A Project Report

On

Predictive Maintenance of Aircraft Engines: Machine Learning Approaches for Remaining Useful Life Estimation

Submitted by

Pappuru Manikanta reddy – 21MID0056

Kothaluri Murari – 21MID0069

Boyana Rahul – 21MID0103

For

Intelligent Database Systems

MDI3004

Slot: G1+TG1, J

Component MTech in Data Science

Table of Contents

List Of Figures:

List Of Tables:

Chapter 1: Introduction to Project

1.1 Introduction

- Aircraft are complex machines with interdependent components, where the engine is critical for safety and performance. Due to extreme operational conditions like high temperatures, pressure, and stress, engines are prone to degradation, potentially leading to catastrophic failures, costly repairs, or loss of life. To address these risks, the aviation industry adopts predictive maintenance, a proactive, data-driven approach that forecasts failures using real-time sensor data. This method enhances safety, reduces operational costs, and optimizes maintenance schedules.This research focuses on developing a machine learning-based predictive model to estimate the Remaining Useful Life (RUL) of aircraft engines, leveraging NASA's C-MAPSS FD001 dataset. Key objectives include:
- **Developing Predictive Models:** Building machine learning models to predict RUL, aiming for performance comparable to deep learning and LSTM methods.
- **Feature Engineering:** Preprocessing data, normalizing features, and applying hyperparameter tuning for noise reduction and accuracy improvement.
- **Integration & Visualization:** Integrating models with PostgreSQL for data management and deploying a Flask-based web interface for dynamic result visualization.
- The dataset, stored in PostgreSQL, includes 26 operational parameters. A target column was computed using defined thresholds (optimal at 130 cycles) to enhance model accuracy. The workflow incorporates Explainable AI (XAI) for feature importance, Min-Max scaling, and detailed visualizations like boxplots for clarity. This integrated approach supports effective maintenance planning, ensuring safety and operational efficiency.

1.2 Problem Definition

Even the minor failure mode can cause devastation in terms of economy and manpower. Hence a detailed Failure Mode and Effect Analysis (FMEA) along with recovery procedures is needed. The sensors are installed at various locations on the system and the data is acquired. The type of sensors may vary from temperature, pressure, vibration, gas etc. Also, the sensor data may be either numeric or in the form of discrete or continuous signals . The challenge addressed by this project is to develop a data-driven predictive model that estimates the engine’s remaining cycles before failure (i.e., predicting the number of cycles left before shutdown), based on machine learning algorithms that estimates the Remaining useful lifetime with performance nearly equal to deep learning methods. This is critical for both operational efficiency and safety in the aerospace industry.

1.3 Project Scope

This project aims to develop a predictive maintenance system specifically designed for aircraft engines, with a primary focus on estimating the Remaining Useful Life (RUL). The following components outline the project's boundaries and expected outcomes:

1. **Data Acquisition**:
   Leverage the NASA Commercial Modular Aero-Propulsion System Simulation (CMAPSS) dataset, which contains time series data collected from various aircraft engines.Import this dataset into a PostgreSQL database to facilitate efficient data management.

2. **Data Preprocessing:**

Perform essential preprocessing tasks, including data cleaning, feature selection, calculation of the target variable (RUL), and normalization to prepare the dataset for modeling.

3. **Model Development:**

Develop and train a range of machine learning models, such as Random Forest, Support Vector Machine (SVM), Linear Regression, XGBoost, and K-Nearest Neighbors (KNN).

Assess and compare the performance of these models' using metrics like Root Mean Squared Error (RMSE) and R^2 score to determine their effectiveness.

4. **Feature Importance Analysis:**

Utilize feature importance techniques to pinpoint the most significant factors impacting engine RUL, with the aim of minimizing prediction errors and enhancing model accuracy.

5. **Visualization and User Interface:**

Design a front-end application that presents model performance metrics visually. This application will retrieve and plot pertinent data from PostgreSQL, enabling users to track and evaluate model performance. 6. Documentation and Reporting:

Create thorough documentation that outlines the methodologies employed, results achieved, and insights gained throughout the project. This documentation will serve as a valuable resource for stakeholders and future research initiatives.

7. **Limitations**:

The focus will be on predicting RUL for aircraft engines using the CMAPSS dataset, and the project may not extend to other machinery types or datasets.

The efficacy of the models will depend on the quality and representativeness of the input data.

By delineating the project scope, this initiative aims to provide a concentrated and organized approach to developing an effective predictive maintenance solution for the aerospace sector.

1.4 Motivation

The motivation behind this project stems from the critical need for enhanced predictive maintenance strategies in the aerospace industry. As global air traffic continues to rise, the demand for safe and reliable aircraft operations becomes increasingly paramount. Unexpected engine failures not only pose significant safety risks but also lead to substantial financial losses due to unplanned downtimes and costly repairs. Predictive maintenance, particularly through accurate RUL predictions, offers a proactive approach to addressing these challenges. By accurately forecasting when an engine may fail, airlines and maintenance organizations can schedule maintenance more effectively, reducing the likelihood of in-flight failures and extending the lifespan of their engines.

Furthermore, advancements in machine learning and data analytics provide unprecedented opportunities to harness large volumes of sensor data for actionable insights. By leveraging historical data and applying sophisticated machine learning algorithms, this project aims to improve the accuracy of RUL predictions, ultimately leading to more informed decision-making in maintenance operations.

Additionally, with the integration of a user-friendly front-end application that visualizes model performance metrics, stakeholders will be equipped with the necessary tools to monitor engine health proactively. This transparency enhances trust in predictive models and allows for timely interventions based on real-time data.

The goal of this project is to contribute to the development of a safer, more efficient, and cost-effective aerospace industry, where predictive maintenance becomes an integral part of operational strategies, benefiting both operators and passengers alike.

1.5 Literature Survey

Remaining Useful Life (RUL) prediction has become a critical focus in industries relying on the longevity and performance of machinery, particularly in aviation and aerospace sectors. The ability to predict the remaining cycles of aircraft engines not only helps in scheduling maintenance but also ensures optimal performance and safety. In recent years, more studies have leveraged deep learning techniques over traditional machine learning approaches to address the challenge of RUL prediction, with a focus on accurately forecasting engine failure based on sensor data, to improve operational efficiency and reduce unforeseen downtimes.

Khelif et al. [1] modelled using SVM and obtained a competitive score value of 448 compared to neural networks that had a score value of 1046. This is attributed to the high classification accuracy when using a kernel function in modelling. Due to its powerful self-learning capabilities, deep learning has already achieved significant success in the manufacturing industry [2]. A core challenge in predicting a system's Remaining Useful Life (RUL) lies in determining the precise output value for each input data point. In real-world applications, the absence of an accurate physics-based model makes it difficult to reliably assess system health status at each time step [2].

Typically, two main approaches address this RUL prediction challenge. The first assigns the desired output as the actual remaining time until failure, while the second derives output values based on a suitable degradation model. In this context, we focus on NASA's Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) dataset, a widely used multivariate benchmark for evaluating predictive deep learning algorithms. For C-MAPSS, a piecewise linear degradation model has been proposed [3], which assumes that system degradation begins only after a specific usage threshold.

Shi and Chehade [5] introduced a dual LSTM framework capable of detecting change points in sensor data, using a health index function to help identify threshold values in the degradation process. Ayodeji et al. [6] proposed a Causal Augmented Convolution Network (CaConvNet), which was further optimized with a dynamic hyperparameter search algorithm to minimize manual tuning and uncertainties, utilizing a piece-wise linear function with a threshold value of 130. Shi and Chehade [5] also advanced a dual-LSTM approach for RUL prediction by assuming the threshold at the midpoint of the degradation process for each engine. Li et al. [8] developed a GRU-based model with a high-level feature fusion block, replacing traditional fully connected layers and using the novel activation function Mish to predict aero-engine RUL, employing a piece-wise linear function with a threshold of 120 across all C-MAPSS subsets [9]. Zhang et al. [8] applied the piece-wise linear model, introducing a unique method to determine the threshold value based on dataset characteristics.

This threshold, observed empirically, varies by dataset and has since been widely adopted in related research as a standard model for RUL prediction [4]. Notably, the threshold value determined in this study is 130, based on observations from the first subset (FD001) and is adapted from [4].

In the research conducted by Zhiqiang Lyu and colleagues [11], a comprehensive analysis was performed to estimate the Remaining Useful Life (RUL) of lithium-ion batteries using both Support Vector Regression (SVR) and Random Forest Regressor models. The study achieved notable results, with an RMSE of 33.5 for the SVR model and 30.6125 for the Random Forest Regressor.

In the research conducted by Mathew Toby and colleagues [12], a thorough analysis was performed on Remaining Useful Life (RUL) prediction using the C-MAPSS dataset. The study employed ten different machine learning models, specifically focusing on Support Vector Machine (SVM), Random Forest (RF), and Gradient Boosting models applied to Dataset 1. The results revealed an RMSE of 48.17 for the SVM model, while the Random Forest model achieved a significantly lower RMSE of 25.86, and the Gradient Boosting model recorded an RMSE of 27.45.

In the study by Sheng et al. [13], a novel self-adapting deep learning network (CSDLN) was developed to predict the Remaining Useful Life (RUL) of aircraft engines. This model achieved a remarkable RMSE of 15.977, showcasing significant improvements in prediction accuracy over previous methods. The CSDLN integrates a multi-branch 1D involution neural network for feature extraction and a trend recognition unit for identifying degradation trends. The authors employed advanced techniques such as comparative and ablation experiments to validate the model's performance, demonstrating its effectiveness in handling complex RUL prediction tasks in aerospace applications.

In the study conducted by Fernando Sánchez Lasheras and colleagues [14], a hybrid model combining Support Vector Machines (SVM) and the Autoregressive Integrated Moving Average (ARIMA) technique was developed to predict the Remaining Useful Life (RUL) of aircraft engines. Utilizing the C-MAPSS dataset, the researchers achieved a Root Mean Squared Error (RMSE) of 39.6843, indicating a high level of accuracy in their predictions.

Chapter 2: Project Planning

2.1 Project Schedule

The project is planned to span from **August 1, 2024**, to **October 5, 2024**. The schedule includes various phases, tasks, and deadlines as outlined below:

Task	Start Date	End Date	Duration	Required Skills
Project Initialization	1-Aug-24	7-Aug-24	1 Week	Project management tools
Requirements Gathering	8-Aug-24	15-Aug-24	1 Week	HTML, kaggle
Design Phase	16-Aug-24	25-Aug-24	10 Days	Figma, Adobe XD
Implementation Phase	26-Aug-24	15-Sep-24	3 Weeks	Python, Keras, Flask
Testing Phase	16-Sep-24	30-Sep-24	2 Weeks	Pytest

Deployment	1-Oct-24	4-Oct-24	4 Days	AWS, Docker
Project Closure	5-Oct-24	5-Oct-24	1 Day	Documentation tools

Table 1:project schedule

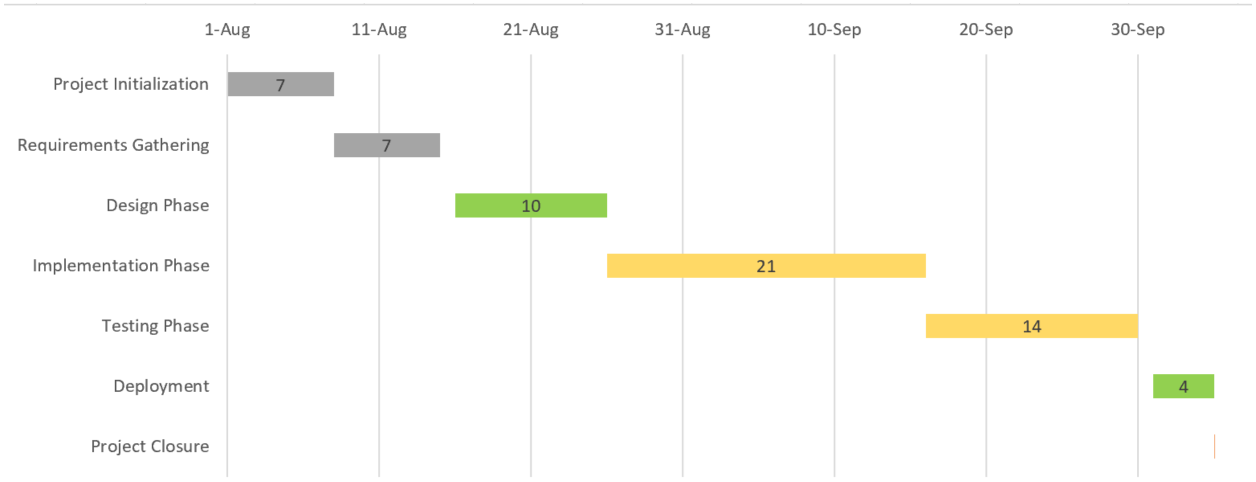


Figure 0 : gantt chart

Colours resembles the member who worked on team:

- All members
- manikanta
- murari
- rahul

2.2 Effort and Resource Estimation

The effort and resource estimation for this project follows the **COCOMO (Constructive Cost Model)**, which provides a structured approach to estimate the effort, duration, and costs involved based on the project size and complexity.

Project Size: The size of the project is estimated in terms of lines of code (LOC). Assuming the final deliverable will consist of approximately **20,000 LOC**.

Estimated Effort: After calculating, we estimate the total effort required for the project is approximately **10 weeks**. **Task Breakdown, Effort, Resources, and Timeline:**

- **Project Initialization**
  - **Timeline**: **August 1 – August 7, 2024**
  - **Effort**: 1 week
  - **Description**: Establish initial project setup, team alignment, and project goals.

  - **Resources Needed**:
    - Team meeting tools: Zoom, Slack
    - Project management software: Asana, Trello
- **Requirements Gathering**
  - **Timeline**: **August 8 – August 15, 2024**
  - **Effort**: 1 week
  - **Description**: Collect, document, and finalize project requirements and scope.
  - **Resources Needed**:
    - Documentation tools: Google Docs, Confluence
    - Data sources: NASA C-MAPSS dataset for RUL prediction
- **Design Phase**
  - **Timeline**: **August 16 – August 25, 2024**
  - **Effort**: 1.5 weeks
  - **Description**: Develop the system architecture and user interface designs.
  - **Resources Needed**:
    - Design tools: Figma, Adobe XD
    - Collaboration tools: Whiteboard for brainstorming, Miro
- **Implementation Phase**
  - **Timeline**: **August 26 – September 22, 2024**
  - **Effort**: 4 weeks
  - **Description**: Core development phase, including backend and frontend coding, model training, and component integration.
  - **Resources Needed**:
    - Development environment: IDE (VS Code, PyCharm)
    - Version control: Git
    - Libraries: Keras, TensorFlow, Flask
- **Testing Phase**
  - **Timeline**: **September 23 – September 30, 2024**
  - **Effort**: 1 week
  - **Description**: Conduct rigorous testing of core modules, model accuracy, and system integration.
  - **Resources Needed**:
    - Testing frameworks: Pytest
    - Test management tools: TestRail, Jira
    - Additional data: Test datasets for RUL model validation
- **Deployment**
  - **Timeline**: **October 1 – October 3, 2024**
  - **Effort**: 3 days
  - **Description**: Deploy the application, setting up cloud infrastructure and database configuration.
  - **Resources Needed**:
    - Cloud services: AWS or Azure for hosting
    - Containerization tools: Docker
    - Deployment scripts
- **Project Closure**
  - **Timeline**: **October 4 – October 5, 2024**
  - **Effort**: 2 days
  - **Description**: Conduct final review, documentation completion, and project handover.
  - **Resources Needed**:
    - Documentation tools: Confluence, Google Docs
    - Review meetings: Final wrap-up and project evaluation meeting

Chapter 3: Requirement Gathering and Analysis

3.1 Dataset

In this project, the dataset plays a crucial role in building the model and achieving the desired outcomes. The dataset consists of multiple features that provide valuable insights for analysis. Below are the details of the dataset used:

- **Dataset Source**: The dataset is obtained from NASA. the dataset is relevant and up-to-date.
- **Dataset Description**:
  - **Type**: The dataset is a structured dataset comprising [number] records and 26 features.
  - **Key Features**: The key features of the dataset include:
    - **Engine**: Identifier for the engine (categorical).
    - **Time**: Time stamp for the recorded data (numerical).
    - **op_setting_1**: Operational setting parameter 1 (numerical).
    - **op_setting_2**: Operational setting parameter 2 (numerical).
    - **op_setting_3**: Operational setting parameter 3 (numerical).
    - **s1 to s21**: Sensor measurements from s1 to s21 (numerical). These represent various sensor readings related to the engine performance, with each sensor providing critical data points.
- **Data Quality**:
  - The dataset is evaluated for completeness and accuracy. Missing values and anomalies are identified and addressed through techniques such as imputation or removal.
- **Data Preprocessing**:
  - Data preprocessing steps include:
    - **Normalization**: Scaling numerical features to a common range.
    - **Encoding**: Converting categorical variables into numerical format, if necessary.
    - **Splitting**: Dividing the dataset into training and testing sets, typically using an 70/30 split.

