and serialization of models and feature lists for reproducibility.

**Fitness for Purpose and Data Quality.** CMAPSS provides rich multivariate signals with ground‑truth RUL suitable for supervised regression. Outliers and anomalies will be addressed through visualization and optional winsorization or robust models. Missing values are not expected in raw files; any derived features that introduce NaNs (e.g., differences) will be imputed conservatively (e.g., zero for first differences).

**Ethical/Legal Considerations.** The dataset is public. No proprietary customer data will be used. Methods and code will be documented for auditability.

### Implementation

Methodology. The project will follow CRISP‑DM adapted for time‑series: business understanding → data understanding → data preparation → modeling → evaluation → deployment (notebook packaging and user guide). Modeling will begin with a Random Forest regressor (interpretable importance, low preprocessing) and may add Gradient Boosting. Deep learning is out of scope for VS Code but will be discussed as future work.

**Implementation Outline.**  
1. **Setup & Data Understanding:** Load FD001; verify schema, compute labels, create basic plots.  
2. **Preparation:** Group‑aware split by unit; baseline features + optional deltas/rolling windows.  
3. **Modeling:** Train Random Forest; tune hyperparameters with group‑aware CV; record metrics.  
4. **Evaluation:** Compute RMSE; generate three visualizations.  
5. **Packaging:** Save model + features; export notebook to HTML; finalize user guide.  
6. **Report:** Complete narrative report.

### Timeline

| Milestone or Deliverable | Project Dependencies | Resources | Start and End Date | Duration |
| --- | --- | --- | --- | --- |
| Requirements & Data Understanding | None | VS Code, CMAPSS FD001 | Jul 27 – Jul 31, 2025 | 5 days |
| Feature Engineering & Splits | Data loaded & labeled | Pandas, NumPy | Aug 1 – Aug 5, 2025 | 5 days |
| Baseline Modeling (RF) | Features ready | scikit‑learn | Aug 6 – Aug 10, 2025 | 5 days |
| Hyperparameter Tuning | Baseline complete | scikit‑learn | Aug 11 – Aug 14, 2025 | 4 days |
| Evaluation & Visualizations | Model trained | Matplotlib | Aug 15 – Aug 17, 2025 | 3 days |
| Packaging & User Guide | Evaluation done | JupyterLite export | Aug 18 – Aug 20, 2025 | 3 days |
| Final Report | All above | Word | Aug 21 – Aug 24, 2025 | 4 days |

*NA where not applicable; dates/times can be adjusted to course deadlines.*

### Evaluation Plan

**Verification During Development.**  
- Unit‑level checks: schema, row counts, per‑unit last cycle labeling.  
- Sanity plots: correlations, distributions, and predicted vs. actual on validation data.  
- Group‑aware data splits verified (no shared unit\_number across train/val).

**Validation Upon Completion.**  
- Metric: Root Mean Squared Error (RMSE) on validation and official test split (last cycle per test unit).  
- Secondary Metric: NASA asymmetric scoring function (penalizes late predictions more than early).

### Resources and Costs

* Software: VS Code (free), Python (NumPy/Pandas/scikit‑learn/matplotlib – free).
* **Hardware:** Standard Windows 10 laptop/desktop.
* **Labor:** ~50 hours @ $0 (student).
* **Environment:** Local browser execution; no hosting costs.
* Contingencies: If interactivity is required beyond VS Code, Google Colab (free) will be used for optional exploration.

## Part C: Application

This section documents the submitted application components.  
- Primary Artifact: capstone.ipynb (VS Code‑compatible) and exported cmapss\_capstone.html.  
- Data Files: train\_FD001.txt, test\_FD001.txt, RUL\_FD001.txt (plus optional FD002–FD004).  
- Model Artifacts: rf\_fd001.pkl, feature\_cols.json.  
- Visualizations (included in the notebook and screenshots in the report):  
1. Predicted vs Actual RUL scatter (colored by absolute error).  
2. Top‑10 Feature Importances (horizontal bar chart).  
3. Correlation heatmap (features + RUL) and Top‑10 correlations with RUL.  
- User Interface: Parameter cells to select scenario, load data, train model, and generate predictions.

