Predictive Maintenance Using Engine Sensor Data

Final Project Report

Siddhartha Mukherjee

February 14, 2026

Executive Summary

This report presents the end-to-end development of a predictive maintenance solution using engine sensor data. The objective is to anticipate engine failure events by leveraging machine learning models trained on historical sensor measurements, and to deploy the model so that stakeholders can use it for operational decisions.

A master project folder with a dedicated `data/` subfolder was created, and the dataset was registered on the Hugging Face Dataset Hub for reproducibility. Exploratory data analysis revealed class imbalance and meaningful sensor patterns prior to failure, which informed data preparation and model choice. The dataset was loaded from Hugging Face, cleaned, split into training and testing sets, saved locally, and the processed splits were uploaded back to the Hub. Model building compared five classifiers: Decision Tree (baseline), Random Forest, Logistic Regression, Gradient Boosting, and XGBoost, each with appropriate tuning where applicable. All models were evaluated on the same test set; the best model by recall (Gradient Boosting in the reported run) was selected and registered on the Hugging Face Model Hub.

The trained model was deployed as a Streamlit application on a Hugging Face Space, with a Dockerfile and dependencies file, and a hosting script to push deployment files to the Space. An automated GitHub Actions workflow runs data preparation, model training, and deployment on every push to the main branch. This report includes output evaluation (repository and workflow screenshots, live app link and screenshot) and actionable insights and recommendations.

1 Introduction

Unplanned engine failures lead to operational downtime, increased maintenance costs, and reduced asset availability. Traditional reactive or schedule-based maintenance strategies are either costly or ineffective at preventing unexpected breakdowns.

Predictive maintenance leverages historical sensor data and machine learning models to anticipate failures before they occur. By identifying early warning patterns in engine operating conditions, maintenance activities can be scheduled proactively, reducing both downtime and cost.

This final report covers the complete lifecycle: data registration, exploratory data analysis, data preparation, model building with experimentation tracking, model deployment, automated GitHub Actions workflow, output evaluation, and actionable insights and recommendations.

2 Data Registration

A **master folder** (project or repository root) was created, with a **data** subfolder to store all datasets. This structure ensures a clear separation between raw data, processed datasets, and experimental artifacts, and supports organized data management and traceability.

To ensure reproducibility, auditability, and version control, the raw dataset was registered on the Hugging Face Dataset Hub. The dataset was uploaded in its original form prior to any preprocessing or transformation, ensuring that all downstream analysis can be traced back to a consistent and immutable data source.

All subsequent data loading operations in the analysis and modeling pipeline retrieve the dataset directly from the Hugging Face repository rather than from local file paths. This approach guarantees consistency across experiments and enables independent verification of results.

Dataset Details

- **Dataset Name:** predictive-maintenance-engine-data
- **Storage Platform:** Hugging Face Dataset Hub
- **Dataset URL:** `https://huggingface.co/datasets/mukherjee78/predictive-maintenance-engine-data`
- **Data Type:** Multivariate engine sensor data
- **Intended Use:** Predictive maintenance and engine failure prediction

3 Exploratory Data Analysis

This section presents an exploratory analysis of the engine sensor dataset to understand its structure, characteristics, and underlying patterns. The objective of this analysis is to gain insights into data quality, feature behavior, and relationships between variables, which will inform subsequent data preparation and modeling decisions.

3.1 Data Collection and Background

The dataset consists of multivariate sensor readings collected from industrial engines operating under varying conditions. Each engine is monitored over multiple operational cycles, with sensor measurements capturing different aspects of engine health and performance. The primary objective of the dataset is to enable predictive maintenance by identifying patterns that precede engine failure events.

Such sensor-driven monitoring systems are commonly used in industrial settings to reduce unplanned downtime and optimize maintenance schedules. By analyzing historical sensor behavior, early warning indicators of potential failures can be identified, allowing proactive intervention.

3.2 Data Overview

An initial structural assessment of the dataset was conducted to understand its scale, composition, and data quality. The dataset consists of multiple numerical variables representing engine operational parameters and sensor measurements recorded over time.

The dataset contains a single target variable indicating engine failure status, which is modeled as a binary classification problem. All remaining variables represent continuous sensor readings related to temperature, pressure, rotational speed, and fuel characteristics.

An inspection of the dataset revealed that the majority of features are numerical in nature, making the dataset well-suited for machine learning algorithms that operate on continuous inputs. A check for missing values confirmed that the dataset does not contain significant missing data, reducing the need for extensive imputation strategies.