3.2 Data Modeling

Data modeling is essential for deriving insights from the dataset and creating predictive models. The following points outline the approach for data modeling in this project:

- **Model Selection**:

In this phase, various machine learning models were developed, trained on the normalized training dataset, and subsequently evaluated using the normalized validation and testing datasets to predict the

Remaining Useful Life (RUL) of 100 engines. We have taken basic LR model to Ensemble learning XGBoost model. These are selected because these are widely used and adaptable to regression models. The predictive results were stored in PostgreSQL for future reference and for dynamic rendering on a web interface. The primary objective of these models was to accurately estimate the number of operational cycles remaining from the current cycle, providing valuable insights into engine health and maintenance requirements.

- Random Forest
- Support vector Regressor
- Linear Regressor
- XGBoost
- KNN Regressor

Each model's performance was evaluated based on Root Mean Squared Error (RMSE) and R² score. For an ideal model, RMSE should be as close to 0 as possible, indicating minimal prediction error, and the R² score should approach 1, signifying a strong fit to the data.

5.2 Hyper Parameter Tuning:

To optimize model performance, GridSearchCV and RandomizedSearchCV were employed for hyperparameter tuning. These techniques identified the most efficient parameter settings for each model, leading to improved prediction accuracy and overall performance. The tuned parameters significantly enhanced the models' ability to generalize to unseen data, ensuring robust RUL predictions.

6. Model Evaluation

After model development and training, overfitting and underfitting were assessed by calculating RMSE and R² scores for both training and validation datasets. A model is balanced when both scores are nearly identical. The models were then tested on a dataset containing sensor data from 100 engines, with predictions focused on identifying remaining cycles from a given point. These predictions were compared with a reference dataset that provides the actual remaining cycles, ensuring accurate model evaluation.

6.1 Best model Selection

The XGBoost Regressor was selected as the best model due to its robust performance, demonstrating neither overfitting nor underfitting. It achieved an acceptable RMSE (<25) and an R² score (65-70), metrics typically associated with deep learning models, thus aligning well with the project's objectives.

Model Comparison				
Model	**Training RMSE**	**Training R2 Score**	**Testing RMSE**	**Testing R2 Score**
Random Forest	**15.79**	**0.86**	**28.15**	**0.54**
Support vector Regressor	**19.76**	**0.79**	**23.7**	**0.67**

Linear Regressor	**22.62**	**0.73**	**29.07**	**0.51**
XGBoost	**20.58**	**0.77**	**23.80**	**0.67**
KNN	**29.16**	**0.55**	**30.73**	**0.45**

Table2: Model performance Comparison Table

This model is serialized into a pickle file and stored within the Flask application to facilitate RUL predictions based on input from the form data.

7. Implementation of Expandable AI(XAI):

After selecting the most effective model, Explainable Artificial Intelligence (XAI) techniques were utilized to determine the significance of each feature in predicting RUL. This process entails mapping feature importance, quantifying the contribution of each attribute to the overall outcome. By leveraging the XGBoost model, we assessed feature importance to identify which factors notably impact RUL predictions, we can observer in figure 10. Each feature is assigned an importance score that indicates its predictive strength, offering essential insights into the key drivers of model performance.

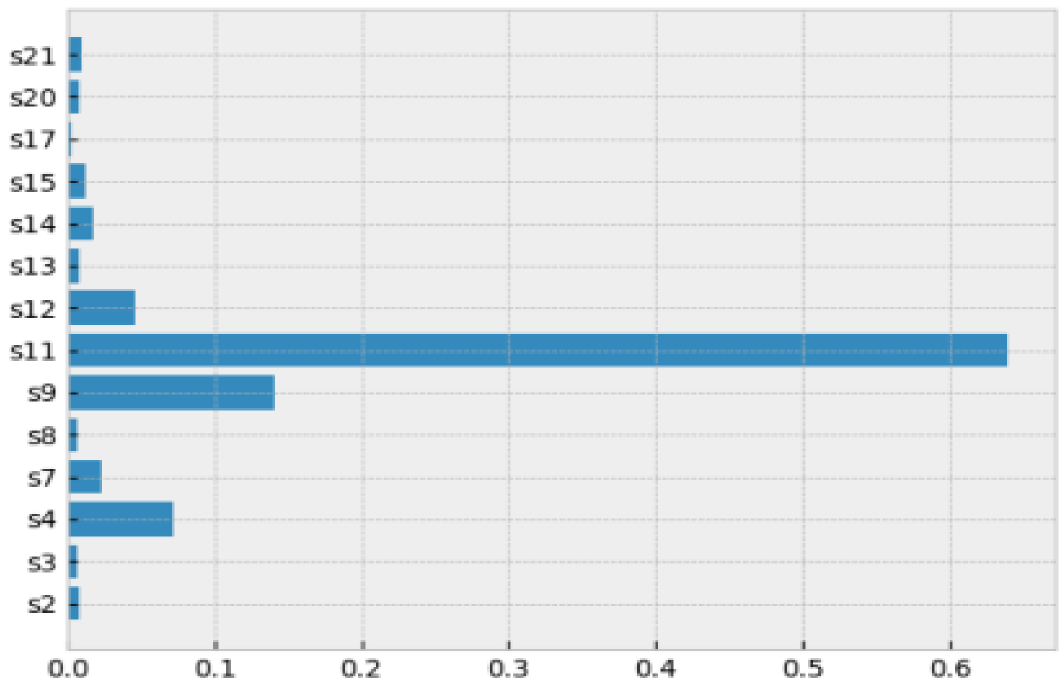


Figure-1: Feature importance using XGBoost technique.

8. Web page rendering:

We have developed a Flask-based web application that dynamically renders the analysis of the models, adapting in real-time as the data within PostgreSQL evolves due to changes in the dataset or fluctuations in model performance. This web interface serves as a comprehensive platform, showcasing the various preprocessing techniques implemented, the predictive outcomes of the models, detailed descriptions of the datasets, and much more.

Additionally, the application enables users to input sensor data directly, facilitating the dynamic prediction of the Remaining Useful Life (RUL) for specific engines utilizing the optimized model encapsulated within the pickle file. This integration not only enhances user engagement but also provides critical insights into engine health, thereby supporting informed decision-making regarding maintenance and operational efficiency. The architecture of the web application ensures seamless interaction with the underlying database, fostering a responsive environment that reflects the latest analytical results and model predictions.

Chapter 4 : Designs

4.1 Architecture diagram

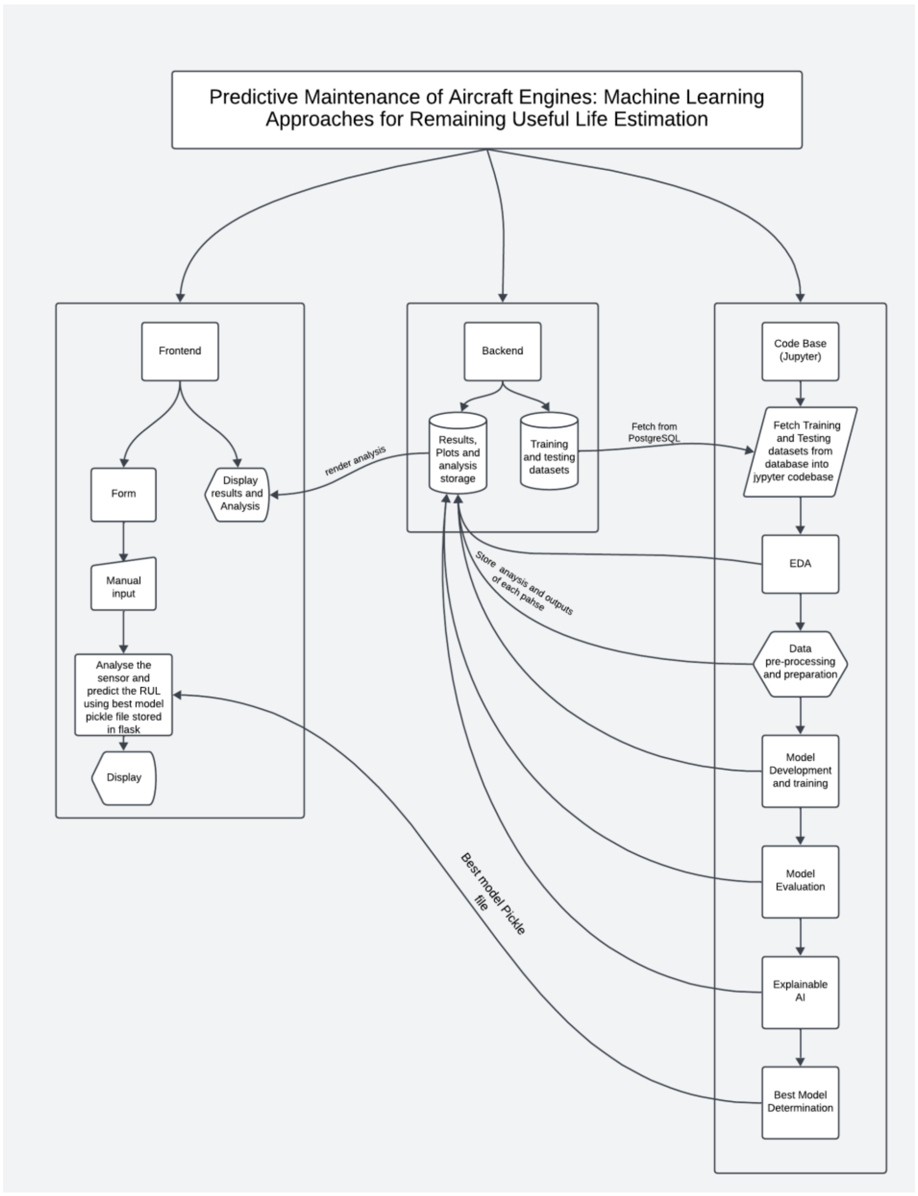

Predictive Maintenance of Aircraft Engines: Machine Learning Approaches for Remaining Useful Life Estimation
Frontend
Form
Display results and Analysis
Manual input
Analyse the sensor and predict the RUL using best model pickle file stored in flask
Display
Backend
Results, Plots and analysis storage
Training and testing datasets
render analysis
Fetch from PostgreSQL
Store anaysis and outputs of each pahse
Best model Pickle file
Code Base (Jupyter)
Fetch Training and Testing datasets from database into jypyter codebase
EDA
Data pre-processing and preparation
Model Development and training
Model Evaluation
Explainable AI
Best Model Determination

Figure 2:Architecture of the project

4.2 ML Model and Flow Charts

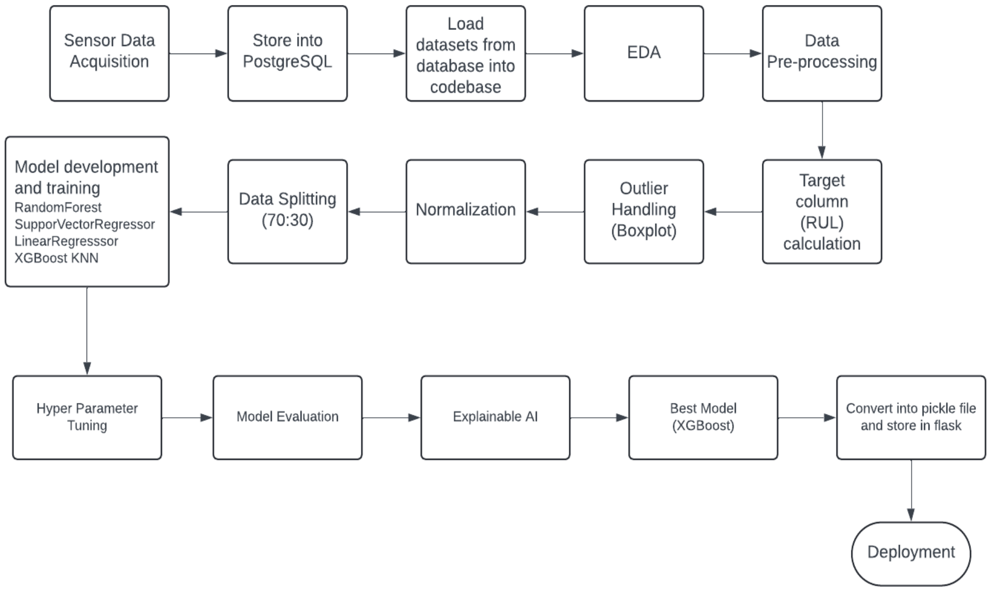

Sensor Data Acquisition
Store into PostgreSQL
Load datasets from database into codebase
EDA
Data Pre-processing
Target column (RUL) calculation
Outlier Handling (Boxplot)
Normalization
Data Splitting (70:30)
Model development and training RandomForest SupporVectorRegressor LinearRegresssor XGBoost KNN
Hyper Parameter Tuning
Model Evaluation
Explainable AI
Best Model (XGBoost)
Convert into pickle file and store in flask
Deployment

Figure 3:Operational Workflow for Engine RUL Analysis and Prediction

Chapter 5: Development

5.1 Tools Description and Development Approach Used

In this project, a variety of tools and technologies were employed to facilitate the development of a predictive maintenance system for aircraft engines, focusing on the prediction of Remaining Useful Life (RUL). Below is an overview of the key tools used and the development approach taken:

Technologies and libraries Description

Technologies Used

PostgreSQL: A powerful open-source relational database system used for efficient data management and storage. It supports complex queries, indexing, and data retrieval, making it ideal for handling large-scale engine health data.

Python: Core language for implementing machine learning algorithms and performing data analysis. Its libraries, such as Pandas, Numpy, and Scikit-learn, provide extensive functionalities for data manipulation, processing, and statistical analysis.

TensorFlow: An open-source machine learning framework used for building, training, and deploying machine learning models. It supports deep learning algorithms and provides scalability for production environments.