## Part D: Post‑implementation Report

### Solution Summary

This project addressed the need for proactive maintenance by predicting Remaining Useful Life (RUL) from engine telemetry. I implemented a VS Code application that ingested NASA CMAPSS data, trained a supervised regression model, and produced decision‑support visualizations. The solution enabled users to estimate time‑to‑failure and understand which sensors influenced model predictions.

### Data Summary

I used the NASA CMAPSS dataset (FD001 as primary). Each record contained an engine unit identifier, time in cycles, three operating settings, and 26 sensor readings. I computed RUL labels by subtracting the current cycle from each unit’s maximum cycle. Data was managed in Pandas, with group‑aware splits to prevent leakage. Optional engineered features (first‑order differences) were imputed with zero for initial positions.

### Machine Learning

**Method (What).** I trained a Random Forest Regressor to map operating settings and sensor values to RUL.

**Development (How).** I performed group‑aware train/validation splits by engine unit, fit the model on baseline features, and evaluated with RMSE. I then produced feature importances to support explainability and generated plots for error analysis.

**Justification (Why**). Random Forests offered strong baseline accuracy with minimal preprocessing and are compatible with VS Code’s environment. They also provide feature importances for transparency, which supports stakeholder trust.

### Validation

**Category.** Supervised learning (regression).  
**Method.** I validated with GroupKFold cross‑validation by unit and reported RMSE on both validation folds and the official test split (predicting the last cycle per test unit and comparing to the provided RUL vector). I also computed the NASA asymmetric score to reflect operational risk preferences.

**Results.**  
- Validation RMSE: **35.50** cycles.  
- Test RMSE (FD001 last‑cycle): **34.32** cycles.  
- NASA Score: **57865.96286503829**.  
*Note: Values are reproducible by executing the included notebook; exact numbers may vary slightly due to randomness unless the random seed is fixed.*

### Visualizations

* **Predicted vs Actual RUL** highlighted where predictions deviated from the identity line and helped diagnose systematic over/under‑prediction.

A graph of a graph

AI-generated content may be incorrect.

* **Top‑10 Feature Importances** identified the most influential sensors and operating settings.

A graph with blue squares

AI-generated content may be incorrect.

* **Correlation Heatmap / Top‑10 Correlations** revealed redundant sensors and those most associated with RUL.

A screenshot of a computer

AI-generated content may be incorrect.

### User Guide (Step‑by‑Step)

1. Open VS Code on Windows 10.
2. **Upload Files**: train\_FD001.txt, test\_FD001.txt, RUL\_FD001.txt (and any additional scenarios).
3. **Open capstone.ipynb** and run the first setup cell (imports).
4. **Load & Label Data** by running the data‑loading cell (whitespace separator and column names) and the RUL labeling cell.
5. **Split by Unit** using the group‑aware split cell (prevents leakage).
6. **Train the Model** by running the Random Forest training cell.
7. **Evaluate** by executing the metrics cell to print RMSE; then run the three plotting cells to generate required visualizations.
8. **Official Test**: Run the test‑evaluation cell to compute last‑cycle predictions versus RUL\_FD001.txt.
9. **Save Artifacts**: Run the save cell to write rf\_fd001.pkl and feature\_cols.json.
10. **Export HTML**: Use File → Save As → Export to HTML



Dependencies/Installs. No local installs are required in VS Code. If the reviewer prefers to run locally or in Colab, requirements.txt will list pandas, numpy, scikit‑learn, and matplotlib.

## Reference Page

* NASA, CMAPSS Turbofan Engine Degradation Simulation Data. Public dataset used for academic research and model development.
* Pedregosa et al., *Scikit‑learn: Machine Learning in Python*, JMLR (for algorithm implementations).
* Hunter, Matplotlib and Seaborn: A 2D Graphics Environment (for plotting).