Descriptive statistics were computed for all numerical features to examine their central tendency and variability. Several sensor variables exhibit wide value ranges, indicating differences in operational regimes and engine conditions. These observations motivate the use of robust models capable of handling heterogeneous feature scales.

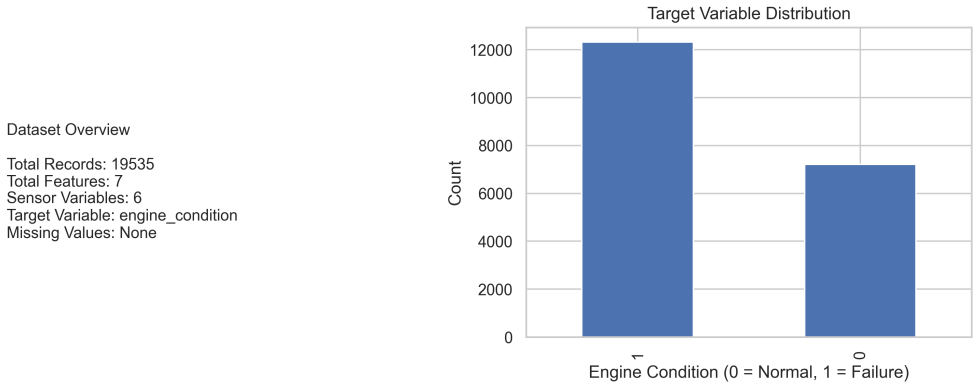

Figure 1: Overview of dataset structure and descriptive statistics

3.3 Univariate Analysis

Univariate analysis was performed to examine the distribution and variability of individual features. This analysis helps identify skewed distributions, potential outliers, and irregular sensor behavior.

The distribution of the target variable was analyzed to assess class balance. Additionally, representative sensor variables were visualized to understand their value ranges and overall behavior across engine cycles.