Keras: A high-level neural networks API written in Python and capable of running on top of TensorFlow. Keras simplifies the process of building deep learning models, allowing for quick prototyping and experimentation.

Apache Kafka: A distributed event streaming platform used for real-time data ingestion. It enables the system to handle high-throughput data streams, facilitating real-time engine health monitoring and prediction.

Docker: Containerization tool used for packaging the application along with its dependencies into a portable container. Docker ensures consistency across various environments and simplifies deployment.

Flask: A micro web framework in Python, used for building lightweight web applications and APIs. Flask enables seamless interaction between the frontend and backend components of the application.

React.js: A JavaScript library for building interactive user interfaces. React enables the development of dynamic and responsive frontend components, improving user experience.

Apache Spark: A powerful data processing framework used for handling large-scale datasets. Spark provides high-speed computation and facilitates distributed data processing, making it suitable for engine health data analysis.

AWS: Cloud platform used for scalable infrastructure. AWS provides resources such as EC2, S3, and RDS to support the deployment, storage, and processing of data-intensive applications.

Libraries Used
Pandas: A powerful data analysis library in Python that allows for efficient data manipulation and handling of tabular data. Pandas is essential for data preprocessing and cleaning tasks.

Numpy: Fundamental package for numerical computing in Python. It provides support for arrays and matrices, including mathematical functions required for data analysis and manipulation.

Scikit-learn: A machine learning library in Python that offers various algorithms for classification, regression, clustering, and dimensionality reduction. It simplifies model implementation and evaluation.

Matplotlib: A comprehensive library for creating static, animated, and interactive visualizations in Python. It is used to create plots and graphs to analyze data patterns.

Seaborn: An extension of Matplotlib, Seaborn provides aesthetic statistical graphics and enhances data visualization, making it easier to interpret complex data.

XGBoost: An optimized gradient boosting library used for creating highly efficient, flexible, and portable machine learning models. XGBoost is commonly used for predictive modeling.

NLTK: The Natural Language Toolkit, a suite of libraries and programs for symbolic and statistical natural language processing in Python. NLTK is used for text processing tasks.

SQLAlchemy: A SQL toolkit and Object Relational Mapper (ORM) for Python. SQLAlchemy provides tools for interacting with databases, making it easier to query and manipulate data.

Plotly: A graphing library in Python that enables the creation of interactive, web-based visualizations. It is used for dashboard creation and real-time data visualization**.**

Development Approach

Data Acquisition and Storage

Sensor Data Acquisition: Gather real-time or historical sensor data from engines.
Store into PostgreSQL: Save the acquired sensor data into PostgreSQL for organized storage, ensuring it's accessible for analysis and modeling. Data will be continuously updated in PostgreSQL as new sensor data is received.

Data Loading and Initial Analysis

Load Datasets from Database into Codebase: Retrieve the stored sensor data from PostgreSQL into the working codebase for further processing.
Exploratory Data Analysis (EDA): Perform EDA to understand the data distribution, identify patterns, and spot anomalies or missing values. Any findings or changes during EDA will be stored and updated in PostgreSQL to maintain data consistency.

Data Preprocessing

Data Pre-processing: Clean and transform the data as necessary, including handling missing values or erroneous entries.
Target Column (RUL) Calculation: Calculate the target variable, Remaining Useful Life (RUL), which predicts the engine's health status. The processed data, along with the RUL calculation, is saved in PostgreSQL and updated whenever preprocessing changes are applied.

Data Cleaning and Transformation

Outlier Handling (Boxplot): Identify and handle outliers using statistical methods, such as boxplots, to improve model accuracy.

Normalization: Scale the data to ensure uniformity across features, essential for optimal model performance. Both steps (outlier handling and normalization) are stored and updated in PostgreSQL after any adjustments.

Data Splitting and Model Selection

Data Splitting (70:30): Divide the data into training (70%) and testing (30%) sets for model training and validation. The split datasets are saved in PostgreSQL.
Model Development and Training: Use various regression algorithms (Random Forest, Support Vector Regressor, Linear Regressor, XGBoost, KNN) to train models on the training set. The trained models are regularly updated in PostgreSQL as part of the system's machine learning assets.

Model Optimization and Evaluation

Hyperparameter Tuning: Fine-tune the model parameters to enhance performance. The tuned parameters are recorded in PostgreSQL.
Model Evaluation: Evaluate model performance using metrics such as MAE, MSE, and R-squared, and store evaluation results in PostgreSQL for tracking model improvements.

Explainable AI and Model Selection

Explainable AI: Apply techniques to make the model's predictions interpretable, helping stakeholders understand how the model makes decisions. Explanations are saved in PostgreSQL, providing insights alongside model predictions.
Best Model Selection (XGBoost): Based on evaluation, choose the best-performing model (e.g., XGBoost) for deployment. The chosen model is saved and updated in PostgreSQL.

Model Serialization and Deployment

Convert into Pickle File and Store in Flask: Serialize the best model into a pickle file and integrate it into the Flask framework for deployment.
Deployment: Deploy the model as an API or web application using Flask, allowing real-time predictions. The deployment status and any updates are documented in PostgreSQL**.**

5.2 Code with visualizations

Initial Data Loading and Exploration:

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('bmh')
import psycopg2
import pandas as pd

# Establish connection to PostgreSQL
# Connect to PostgreSQL
def connect_db():
    return psycopg2.connect(
        host="localhost",       # If using remote, replace with your remote host address
        database="IDBMS DATASETS", # Replace with your database name
        user="postgres",    # Replace with your PostgreSQL username
        password="1234"
    )

connection = connect_db()
# Create a cursor object to interact with PostgreSQL
cursor = connection.cursor()
# SQL query to select all data from your table
training_query = "SELECT * FROM train_fd001"
testing_query = "SELECT * FROM testing_fd001"
traing_rul_query = "SELECT * FROM testing_rul"
```

# Read the data into a pandas DataFrame

data_train = pd.read_sql(training_query, connection)

data_val = pd.read_sql(testing_query, connection)

y_val = pd.read_sql(traing_rul_query, connection)

Close the connection

cursor.close()

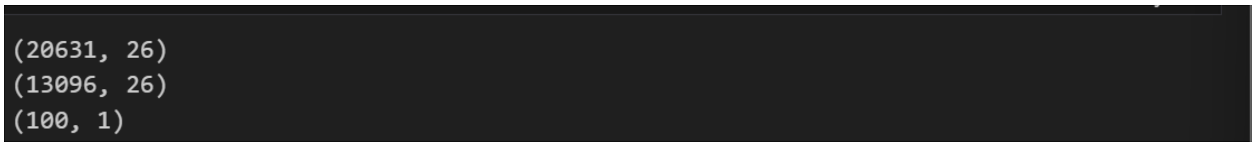
connection.close()

# Check the first few rows of the dataset

print(data_train.shape)

print(data_val.shape)

print(y_val.shape)

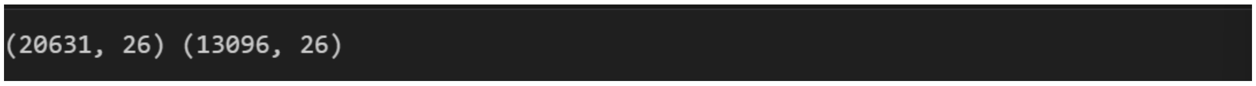


```
(20631, 26)
(13096, 26)
(100, 1)
```

df = data_train.copy()

df_val = data_val.copy()

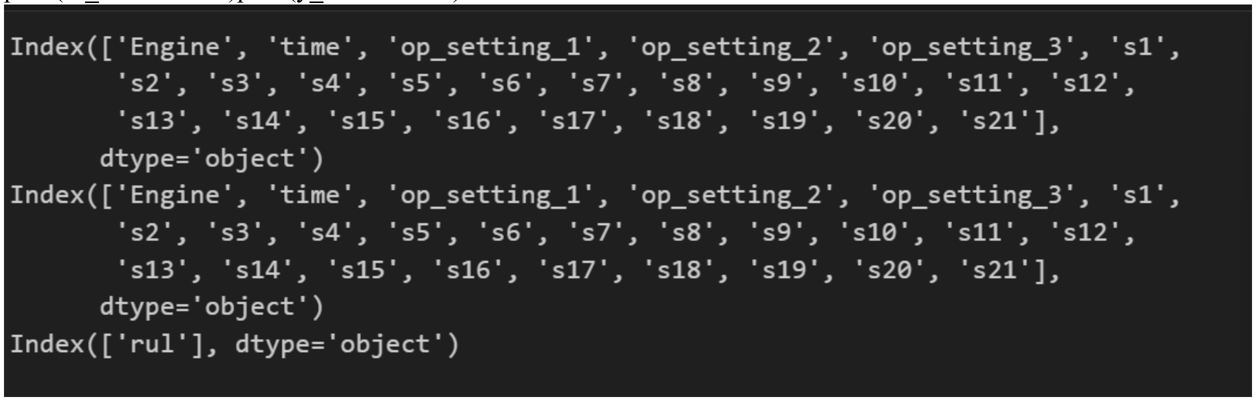
print(df.shape , df_val.shape)



```
(20631, 26) (13096, 26)
```

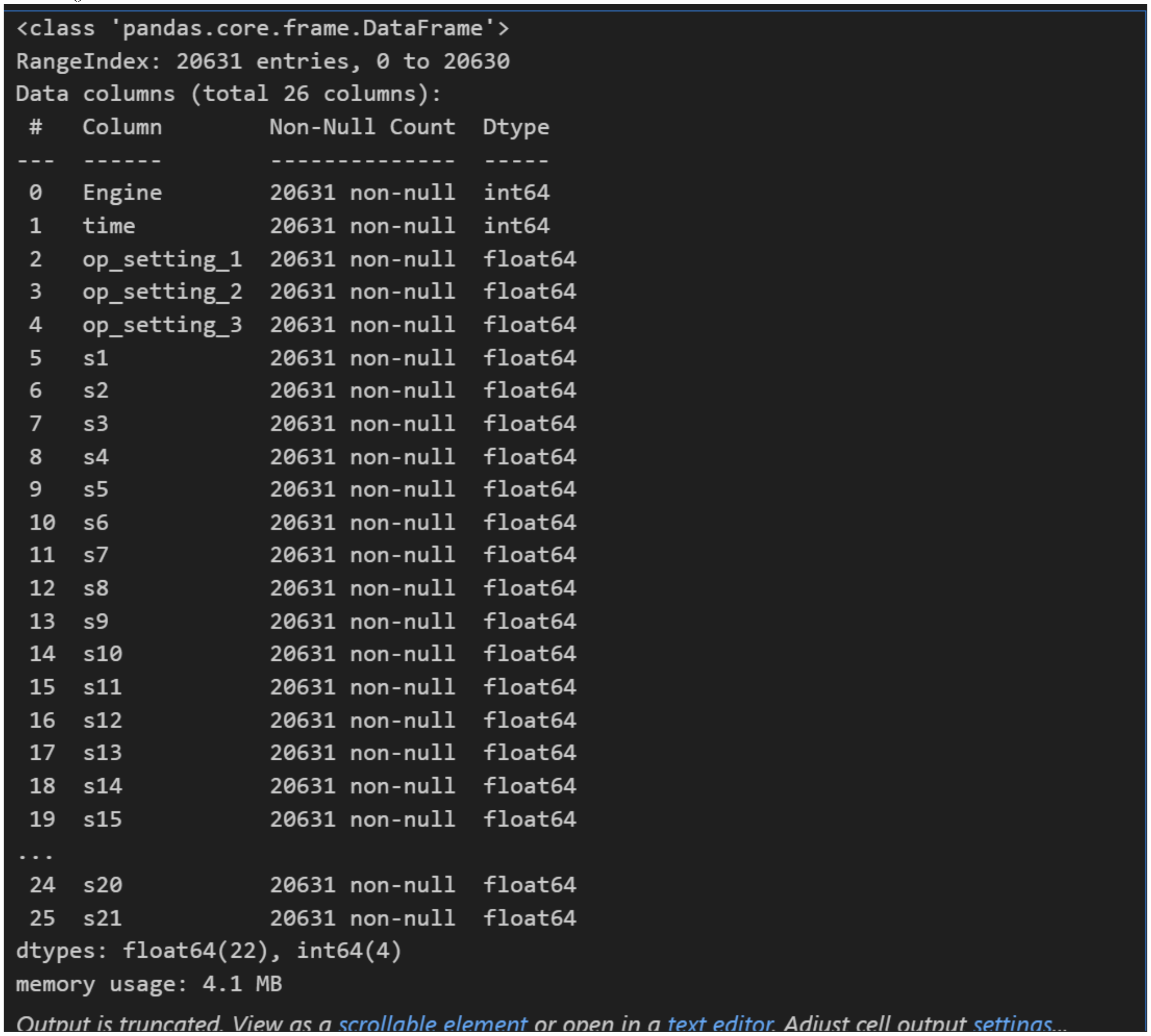
print(df.columns)

print(df_val.columns)print(y_val.columns)



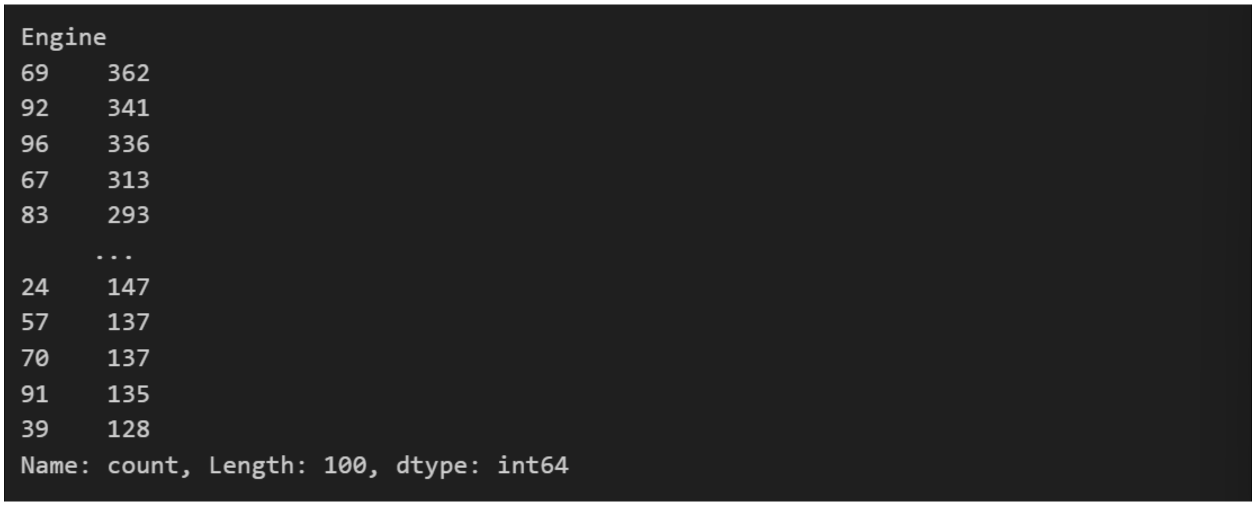
```
Index(['Engine', 'time', 'op_setting_1', 'op_setting_2', 'op_setting_3', 's1',
       's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12',
       's13', 's14', 's15', 's16', 's17', 's18', 's19', 's20', 's21'],
      dtype='object')
Index(['Engine', 'time', 'op_setting_1', 'op_setting_2', 'op_setting_3', 's1',
       's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12',
       's13', 's14', 's15', 's16', 's17', 's18', 's19', 's20', 's21'],
      dtype='object')
Index(['rul'], dtype='object')
```

df.info()



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20631 entries, 0 to 20630
Data columns (total 26 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Engine        20631 non-null  int64  
 1   time          20631 non-null  int64  
 2   op_setting_1  20631 non-null  float64
 3   op_setting_2  20631 non-null  float64
 4   op_setting_3  20631 non-null  float64
 5   s1            20631 non-null  float64
 6   s2            20631 non-null  float64
 7   s3            20631 non-null  float64
 8   s4            20631 non-null  float64
 9   s5            20631 non-null  float64
 10  s6            20631 non-null  float64
 11  s7            20631 non-null  float64
 12  s8            20631 non-null  float64
 13  s9            20631 non-null  float64
 14  s10           20631 non-null  float64
 15  s11           20631 non-null  float64
 16  s12           20631 non-null  float64
 17  s13           20631 non-null  float64
 18  s14           20631 non-null  float64
 19  s15           20631 non-null  float64
...
 24  s20           20631 non-null  float64
 25  s21           20631 non-null  float64
dtypes: float64(22), int64(4)
memory usage: 4.1 MB
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

```
# usefull info of dataset

df_info = pd.concat ( [data_train.isna().sum() , data_train.nunique() , data_train.dtypes] , axis = 1  )

df_info.columns = ['missing value' , 'no.of unique value' , 'dtype']

df_info

df.loc[:,['Engine','time']].describe()

df.loc[:,'s2':'s21'].describe().transpose()

engine_counts = df['Engine'].value_counts()


print(engine_counts)
```



```
Engine
69    362
92    341
96    336
67    313
83    293
     ...
24    147
57    137
70    137
91    135
39    128
Name: count, Length: 100, dtype: int64
```

Data Visualization: -

Selecting and Plotting Sensor Data for a Specific Engine

```
# Plot individual line plots for each setting and sensor
for ax, column in zip(axes.flatten(), columns_to_plot):
    ax.plot(engine_data['time'], engine_data[column], label=column)
    ax.set_title(f'{column} over Cycles')
    ax.set_xlabel('Cycle')
    ax.set_ylabel('Value')
    ax.legend()
    ax.grid(True)

# Adjust the layout so that subgraphs do not overlap
plt.tight_layout()
plt.show()
```

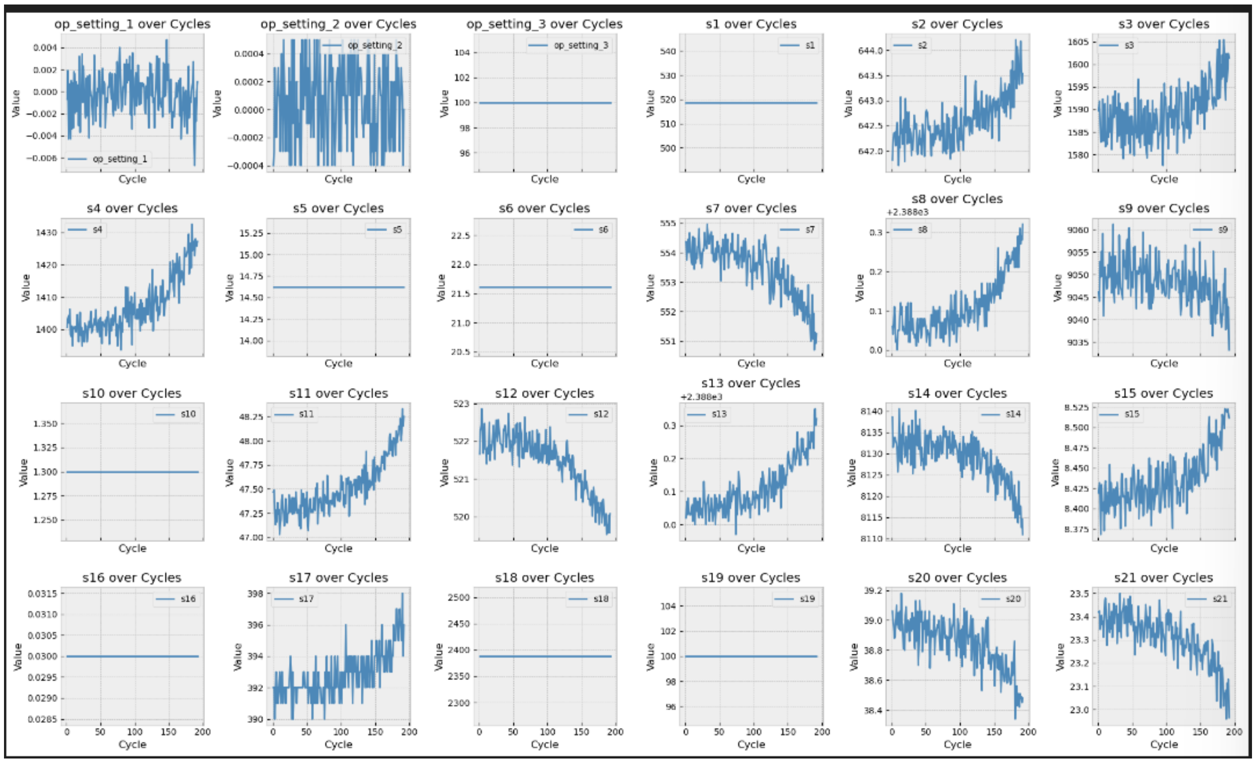
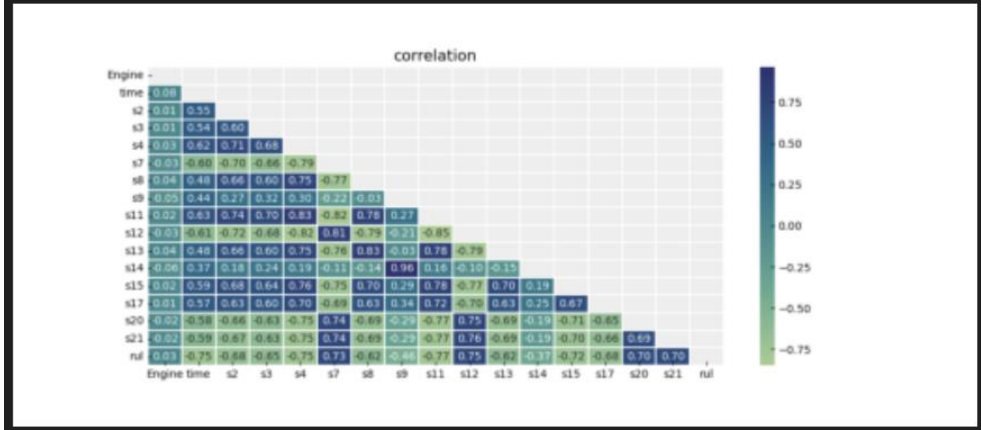


Figure 4:Plotting Sensor Data for a Specific Engine

Correlation Analysis and Heatmap of Sensor and Setting Variables

```
df_corr = df.corr()

mask = np.tril(np.ones(df_corr.shape),k = -1).astype(bool)

df_corr = df_corr.where(mask)

plt.figure(figsize = (12,5))

plt.grid() , plt.title('correlation')

sns.heatmap(df_corr , annot=True , fmt = '0.2f' , cmap='crest' , linewidths=0.01)


insert_plot_to_db('corr_graph',plt)

retrive_from_pgres('corr_graph')
```

```
Inserted new plot with ID corr_graph.
Retrieved binary data length: 111929
Displaying plot from database
```



```
Displayed
```

Figure 5: Correlation Analysis and Heatmap of Sensor and Setting Variables

Engine Count Distribution Across Observed Time Cycles

```
# detect more than 95% correlation
high_corr = []
for col in df_corr.columns:
    for row in df_corr.index:
        if abs(df_corr.loc[col , row]) > 0.95 :
            high_corr.append((col , row))
high_corr
# these 2 feature has very high correlation , no need for both of them , we can drop one of them
# df.drop(columns = ['s9'] , inplace = True)
df.columns
engine_counts = df['Engine'].value_counts().reset_index()
engine_counts.columns = ['Engine', 'count']

# Drawing bars with seaborn
plt.figure(figsize=(12, 8))
sns.barplot(x='Engine', y='count', data=engine_counts)
```

```
plt.title('Count of Each Engine')

plt.xlabel('Engine Number')

plt.ylabel('Count')

plt.xticks(rotation=90)

plt.show()
```

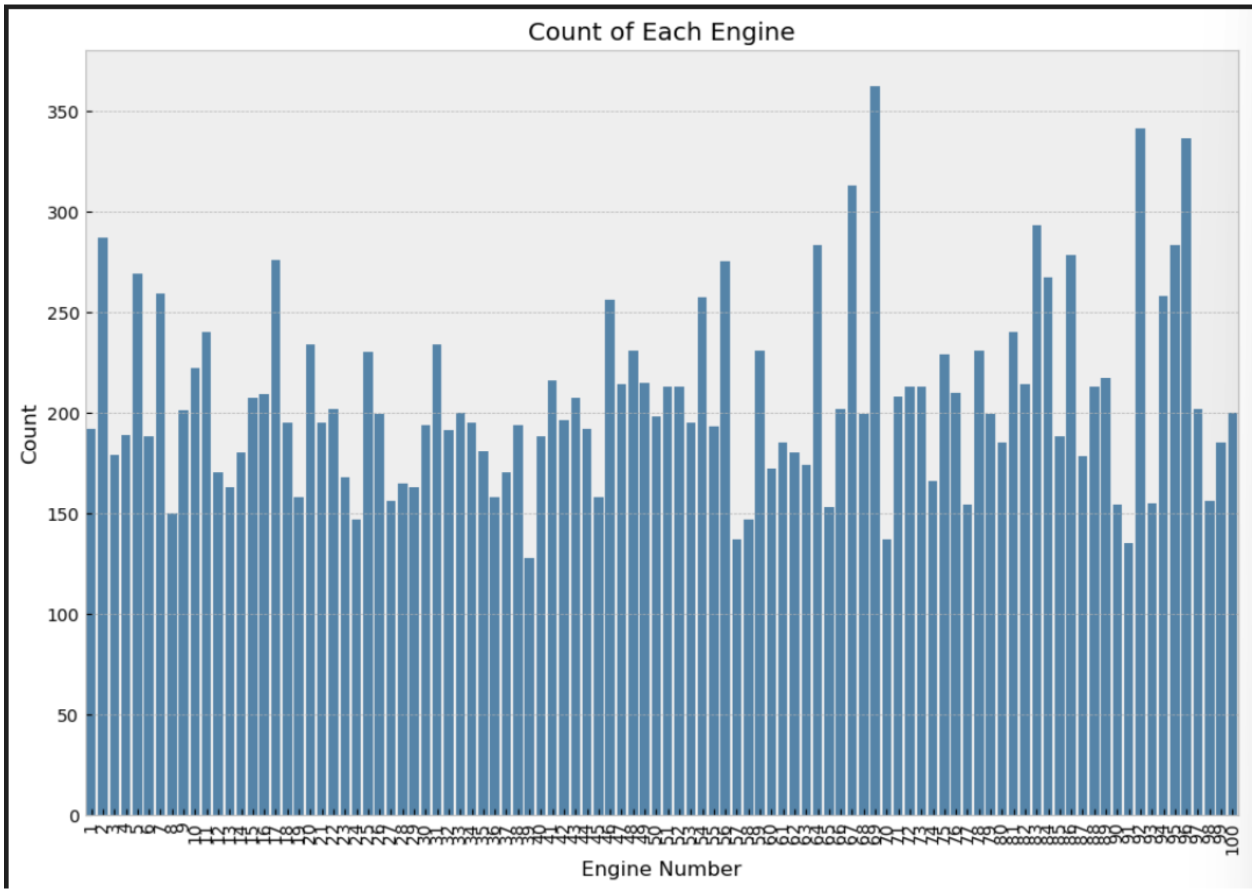
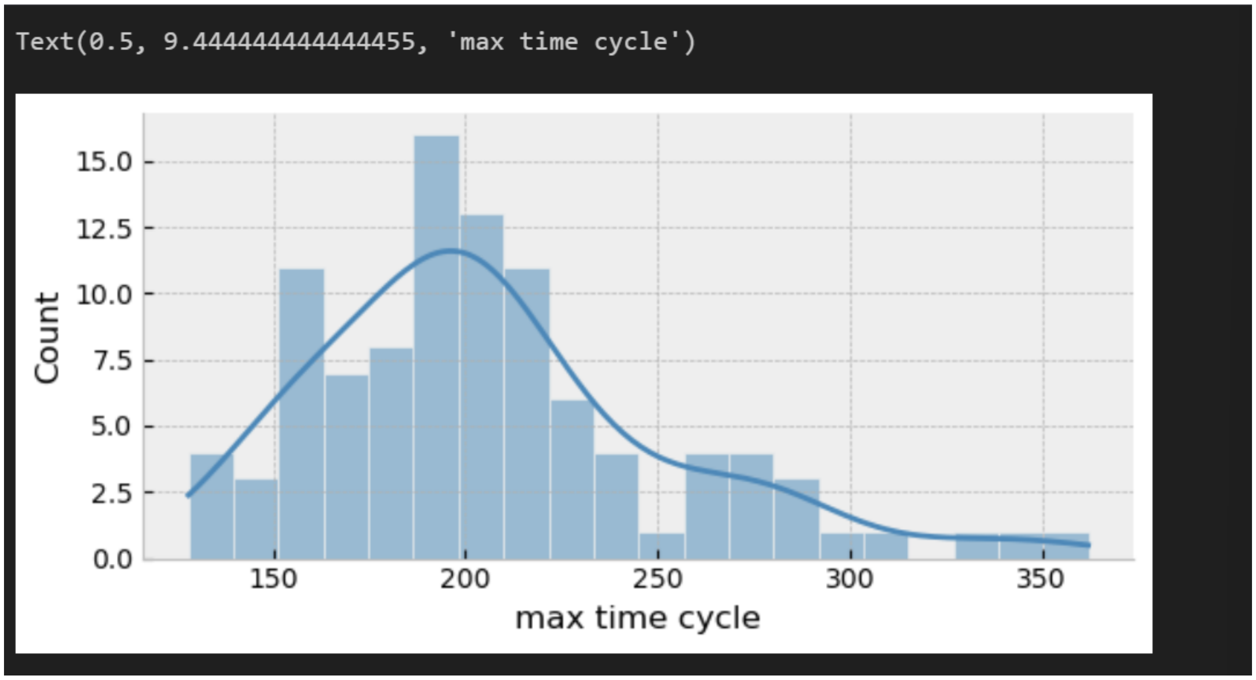


Figure 6: Engine Count Distribution Across Observed Time Cycles

Distribution of Maximum Time Cycles for Engines

```
#no of engines have particular rul.