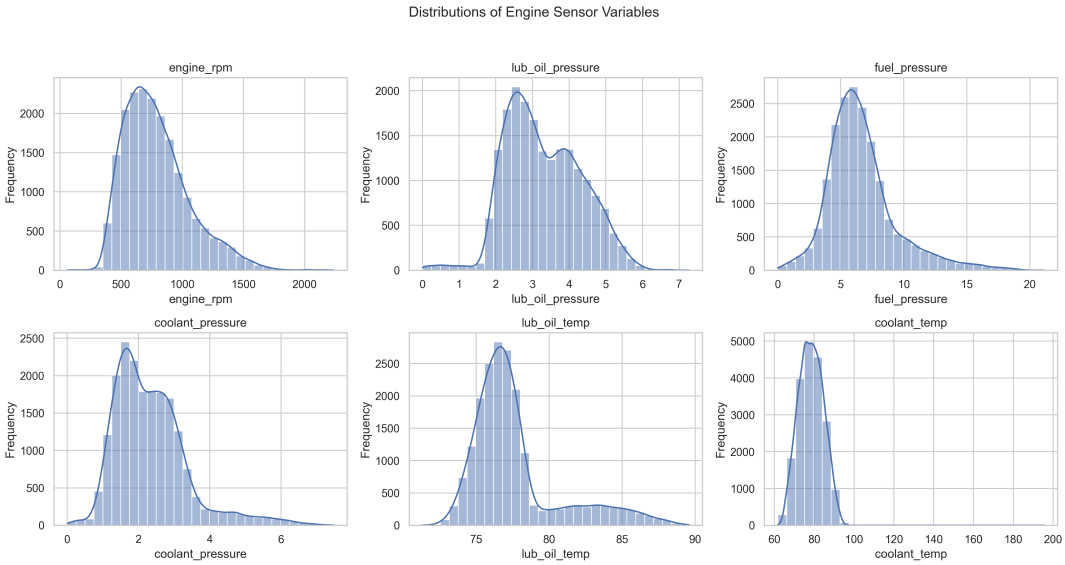

Figure 2: Univariate distributions of key engine sensor variables

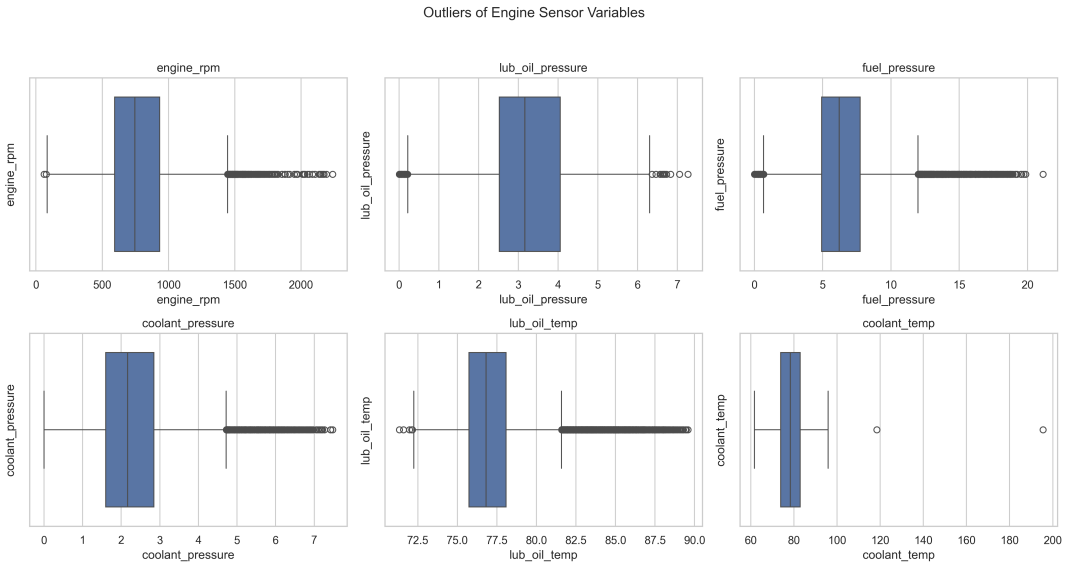

Figure 3: Outlier patterns observed across engine sensor variables

3.4 Bivariate Analysis

Bivariate analysis was performed to examine the relationship between individual sensor variables and the engine failure target. Sensor readings were compared across failure and non-failure instances to identify features that exhibit distinguishable behavior prior to failure events.

Boxplots were used to visualize the distribution of selected sensor variables across the two target classes. Several sensors show noticeable shifts in median values and increased variability in failure cases, suggesting that these variables may contain predictive signals related to engine degradation.

In addition to visual analysis, summary statistics grouped by failure status were examined to quantify differences in sensor behavior between the two classes. These comparisons highlight sensor variables that demonstrate systematic changes as engines approach failure.

The results of this analysis provide early intuition regarding feature relevance and support the selection of models that can capture non-linear relationships between sensor readings and failure outcomes.