index_names = ['Engine', 'time']

max_time_cycles=df[index_names].groupby('Engine').max()

sns.displot(max_time_cycles['time'],kde=True,bins=20,height=3,aspect=2)

plt.xlabel('max time cycle')
```

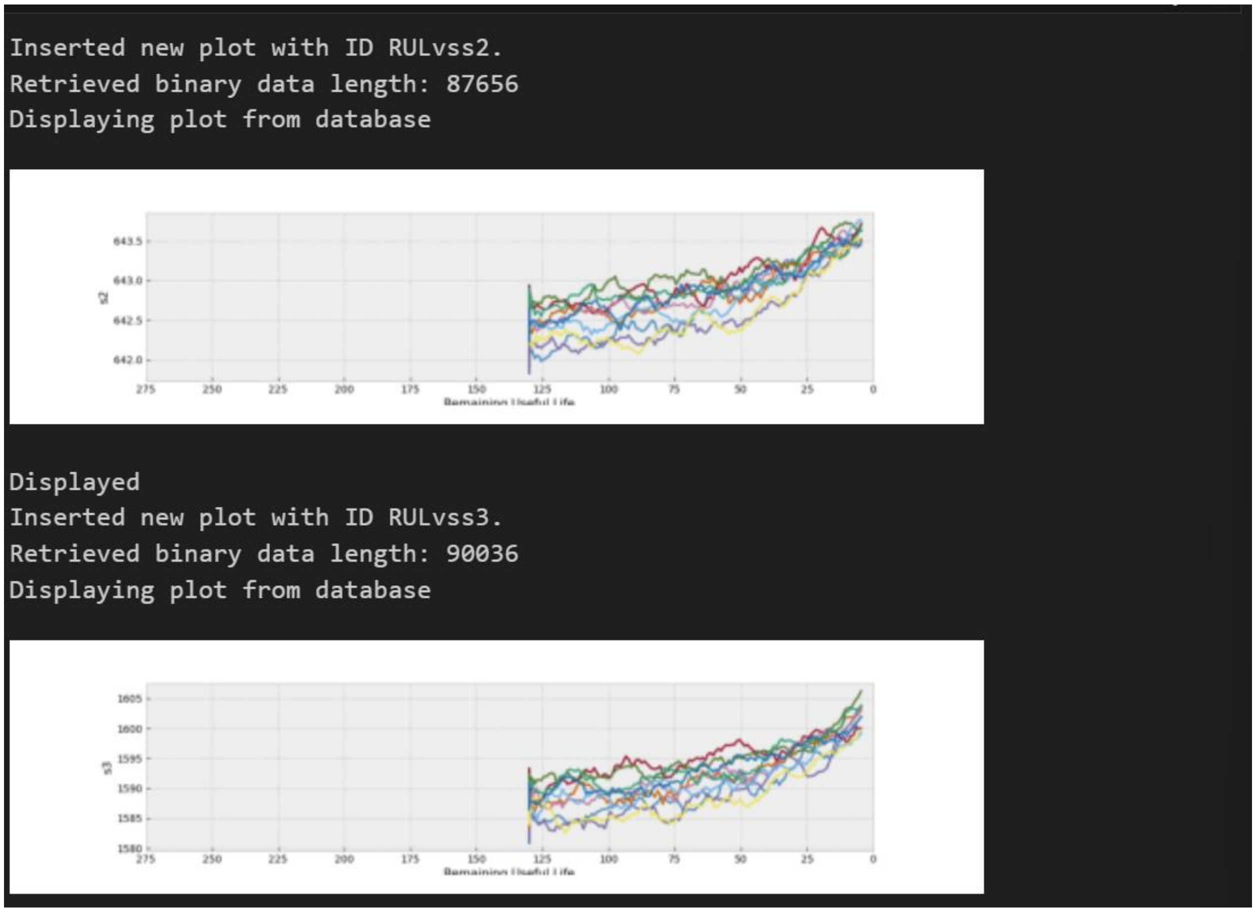


```
Text(0.5, 9.444444444444455, 'max time cycle')
```

Figure 7: Distribution of Maximum Time Cycles for Engines

Plotting Sensor Data vs. Remaining Useful Life (RUL) for Selected Engines

```
    plt.xlim(250, 0)  # reverse the x-axis so RUL counts down to zero
    plt.xticks(np.arange(0, 300, 25))
    plt.ylabel(signal_name)
    plt.xlabel('Remaining Useful Life')
    plot_id = f'RULvs{signal_name}'  # f-string to correctly insert signal_name into the plot_id
    insert_plot_to_db(plot_id, plt)
    retrive_from_pgres(plot_id)
    plt.show()
for i in range(1, 22):
    column_name = 's' + str(i)
    if column_name in df.columns:
        try:
            plot_signal(df,'s'+str(i))
        except:
            pass
```



```
Inserted new plot with ID RULvss2.
Retrieved binary data length: 87656
Displaying plot from database
```

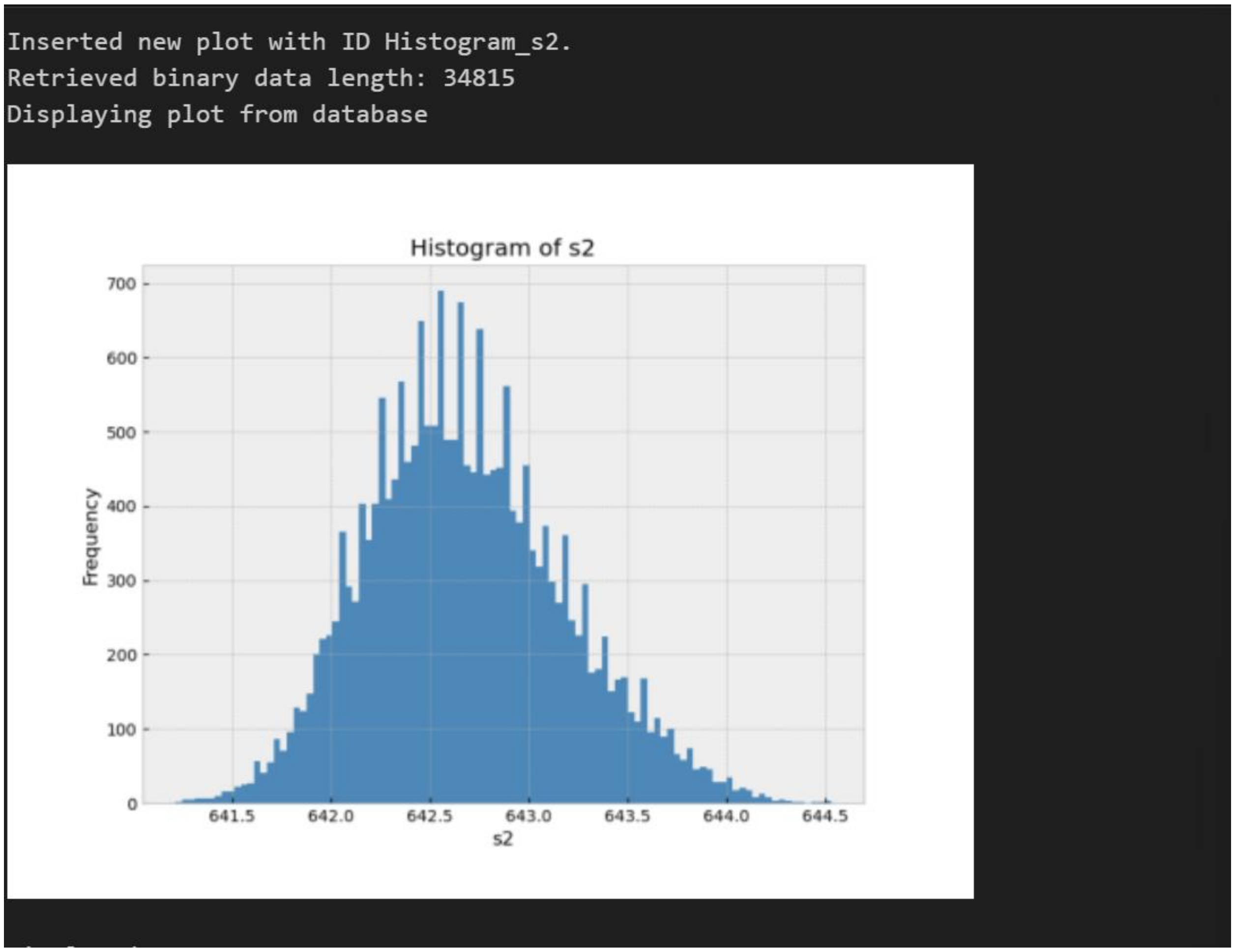
```
Displayed
Inserted new plot with ID RULvss3.
Retrieved binary data length: 90036
Displaying plot from database
```

Figure 8: Plotting Sensor Data vs. Remaining Useful Life (RUL) for Selected Engines

Creating Histograms for Sensor Value Distributions

```
# Create a histogram for the current sensor
plt.figure(figsize=(8, 6))  # Adjust the size of each individual plot
df[sensor].hist(bins=100)


# Set plot title and labels
plt.title(f'Histogram of {sensor}')
plt.xlabel(sensor)
plt.ylabel('Frequency')


# Create a unique plot ID for each sensor
plot_id = f'Histogram_{sensor}'
```

```
        # Insert the plot into PostgreSQL
        insert_plot_to_db(plot_id, plt)

        # Optionally clear the figure after inserting to avoid overlap with future plots
        plt.clf()

        # Retrieve the plot from the database and display it
        retrive_from_pgres(plot_id)
        plt.show()  # Ensure the plot is shown after retrieval

# Call the function with the dataframe `df` and sensor_names
plot_histogram_for_each_sensor(df, sensor_names)
```

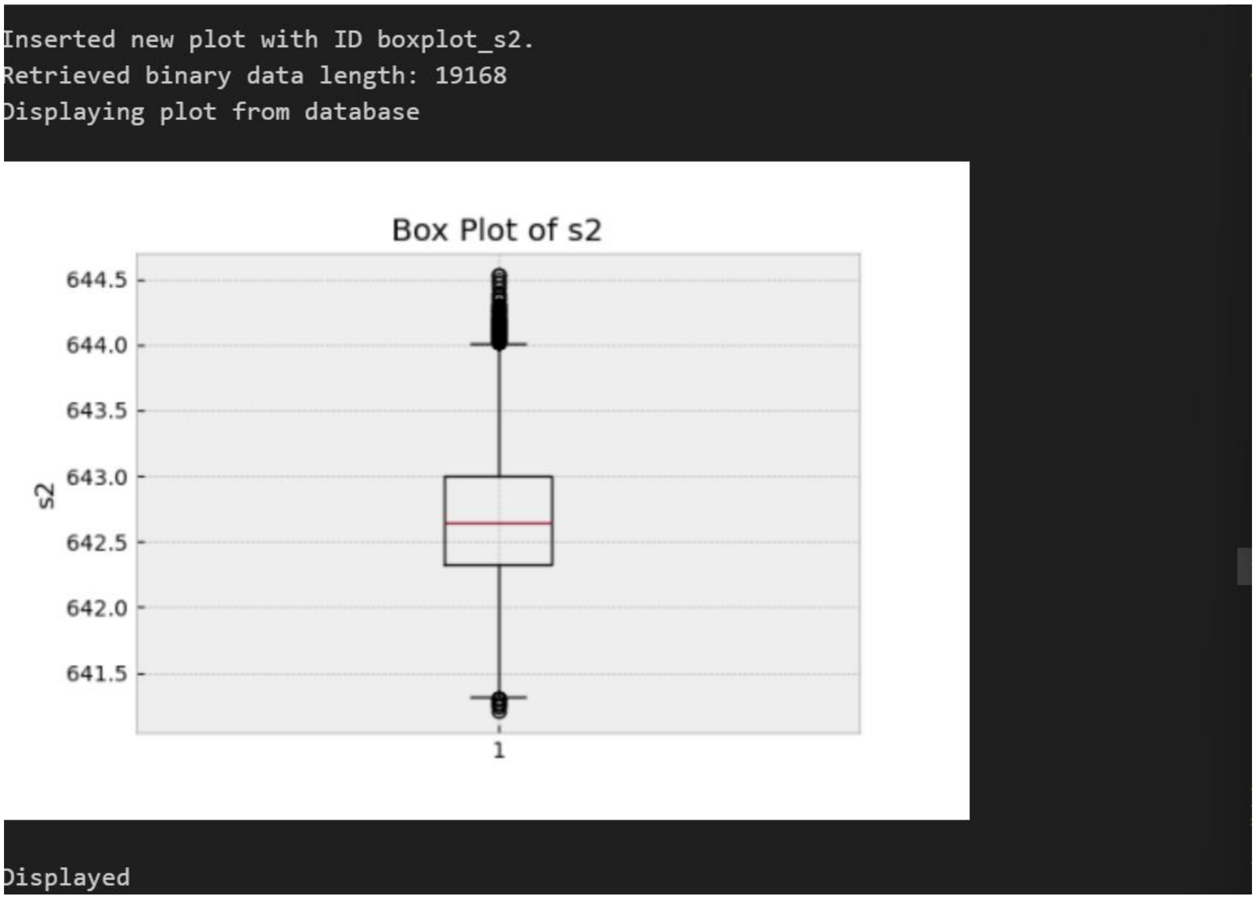


```
Inserted new plot with ID Histogram_s2.
Retrieved binary data length: 34815
Displaying plot from database
```

Figure 9: Histograms for Sensor Value Distributions

Generating and Storing Box Plots for Each Sensor

```
# Generate and insert box plots for each sensor
def plot_boxplot_for_each_sensor(df, sensor_names):
    for i, sensor in enumerate(sensor_names):
        # Create a new figure for each sensor
        plt.figure(figsize=(6, 4))

        # Create box plot for the current sensor
        plt.boxplot(df[sensor])

        # Set title and labels
        plt.title(f'Box Plot of {sensor}')
        plt.ylabel(sensor)

        # Create a unique plot ID for each box plot
        plot_id = f'boxplot_{sensor}'  # Unique name, where i+1 corresponds to the sensor number

        # Insert the plot into PostgreSQL
        insert_plot_to_db(plot_id, plt)

        # Optionally clear the figure after inserting to avoid overlap with future plots
        plt.clf()

        # Retrieve the plot from PostgreSQL and display it
        retrive_from_pgres(plot_id)
        plt.show()  # Ensure the plot is shown after retrieval

# Call the function to generate and store box plots
plot_boxplot_for_each_sensor(df, sensor_names)
```


```
Inserted new plot with ID boxplot_s2.
Retrieved binary data length: 19168
Displaying plot from database
```

```
Displayed
```

Figure 10; Box Plots for Each Sensor

#BEST FEATURE FOR XGBoost

rf.fit(X_train, y_train)

plt.barh(X_train.columns, rf.feature_importances_)

# insert_plot_to_db('FeatureImportance', plt)

# retrive_from_pgres('FeatureImportance')

### Creating new df for rf

Train_rf = X_train.copy()

Test_rf = X_test.copy()

Valid_rf = X_val.copy()

Train_rf.drop(columns=['s4','s7','s9','s11','s12'], inplace=True)

Test_rf.drop(columns=['s4','s7','s9','s11','s12'], inplace=True)

```
Valid_rf.drop(columns=['s4','s7','s9','s11','s12'], inplace=True)

Train_rf_s=scaler.fit_transform(Train_rf)

Test_rf_s=scaler.fit_transform(Test_rf)

Valid_rf_s=scaler.fit_transform(Valid_rf)

from sklearn.svm import SVR

xgb = xgboost.XGBRegressor(n_estimators=110, learning_rate=0.02, gamma=0,
subsample=0.8,colsample_bytree=0.5, max_depth=3)

xgb.fit(X_train_s,y_train)

y_hat_train = xgb.predict(X_train_s)

evaluate(y_train,y_hat_train, label='train')

y_hat_test = xgb.predict(X_test_s)

evaluate(y_test, y_hat_test, label='test')

y_hat_val = xgb.predict(X_val_s)

evaluate(y_val, y_hat_val, label='valid')
```

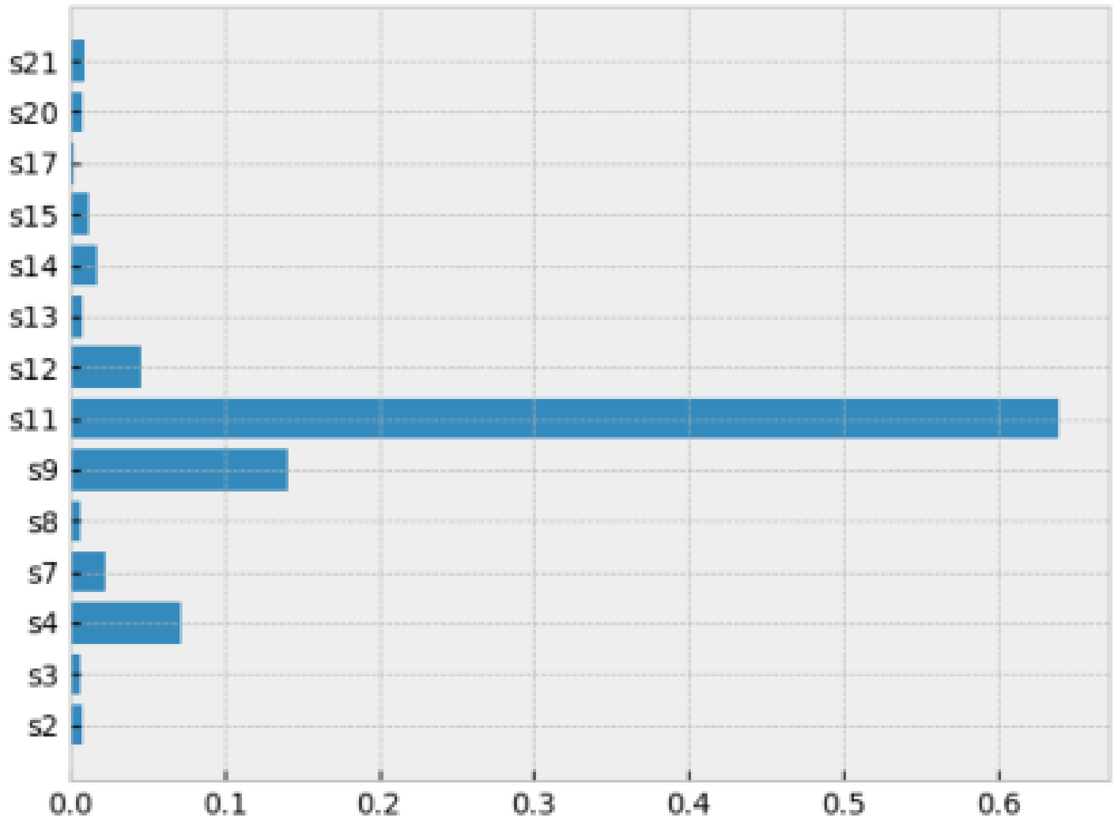


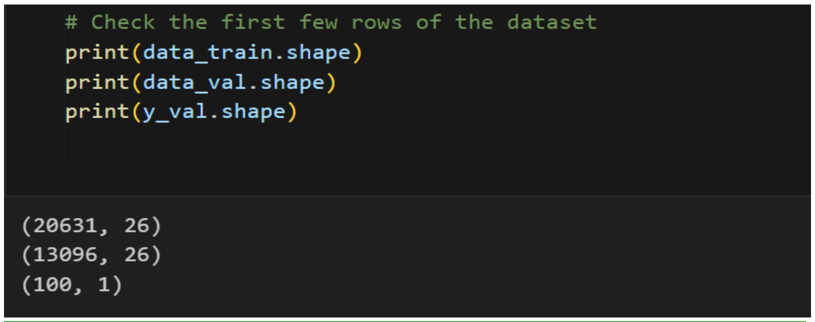
Figure 11:BEST FEATURE FOR XGBoost

Chapter 6: Testing

6.1Test cases for Important Modules

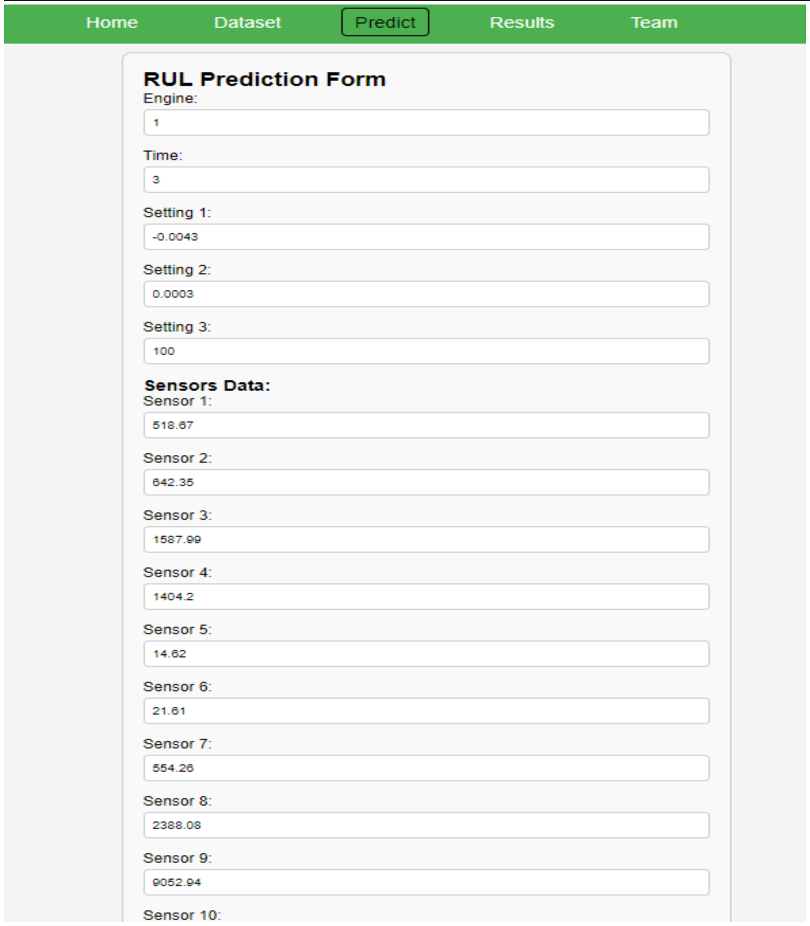
Verify that PostgreSQL is successfully connected, and that the dataset is loaded into the Jupiter Notebook environment. –

Test Case: Attempt to retrieve a first few data entry.



```
# Check the first few rows of the dataset
print(data_train.shape)
print(data_val.shape)
print(y_val.shape)
```