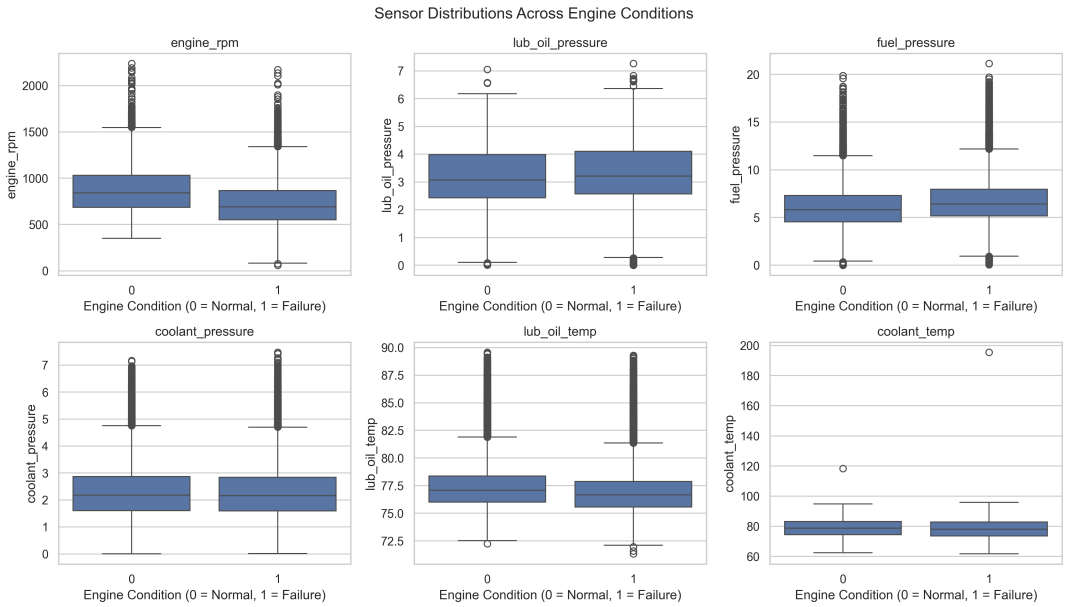

Figure 4: Sensor value comparison across normal and failure engine conditions

3.5 Multivariate Analysis

Multivariate analysis was conducted to explore relationships among multiple sensor variables simultaneously and to identify potential interdependencies. A correlation analysis was performed across sensor features to assess the degree of linear association between variables.

The correlation matrix reveals the presence of several strongly correlated sensor pairs, indicating redundancy in the information captured by certain measurements. Such multicollinearity can impact model interpretability and may influence feature selection decisions.

Understanding these inter-feature relationships is important for guiding downstream modeling choices. In particular, the presence of correlated inputs supports the use of tree-based ensemble models, which are generally robust to multicollinearity and can effectively leverage redundant signals without requiring explicit feature elimination.

This analysis also provides a foundation for potential dimensionality reduction or feature pruning in later stages of the project, if required.

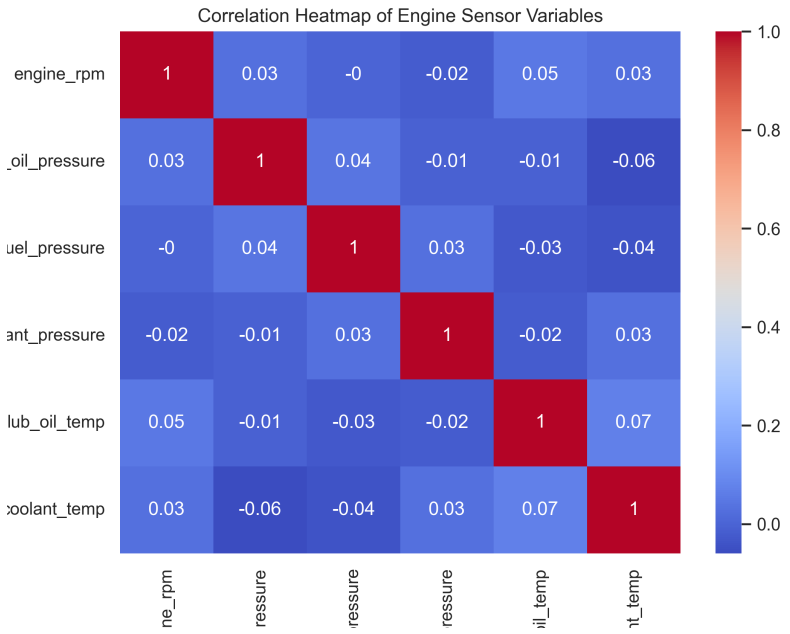

Figure 5: Correlation heatmap of engine sensor variables

3.6 EDA Insights and Observations

- Engine failure events are relatively rare, indicating class imbalance and the need for recall-focused evaluation metrics.
- Pressure- and temperature-related sensors show systematic shifts prior to failure, highlighting their predictive relevance.
- Several sensor variables are strongly correlated, suggesting redundancy but supporting the use of tree-based ensemble models.
- The observed non-linear relationships motivate the use of ensemble learning approaches over linear models.

These insights directly inform the data preparation strategy and guide the selection of appropriate modeling techniques in the subsequent stages of the project.

4 Data Preparation

This section describes the steps undertaken to prepare the dataset for machine learning modeling. The workflow satisfies the required steps: (1) load the dataset directly from the Hugging Face data space; (2) perform data cleaning and remove unnecessary columns; (3) split the cleaned dataset into training and testing sets and save them locally; (4) upload the resulting train and test datasets back to the Hugging Face data space.

4.1 Data Loading

The dataset was loaded directly from the Hugging Face Dataset Hub to ensure consistency with the registered data source. Loading the data from a centralized repository guarantees reproducibility and ensures that all experiments are conducted on a version-controlled dataset rather than local copies.

4.2 Data Cleaning and Feature Selection

An initial data cleaning step was performed to remove columns that are not relevant for predictive modeling. Since the objective is to predict engine failure based on sensor behavior, only numerical sensor variables and the target variable were retained.

The dataset did not contain missing values across the selected features, eliminating the need for imputation. No feature scaling or normalization was applied at this stage, as the subsequent modeling phase focuses on tree-based algorithms that are inherently robust to differences in feature scale and distribution.

4.3 Train–Test Split

The cleaned dataset was split into training and testing subsets to enable unbiased model evaluation. A stratified split strategy was employed to preserve the original class distribution of the target variable in both subsets, which is particularly important given the class imbalance observed during exploratory data analysis.

The training and testing datasets were saved locally to maintain a clear separation between raw, processed, and modeling-ready data.

4.4 Dataset Versioning and Upload

To maintain reproducibility and support experiment tracking, the processed training and testing datasets were uploaded back to the Hugging Face Dataset Hub. This ensures that all modeling experiments reference a fixed and verifiable version of the prepared data.

By registering both raw and processed datasets on the Hugging Face platform, the data preparation workflow remains transparent, auditable, and reproducible.

Methodology Summary

The data preparation process involved loading version-controlled datasets from the Hugging Face platform, selecting relevant sensor variables, and performing a stratified train–test split to preserve class distribution. No feature scaling was applied, as tree-based models were selected for downstream modeling.

5 Model Building with Experimentation Tracking

This section describes the development, tuning, evaluation, and registration of machine learning models for predicting engine failure. The objective is to establish a transparent and reproducible experimentation workflow that enables systematic comparison of model performance.

5.1 Data Loading

The training and testing datasets were loaded directly from the Hugging Face Dataset Hub to ensure consistency with the versioned outputs of the data preparation stage. This approach guarantees that all modeling experiments are conducted on fixed and auditable datasets.

5.2 Model Selection and Baseline

Based on insights from exploratory data analysis, tree-based machine learning algorithms were selected due to their ability to model non-linear relationships, robustness to outliers, and tolerance to correlated features. A Decision Tree classifier was used as a baseline model to establish a performance reference point.

5.3 Hyperparameter Tuning and Experiment Tracking

Models and parameters were defined as follows: a **Decision Tree** classifier as baseline (no tuning); **Random Forest**, **Gradient Boosting**, and **XGBoost**, each with a defined **parameter grid** and **tuned** using `GridSearchCV` with recall as the scoring metric; and a **Logistic Regression** (with `StandardScaler`) as a linear baseline. **All tuned parameters** and their cross-validation scores were **logged** via `grid_search.cv_results_`, and the top configurations were recorded in an experiment log for transparency and reproducibility.

5.4 Model Evaluation

Model performance was evaluated on the held-out test dataset using metrics appropriate for imbalanced classification problems. Precision, recall, F1-score, and ROC-AUC were used to assess the trade-off between failure detection and false maintenance alerts.

The confusion matrix was analyzed to understand error distribution and to quantify the cost of missed failure events.

Table 1: Comparison of all five models on test data (best model by recall in bold)

Model	Precision	Recall	F1-score	ROC-AUC
Decision Tree	0.674	0.678	0.676	0.559
Random Forest	0.672	0.898	0.769	0.700
Logistic Regression	0.679	0.878	0.766	0.692
Gradient Boosting	**0.630**	**1.000**	**0.773**	0.686
XGBoost	0.649	0.968	0.777	0.690

A recall of 1.0 for Gradient Boosting means that the model did not miss any actual failure in the test set (zero false negatives). This is a deliberate choice for predictive maintenance, where undetected failures are typically more costly than unnecessary inspections. The trade-off is lower precision (0.63): the model flags more cases as "failure," so there are more false positives and thus more unnecessary maintenance alerts. If false alarms become too costly or frequent in practice, a model with slightly lower recall but higher precision (e.g., Random Forest or XGBoost) may be preferred. The reported metrics and confusion matrix allow stakeholders to make this trade-off explicitly.

5.5 Best Model Selection and Registration

The best-performing model among the five was selected **by recall** to minimize undetected failure events. In the reported run, **Gradient Boosting** achieved the highest recall (1.000 on the test set), followed by XGBoost (0.968) and Random Forest (0.898). Gradient Boosting was therefore chosen as the best model and registered on the Hugging Face Model Hub, along with its configuration and performance metrics. The deployed Streamlit application loads this best model at runtime from the Hub (the selected model may be Gradient Boosting, Random Forest, XGBoost, or another of the five depending on the training run).

Registering the model on the Hugging Face platform ensures reproducibility, enables version control, and supports future deployment and evaluation workflows.