```
(20631, 26)
(13096, 26)
(100, 1)
```

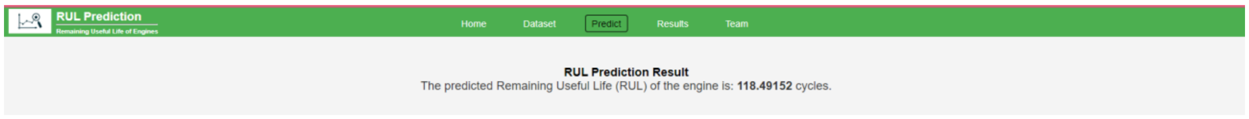


RUL Prediction Form

Engine:	1
Time:	3
Setting 1:	-0.0043
Setting 2:	0.0003
Setting 3:	100

Sensors Data:

Sensor 1:	518.67
Sensor 2:	642.35
Sensor 3:	1587.99
Sensor 4:	1404.2
Sensor 5:	14.62
Sensor 6:	21.61
Sensor 7:	554.26
Sensor 8:	2388.08
Sensor 9:	9052.94
Sensor 10:	



RUL Prediction Result
The predicted Remaining Useful Life (RUL) of the engine is: **118.49152** cycles.

6.2 model efficiency

Model Performance

i. Model Development and Evaluation

To forecast Remaining Useful Life (RUL) in aircraft engines, a range of machine learning models was rigorously evaluated, including Random Forest Regressor, XGBoost Regressor, Support Vector Regressor, Linear Regressor, and K-Nearest Neighbors. These models were meticulously trained and assessed using historical engine operational data, enabling the accurate differentiation between engines nearing failure and those operating optimally.

1. RandomForestRegressor:

Upon training and validating the RandomForestRegressor model, the following results were obtained:

- Training Set: RMSE: 15.80, R^2: 0.87
- Validation Set: RMSE: 23.24, R^2: 0.71
- Test Set: RMSE: 28.16, R^2: 0.54

These metrics indicate that while the model demonstrates strong performance on the training dataset, it exhibits a significant decline in predictive accuracy when applied to the testing dataset. This discrepancy suggests that the model is overfitting, capturing noise and specific patterns from the training data rather than generalizing effectively to unseen data.

The predicted vs actual RUL for RandomForestRegressor is as in figure 12.

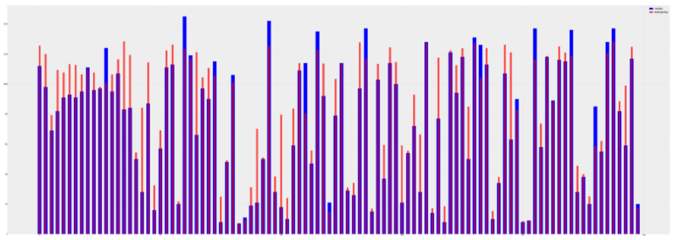


Figure-12: Actual RUL (blue) vs Predicted RUL (red) of RandomForestRegressor

2. Support Vector Regressor

The performance metrics for the Support Vector Regressor (SVR) model are delineated as follows:

- Train set RMSE:19.768, R2:0.795
- Validation set RMSE:23.987, R2:0.688
- test set RMSE:23.784, R2:0.672

These results elucidate that the SVR model exhibits commendable performance on the training dataset, the model is proficient in capturing a substantial portion of the variance present in the training data. However, the metrics for the validation and test datasets indicate a decline in performance.

The relatively stable RMSE values across the validation and test sets imply that the model is generalizing more effectively than the RandomForestRegressor, albeit with some limitations. The R² scores reflect a deficiency in the model's ability to account for a significant portion of the variance in the validation and test datasets, suggesting potential inadequacies in its predictive fidelity.

the Support Vector Regressor demonstrates an admirable capacity to fit the training data, its performance on unseen data underscores a need for enhancement in model generalization.

The predicted vs actual RUL for Support Vector Regressor is as in figure 13.

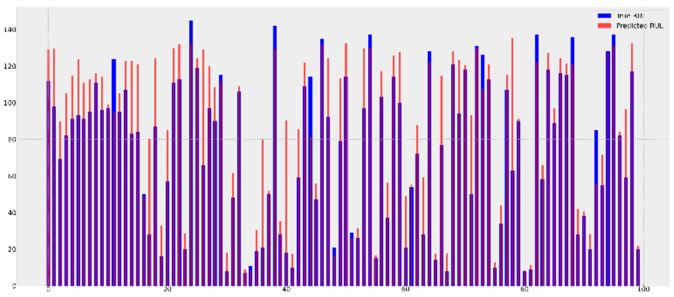


Figure-13: Actual RUL (blue) vs Predicted RUL(red) of SupportVectorRegressor

3. Linear Regressor

   - train set RMSE:22.628, R2:0.731
   - validation set RMSE:24.961, R2:0.663
   - test set RMSE:29.0749, R2:0.51

An analysis of these results reveals that the Linear Regressor demonstrates a moderate level of performance on the training dataset. These values indicate that the model captures a fair proportion of the variance in the training data. But the performance metrics for both the validation and test datasets indicate a noticeable decline.

The increase in RMSE and the corresponding decrease in R² for the validation and test sets suggest that the Linear Regressor is struggling to generalize effectively to unseen data. The R² score of 0.51 on the test set implies that less than half of the variance in the data is being explained, which raises concerns about the model's robustness in predictive tasks.

The Linear Regressor provides a baseline performance, its inability to maintain predictive accuracy across different datasets indicates a potential for underfitting.

The predicted vs actual RUL for Linear Regressor is as in figure 14.

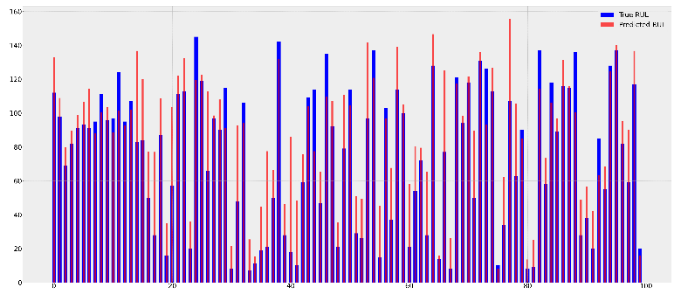


Figure-14: Actual RUL (blue) vs Predicted RUL(red) of LinearRegressor

4. XGBoost Regressor:

- train set RMSE:20.581, R2:0.778
- validation set RMSE:22.501, R2:0.726
- test set RMSE:23.801, R2:0.672

The results indicate a robust capability in predicting the Remaining Useful Life (RUL) of engines. The achieved training metrics suggest that the model effectively captures the underlying patterns in the training data, demonstrating a high degree of accuracy in estimating RUL. The relatively low RMSE indicates minimal deviation between the predicted and actual values, while the R² score signifies that approximately 78% of the variance in the training data is explained by the model. The validation performance illustrates the model's ability to generalize well to unseen data, maintaining a strong predictive capability. The validation RMSE, while slightly higher than that of the training set, remains within an acceptable range, indicating that the model is not overfitting but rather adjusting appropriately to the nuances of the validation data. The model's performance on the test set confirms the model's robustness, demonstrating its ability to predict RUL effectively even with completely new data. The R² score indicates that around **67%** of the variance in the test data can be explained by the model, showcasing its reliability in real-world applications.

The predicted vs actual RUL for XGboost Regressor is as in figure 15.

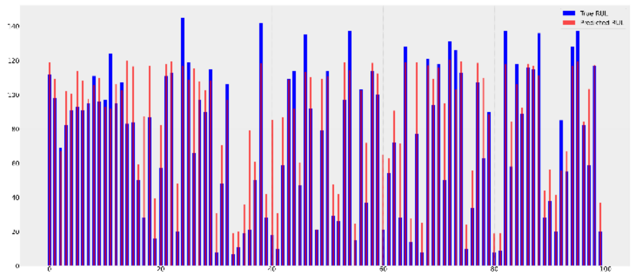


Figure-15: Actual RUL (blue) vs Predicted RUL (red) of XGBoostRegressor

5. KNN

- train set RMSE:29.169, R2:0.55
- validation set RMSE:30.435, R2:0.499
- test set RMSE:30.735, R2:0.453

The K-Nearest Neighbors (KNN) algorithm exhibited subpar performance across all evaluated datasets. The model showed only moderate accuracy and limited predictive power. On the validation set indicates a decline in performance on unseen data. This trend continued with the test set further demonstrating the model's inadequacy in generalizing beyond the training data. Overall, the KNN model's inability to provide reliable estimates for Remaining Useful Life (RUL) suggests that it may not be suitable for this predictive maintenance task.

The predicted vs actual RUL for KNN is as in figure 16.

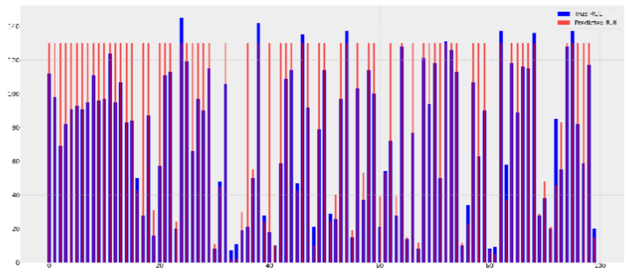


Figure-16: Actual RUL (blue) vs Predicted RUL(red) of KNN

6.3 Evaluation metrics

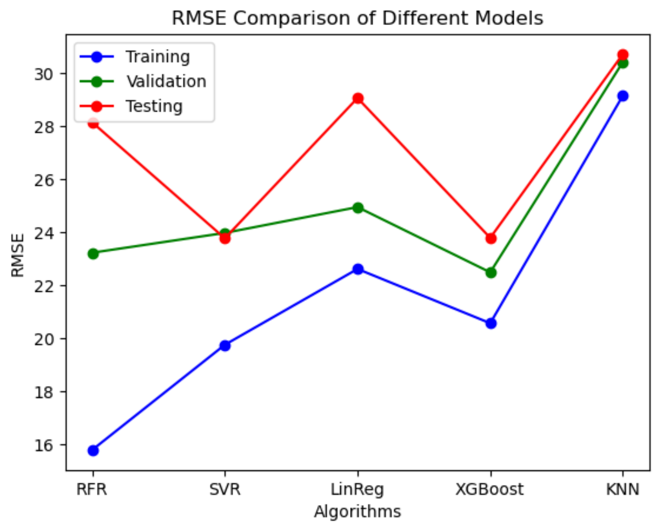


Figure-17: RMSE comparison for different models

Unlike other models evaluated in this analysis, XGBoost shows a consistent predictive capability across different datasets. The validation R² of 0.73 signifies that the model explains a substantial portion of the variance in RUL, reinforcing its utility in practical applications. Moreover, the test R² score of 0.67

suggests that XGBoost maintains robust predictive performance even when faced with unseen data, highlighting its proficiency in generalizing from the training examples.

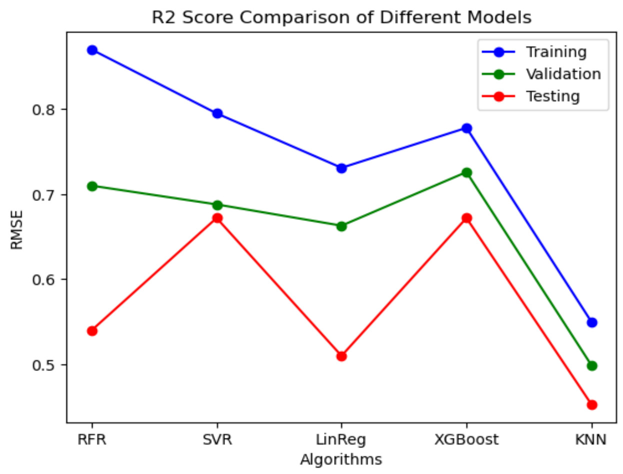
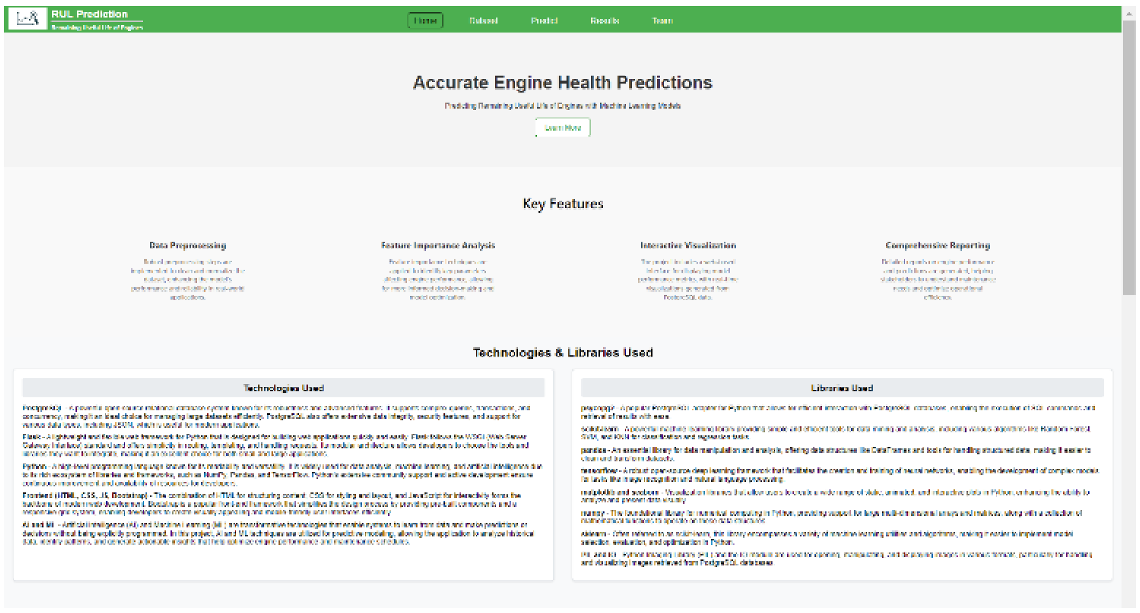


Figure-18: R^2 score comparison for different models

Chapter 7: Screenshots of Development Product

analytical results and model predictions. the images of the flask-based web page.



Accurate Engine Health Predictions

Predicting Remaining Useful Life of Engines with Machine Learning Models

Key Features

Data Preprocessing

Robust preprocessing steps are implemented to clean and normalize the dataset, enhancing the model's performance and reliability in real-world applications.

Feature Importance Analysis

Feature importance techniques are applied to identify key parameters affecting engine performance, allowing for more informed decision-making and model optimization.

Interactive Visualization

The project includes a web-based interface for displaying model performance metrics, with real-time visualizations generated from PostgreSQL data.

Comprehensive Reporting

Detailed reports on engine performance and predictions are generated, helping stakeholders to understand maintenance needs and optimize operational efficiency.

Technologies & Libraries Used

Technologies Used

PostgreSQL - A powerful open source relational database system known for its robustness and advanced features. It supports complex queries, transactions, and concurrency, making it an ideal choice for managing large datasets efficiently. PostgreSQL also offers extensive data integrity, security features, and support for various data types, including JSON, which is useful for modern applications.

Flask - A lightweight and flexible web framework for Python that is designed for building web applications quickly and easily. Flask follows the WSGI (Web Server Gateway Interface) standard and offers simplicity in routing, templating, and handling requests. Its modular architecture allows developers to choose the tools and libraries they want to integrate, making it an excellent choice for both small and large applications.

Python - A high-level programming language known for its readability and versatility. It is widely used for data analysis, machine learning, and artificial intelligence due to its rich ecosystem of libraries and frameworks, such as NumPy, Pandas, and TensorFlow. Python's extensive community support and active development ensure continuous improvement and availability of resources for developers.

Frontend (HTML, CSS, JS, Bootstrap) - The combination of HTML for structuring content, CSS for styling and layout, and JavaScript for interactivity forms the backbone of modern web development. Bootstrap is a popular front-end framework that simplifies the design process by providing pre-built components and a responsive grid system, enabling developers to create visually appealing and mobile-friendly user interfaces efficiently.

AI and ML - Artificial Intelligence (AI) and Machine Learning (ML) are transformative technologies that enable systems to learn from data and make predictions or decisions without being explicitly programmed. In this project, AI and ML techniques are utilized for predictive modeling, allowing the application to analyze historical data, identify patterns, and generate actionable insights that help optimize engine performance and maintenance schedules.