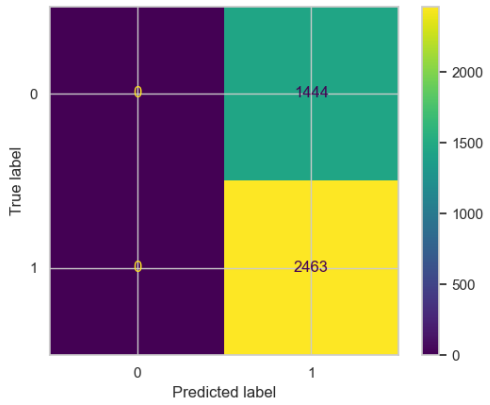

Figure 6: Confusion matrix for the best model (Gradient Boosting, selected by recall) on the test dataset

Methodology Summary

Five models were trained and evaluated: Decision Tree (baseline), Random Forest, Logistic Regression, Gradient Boosting, and XGBoost. Tree-based and ensemble models were chosen for their robustness to non-linear relationships, outliers, and correlated features; Logistic Regression with scaled features provided a linear baseline. Hyperparameter tuning for Random Forest, Gradient Boosting, and XGBoost was performed via grid search with recall as the scoring metric. The best model by recall (Gradient Boosting in the reported run) was selected and registered on the Hugging Face Model Hub.

6 Model Deployment

This section describes how the selected best model was made available for use through a web application hosted on the Hugging Face Spaces platform.

6.1 Deployment Architecture

The deployment uses a **Dockerfile** that defines the runtime environment: a Python base image, installation of dependencies from `requirements.txt`, and configuration of the Streamlit application to listen on **port 7860**, which is the default port for Hugging Face Docker Spaces. A health check and the appropriate `CMD` ensure the app starts correctly on the platform.

The application **loads the saved model from the Hugging Face Model Hub** at runtime (via `hf_hub_download`) rather than bundling the model inside the image. User **inputs** (sensor values) are collected via the Streamlit interface, **saved into a DataFrame** with the same feature columns and order as in training, and passed to the model to obtain a failure prediction.

6.2 Dockerfile and Configurations

The Dockerfile includes the following configurations:

- **FROM** `python:3.11-slim` (base image).
- **WORKDIR** `/app`.
- **RUN** system dependencies (e.g., `build-essential`, `curl` for health check).
- **COPY** `requirements.txt` and **RUN** `pip install -r requirements.txt`.
- **COPY** `app.py`.
- **EXPOSE 7860** (Hugging Face Spaces Docker default).
- **HEALTHCHECK** using `curl` to `localhost:7860/_stcore/health`.
- **CMD** to run Streamlit with `--server.port=7860`, `--server.address=0.0.0.0`, `--server.headless=t`

6.3 Key Deployment Components

- **Application script (`app.py`):** Streamlit UI for entering sensor values; lazy loading of the model on first prediction; explicit feature column list; inputs assembled into a DataFrame before prediction; error handling for load and prediction.
- **Dependencies file (`requirements.txt`):** Lists all packages required for deployment (e.g., `streamlit`, `pandas`, `scikit-learn`, `joblib`, `huggingface_hub`).
- **Hosting script (`scripts/deploy.py`):** Pushes all deployment files (`app.py`, `requirements.txt`, `Dockerfile`) to the Hugging Face Space via `HfApi().upload_file()` and ensures the Space exists with `create_repo(..., space_sdk="docker")`. The workflow runs this script so that code updates on the main branch are automatically reflected on the Space.

6.4 Hosting Platform

The application is hosted as a **Hugging Face Space** using the Docker SDK. The Space builds and runs the container; users access the app via the Space URL. The use of port 7860 is required for Hugging Face Spaces so that the platform can route traffic to the application correctly.

7 Automated GitHub Actions Workflow

An automated pipeline was implemented using GitHub Actions so that data preparation, model training, and deployment run consistently on every push to the main branch.

7.1 Workflow Configuration

The workflow is defined in `.github/workflows/pipeline.yml`. It is triggered on **push** events to the **main** branch. Pushing code updates to the main branch automatically runs the end-to-end pipeline (data preparation, model training, and deployment to the Hugging Face Space), so the live app and model stay in sync with the repository without manual steps.

7.2 Pipeline Steps

The workflow executes the following steps in sequence:

1. **Checkout:** The repository is checked out so that scripts and configuration files are available.
2. **Data preparation:** A dedicated job or step runs the data preparation script (e.g., `scripts/data_preparation.py`). This loads the dataset from the Hugging Face Dataset Hub, performs cleaning and train–test split, and uploads the processed datasets back to the Hub (and optionally saves artifacts for downstream steps).
3. **Model training:** The model training script (`scripts/model_training.py`) is executed. It loads the prepared data from the Hub, trains all five models (Decision Tree, Random Forest, Logistic Regression, Gradient Boosting, XGBoost), evaluates them on the test set, selects the best by recall, and uploads that model to the Hugging Face Model Hub.
4. **Deployment:** The deployment step pushes the application code (e.g., `app.py`, `Dockerfile`, `requirements.txt`) to the Hugging Face Space, so the Space rebuilds and serves the updated app with the latest model.

Secrets (e.g., Hugging Face token) are stored in the repository secrets and used by the workflow for Hub and Space authentication. This setup provides a single, repeatable path from code change to live deployment.

8 Output Evaluation

This section summarizes the project deliverables and how the pipeline and deployment can be verified.

8.1 Project Artifacts

The repository (see Summary of Links below) contains the full folder structure, including `data/`, `scripts/`, `notebooks/`, `reports/`, `app.py`, `Dockerfile`, `requirements.txt`, and `.github/workflows/`. The accompanying notebook includes screenshots of the repository layout and a completed GitHub Actions run (Data preparation, Model training, Deploy to HF Spaces). The Streamlit application is available at the Hugging Face Space linked below, with a screenshot of the live app (form and prediction result) in the notebook.

8.2 Repository and Workflow

The project code and configuration are maintained in a **GitHub repository**. The repository contains the master folder structure, including the `data/` subfolder, scripts for data preparation and model training, the Streamlit application and Dockerfile, and the GitHub Actions workflow file (`.github/workflows/pipeline.yml`). Inspect the repository folder structure and the Actions tab to confirm a successful run; screenshots are in the accompanying notebook.

8.3 Deployed Application

The predictive maintenance application is hosted on a **Hugging Face Space**. The Space URL is given below. Users can enter sensor values and click "Predict" to obtain a failure prediction. A screenshot of the running app is in the accompanying notebook.

8.4 Summary of Links

- **GitHub repository:** `https://github.com/siddmkrj/predictive_maintenance` (project code and workflow).
- **Hugging Face Space:** `https://huggingface.co/spaces/mukherjee78/predictive-maintenance-app`
- **Dataset:** `https://huggingface.co/datasets/mukherjee78/predictive-maintenance-engine-data`
- **Model:** `https://huggingface.co/mukherjee78/predictive-maintenance-random-forest`

Figure 7 shows the deployed Streamlit application on the Hugging Face Space.

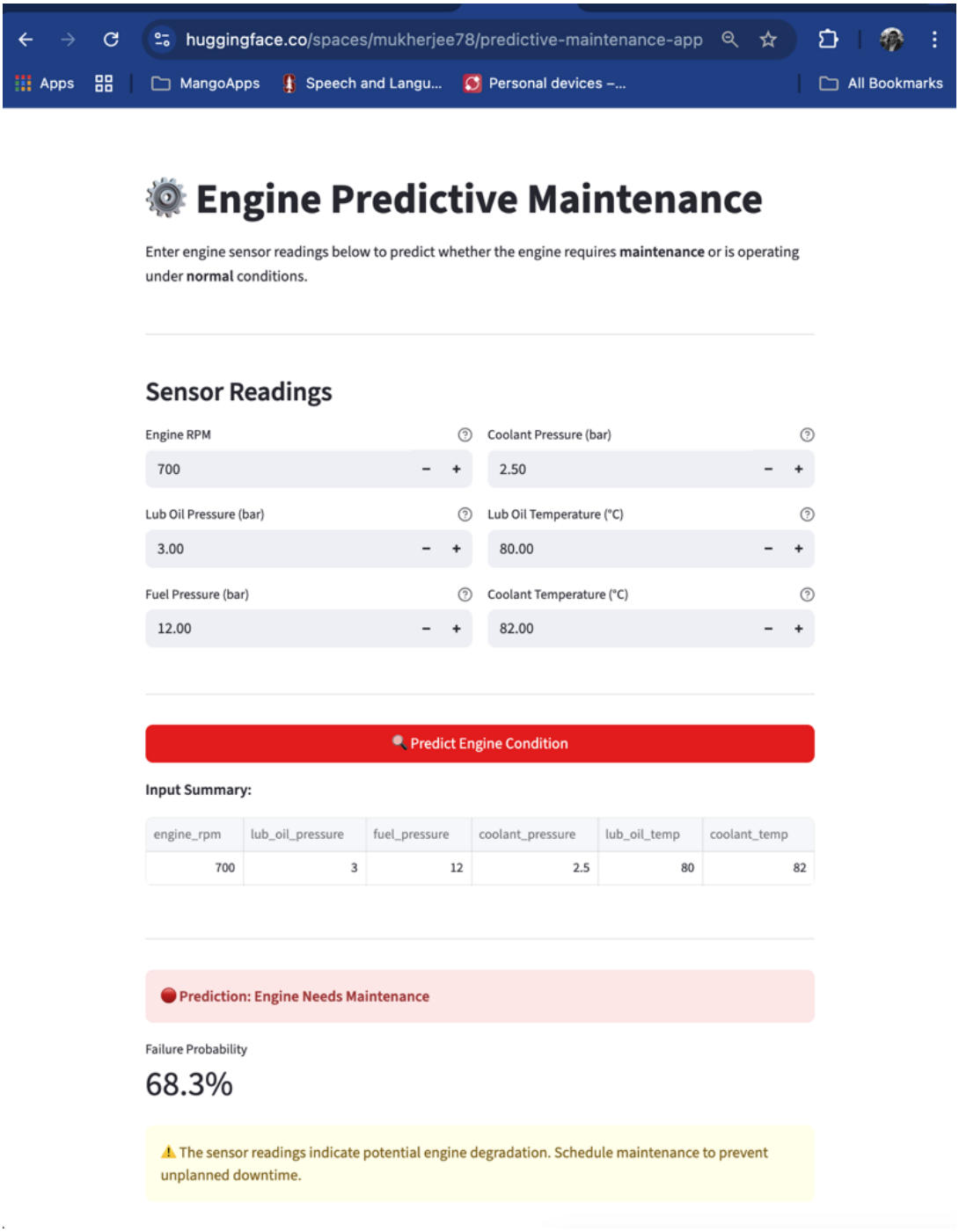

Figure 7: Streamlit app on Hugging Face Spaces: Engine Predictive Maintenance interface with sensor inputs and prediction result

9 Actionable Insights and Recommendations

This section connects the technical findings to the business context with insights and recommendations.

9.1 Insights

- **Class imbalance:** Engine failure events are relatively rare in the data. Metrics such as recall and F1-score are more meaningful than accuracy for assessing how well the model detects actual failures and balances false alarms.
- **Sensor relevance:** Pressure- and temperature-related sensors show systematic changes before failure; these signals are the most predictive and should be prioritized in monitoring and in any future sensor or feature decisions.
- **Model performance:** All five models were evaluated on the same test set. The Decision Tree baseline achieved recall 0.68 and F1 0.68. Ensemble and linear models performed better: Random Forest (recall 0.90, F1 0.77), Logistic Regression (recall 0.88, F1 0.77), Gradient Boosting (recall 1.00, F1 0.77), and XGBoost (recall 0.97, F1 0.78). Gradient Boosting was selected as best by recall (1.00) to minimize missed failures, with the trade-off of lower precision (0.63) and more potential false positives than Random Forest or XGBoost.
- **Pipeline and automation:** The GitHub Actions pipeline runs data preparation, model training, and deployment on every push to the main branch, reducing manual steps and the risk of inconsistent releases.

9.2 Recommendations

- **Operational use:** Use the deployed Streamlit app as a decision-support tool. For predictions indicating elevated failure risk, schedule inspections or maintenance rather than relying on the model output alone for safety-critical decisions.
- **Monitoring and retraining:** Log predictions and outcomes (e.g., whether maintenance was performed and whether failure occurred) to monitor model drift and to collect data for periodic retraining.
- **Extensibility:** Keep the data and model on the Hugging Face Hub and the pipeline in version control so that new sensors, features, or models can be integrated with minimal disruption.

A Appendix

This appendix contains supplementary material supporting the analyses presented in the main report. In line with the project guidelines, raw code, extended outputs, and detailed experiment logs are not included in the main body. All code, execution outputs, and additional plots are provided in HTML format in the accompanying notebook, which supports verification of results and code execution.