Libraries Used

psycopg2 - A popular PostgreSQL adapter for Python that allows for efficient interaction with PostgreSQL databases, enabling the execution of SQL commands and retrieval of results with ease.

scikit-learn - A powerful machine learning library providing simple and efficient tools for data mining and analysis, including various algorithms like Random Forest, SVM, and KNN for classification and regression tasks.

pandas - An essential library for data manipulation and analysis, offering data structures like DataFrames and tools for handling structured data, making it easier to clean and transform datasets.

tensorflow - A robust open-source deep learning framework that facilitates the creation and training of neural networks, enabling the development of complex models for tasks like image recognition and natural language processing.

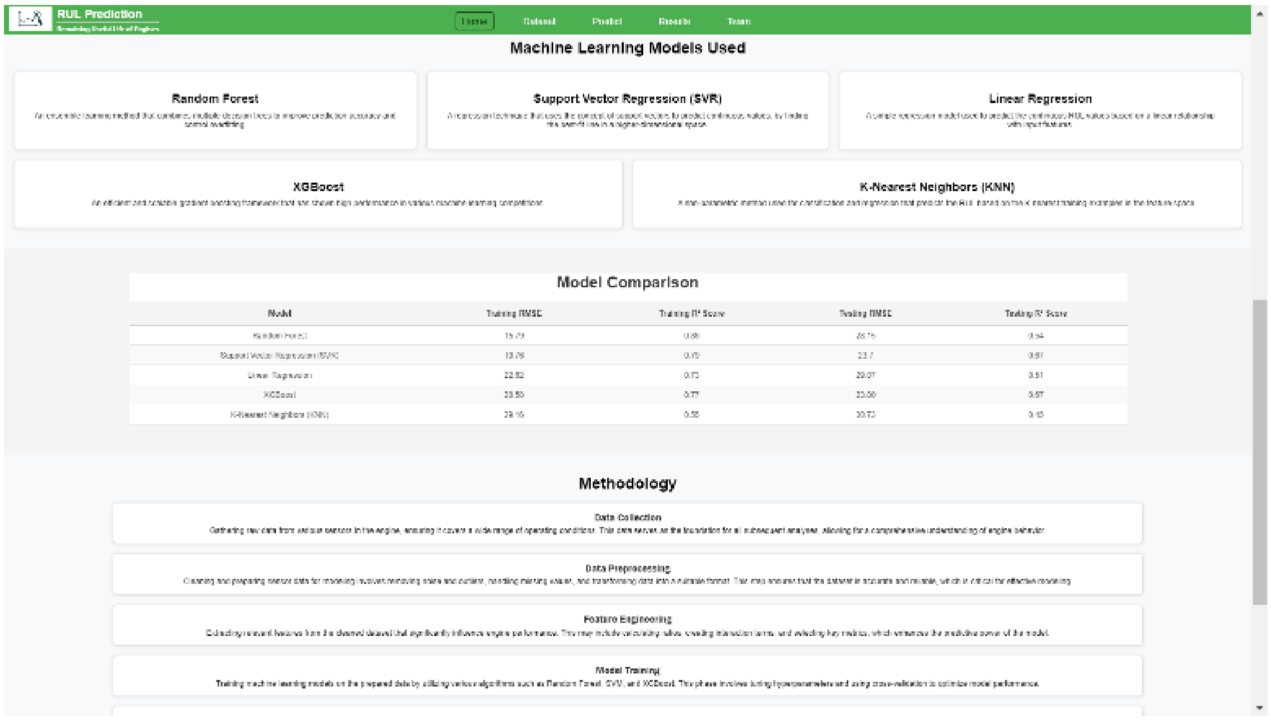
matplotlib and seaborn - Visualization libraries that allow users to create a wide range of static, animated, and interactive plots in Python, enhancing the ability to analyze and present data visually.

numpy - The foundational library for numerical computing in Python, providing support for large multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these data structures.

sklearn - Often referred to as scikit-learn, this library encompasses a variety of machine learning utilities and algorithms, making it easier to implement model selection, evaluation, and optimization in Python.

PIL and io - Python Imaging Library (PIL) and the io module are used for opening, manipulating, and displaying images in various formats, particularly for handling and visualizing images retrieved from PostgreSQL databases.

Figure 19.1 : Home page containing key features and technologies , libraries used



Machine Learning Models Used

Random Forest
An ensemble learning method that combines multiple decision trees to improve prediction accuracy and control overfitting.

Support Vector Regression (SVR)
A regression technique that uses the concept of support vectors to predict continuous values, by finding the best-fit line in a higher-dimensional space.

Linear Regression
A simple regression model used to predict the continuous RUL values based on a linear relationship with input features.

XGBoost
An efficient and scalable gradient boosting framework that has shown high performance in various machine learning competitions.

K-Nearest Neighbors (KNN)
A non-parametric method used for classification and regression that predicts the RUL based on the K nearest training examples in the feature space.

Model Comparison

Model	Training RMSE	Training R² Score	Testing RMSE	Testing R² Score
Random Forest	15.79	0.86	28.75	0.54
Support Vector Regression (SVR)	18.76	0.79	23.7	0.67
Linear Regression	22.52	0.73	29.07	0.51
XGBoost	23.50	0.77	23.00	0.67
K-Nearest Neighbors (KNN)	28.16	0.58	30.73	0.45

Methodology

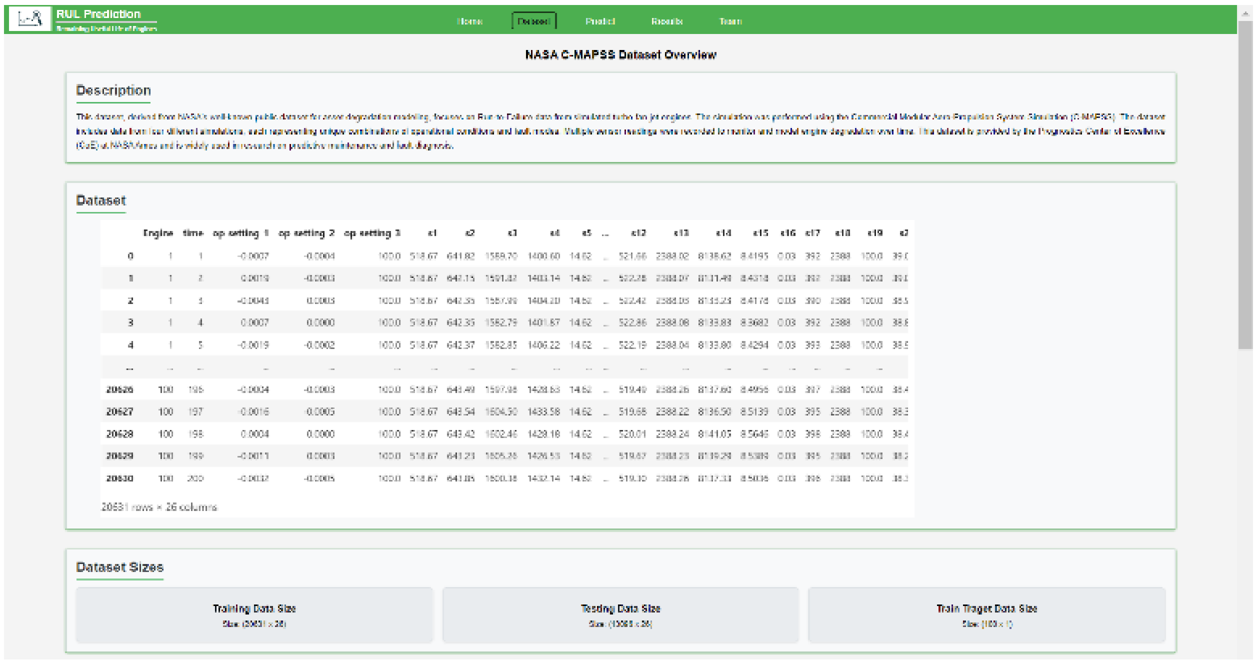
Data Collection
Gathering raw data from various sensors in the engine, ensuring it covers a wide range of operating conditions. This data serves as the foundation for all subsequent analyses, allowing for a comprehensive understanding of engine behavior.

Data Preprocessing
Cleaning and preparing sensor data for modeling involves removing noise and outliers, handling missing values, and transforming data into a suitable format. This step ensures that the dataset is accurate and reliable, which is critical for effective modeling.

Feature Engineering
Extracting relevant features from the cleaned dataset that significantly influence engine performance. This may include calculating ratios, creating interaction terms, and selecting key metrics, which enhances the predictive power of the model.

Model Training
Training machine learning models on the prepared data by utilizing various algorithms such as Random Forest, SVR, and XGBoost. This phase involves tuning hyperparameters and using cross-validation to optimize model performance.

Figure 19.2 : Home page containing machine learning models used



NASA C-MAPSS Dataset Overview

Description

This dataset, derived from NASA's well-known public dataset for asset degradation modeling, focuses on Run-to-Failure data from simulated turbo fan jet engines. The simulation was performed using the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS). The dataset includes data from four different simulations, each representing unique combinations of operational conditions and fault modes. Multiple sensor readings were recorded to monitor and model engine degradation over time. This dataset is provided by the Prognostics Center of Excellence (CoE) at NASA Ames and is widely used in research on predictive maintenance and fault diagnosis.

Dataset

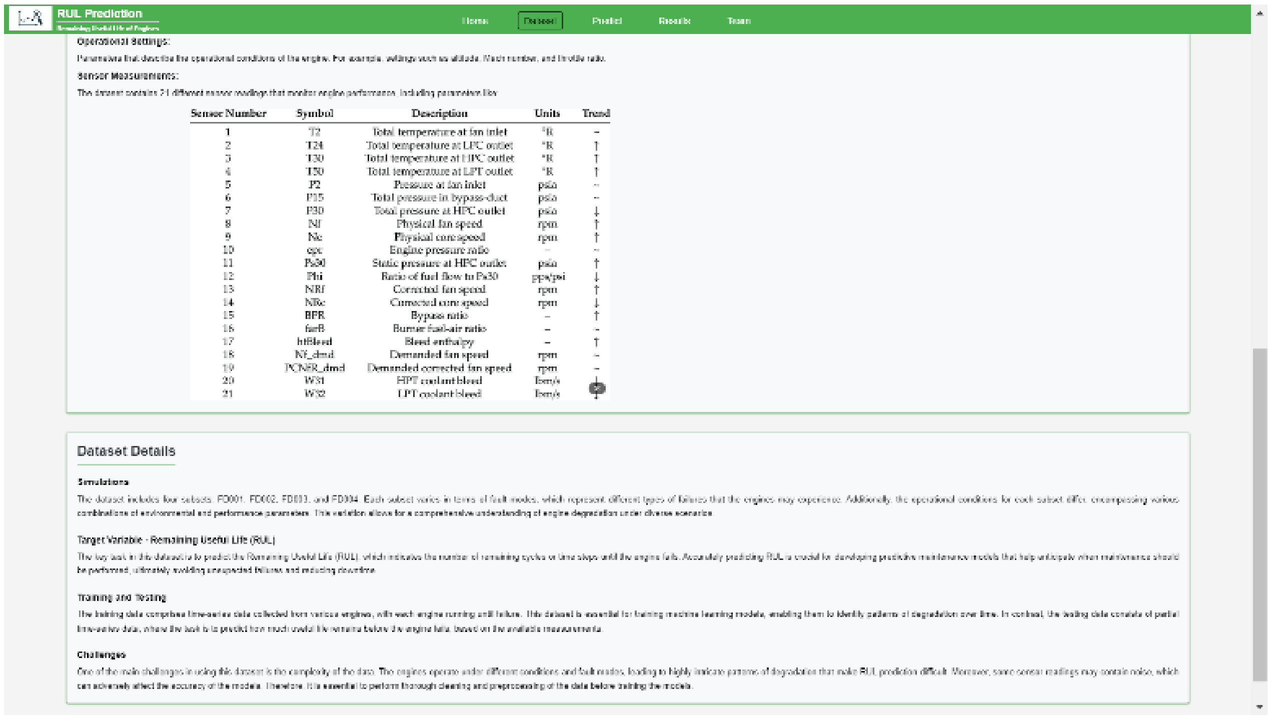
	Engine	time	op setting 1	op setting 2	op setting 3	s1	s2	s3	s4	s5	...	s12	s13	s14	s15	s16	s17	s18	s19	s2
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	521.66	2388.02	8138.62	8.4195	0.03	392	2388	100.0	39.0
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	522.28	2388.07	8131.49	8.4318	0.03	392	2388	100.0	39.0
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	522.42	2388.03	8133.23	8.4178	0.03	390	2388	100.0	38.9
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	522.86	2388.08	8133.83	8.3682	0.03	392	2388	100.0	38.8
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	522.19	2388.04	8133.80	8.4294	0.03	393	2388	100.0	38.9
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
20626	100	196	-0.0004	-0.0003	100.0	518.67	643.49	1597.98	1428.63	14.62	...	519.49	2388.26	8137.60	8.4956	0.03	397	2388	100.0	38.4
20627	100	197	-0.0016	-0.0005	100.0	518.67	643.54	1604.50	1433.58	14.62	...	519.68	2388.22	8136.50	8.5139	0.03	395	2388	100.0	38.3
20628	100	198	0.0004	0.0000	100.0	518.67	643.42	1602.46	1428.18	14.62	...	520.01	2388.24	8141.05	8.5646	0.03	398	2388	100.0	38.4
20629	100	199	-0.0011	0.0003	100.0	518.67	643.23	1605.26	1426.53	14.62	...	519.67	2388.23	8139.29	8.5389	0.03	395	2388	100.0	38.2
20630	100	200	-0.0032	-0.0005	100.0	518.67	643.85	1600.38	1432.14	14.62	...	519.30	2388.26	8137.33	8.5036	0.03	396	2388	100.0	38.3

20631 rows × 26 columns

Dataset Sizes

Training Data Size	Testing Data Size	Train Traget Data Size
Size: (20631 x 26)	Size: (13096 x 26)	Size: (100 x 1)

Figure 19.3 : dataset page



Operational Settings:

Parameters that describe the operational conditions of the engine. For example, settings such as altitude, Mach number, and throttle ratio.

Sensor Measurements:

The dataset contains 21 different sensor readings that monitor engine performance, including parameters like:

Sensor Number	Symbol	Description	Units	Trend
1	T2	Total temperature at fan inlet	°R	~
2	T24	Total temperature at LPC outlet	°R	↑
3	T30	Total temperature at HPC outlet	°R	↑
4	T50	Total temperature at LPT outlet	°R	↑
5	P2	Pressure at fan inlet	psia	~
6	P15	Total pressure in bypass-duct	psia	~
7	P30	Total pressure at HPC outlet	psia	↓
8	Nf	Physical fan speed	rpm	↑
9	Nc	Physical core speed	rpm	↑
10	epr	Engine pressure ratio	–	~
11	Ps30	Static pressure at HPC outlet	psia	↑
12	Phi	Ratio of fuel flow to Ps30	pps/psi	↓
13	NRf	Corrected fan speed	rpm	↑
14	NRc	Corrected core speed	rpm	↓
15	BPR	Bypass ratio	–	↑
16	farB	Burner fuel-air ratio	–	~
17	htBleed	Bleed enthalpy	–	↑
18	Nf_dmd	Demanded fan speed	rpm	~
19	PCNfR_dmd	Demanded corrected fan speed	rpm	~
20	W31	HPT coolant bleed	lbm/s	↓
21	W32	LPT coolant bleed	lbm/s	↓

Dataset Details

Simulations

The dataset includes four subsets: FD001, FD002, FD003, and FD004. Each subset varies in terms of fault modes, which represent different types of failures that the engines may experience. Additionally, the operational conditions for each subset differ, encompassing various combinations of environmental and performance parameters. This variation allows for a comprehensive understanding of engine degradation under diverse scenarios.

Target Variable - Remaining Useful Life (RUL)

The key task in this dataset is to predict the Remaining Useful Life (RUL), which indicates the number of remaining cycles or time steps until the engine fails. Accurately predicting RUL is crucial for developing predictive maintenance models that help anticipate when maintenance should be performed, ultimately avoiding unexpected failures and reducing downtime.

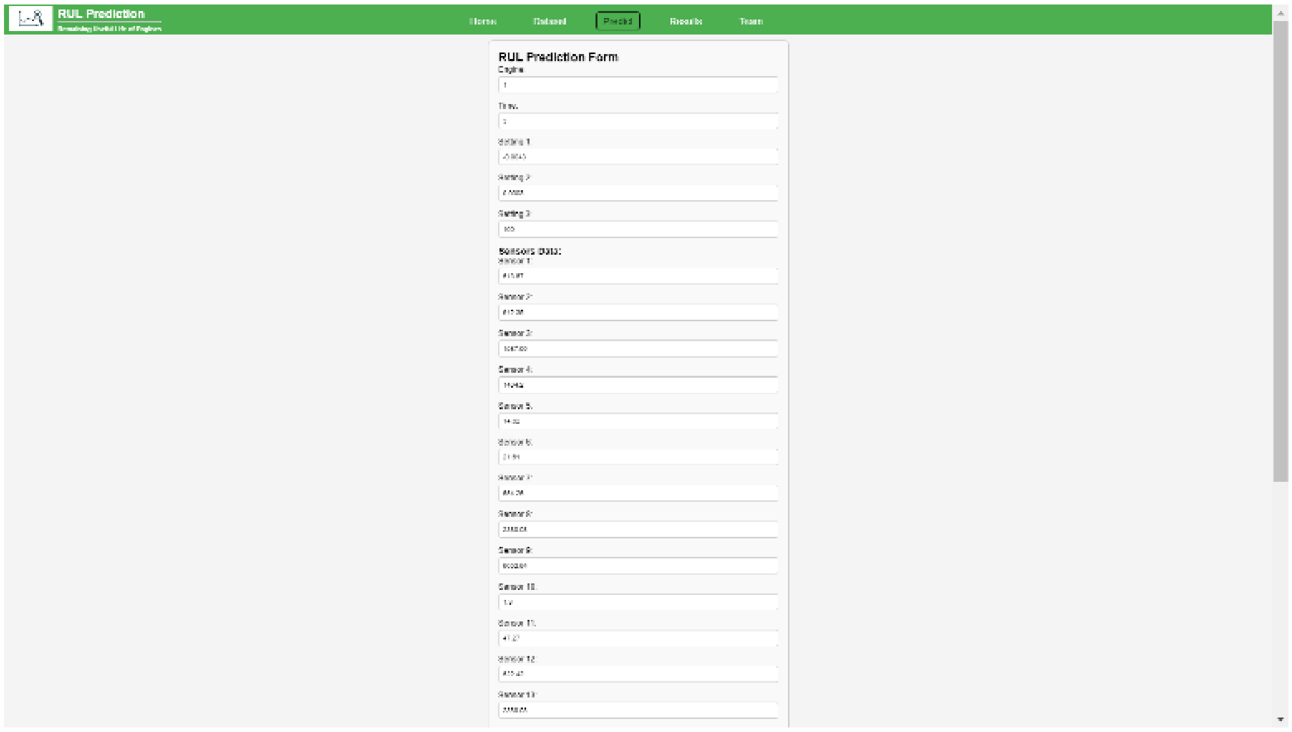
Training and Testing

The training data comprises time-series data collected from various engines, with each engine running until failure. This dataset is essential for training machine learning models, enabling them to identify patterns of degradation over time. In contrast, the testing data consists of partial time-series data, where the task is to predict how much useful life remains before the engine fails, based on the available measurements.

Challenges

One of the main challenges in using this dataset is the complexity of the data. The engines operate under different conditions and fault modes, leading to highly intricate patterns of degradation that make RUL prediction difficult. Moreover, some sensor readings may contain noise, which can adversely affect the accuracy of the models. Therefore, it is essential to perform thorough cleaning and preprocessing of the data before training the models.

Figure 19.3 : dataset page with dataset details



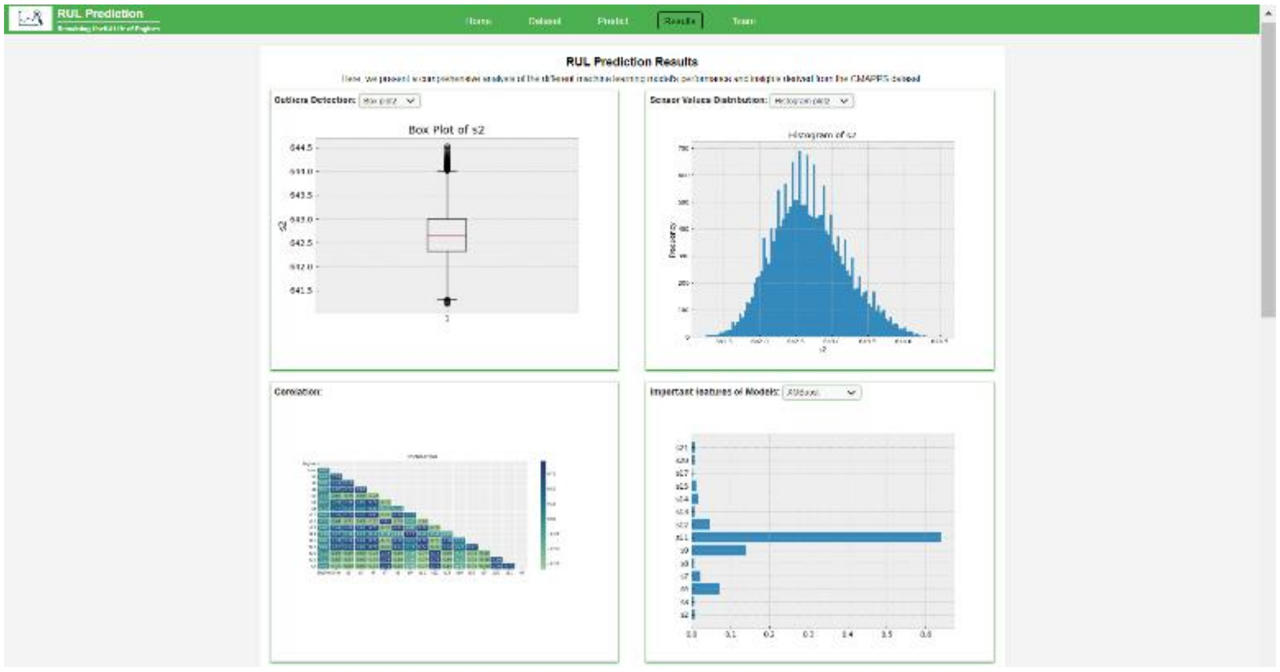
RUL Prediction Form

Engine	1
Time	[illegible]
Setting 1	[illegible]
Setting 2	[illegible]
Setting 3	100

Sensors Data:

Sensor 1:	[illegible]
Sensor 2:	[illegible]
Sensor 3:	[illegible]
Sensor 4:	[illegible]
Sensor 5:	[illegible]
Sensor 6:	[illegible]
Sensor 7:	[illegible]
Sensor 8:	[illegible]
Sensor 9:	[illegible]
Sensor 10:	[illegible]
Sensor 11:	[illegible]
Sensor 12:	[illegible]
Sensor 13:	[illegible]

Figure 19.5 : Result page



RUL Prediction Results

Here, we present a comprehensive analysis of the different machine learning models' performance and insights derived from the CMAPSS dataset

Figure 19.6 : Visualizations of result

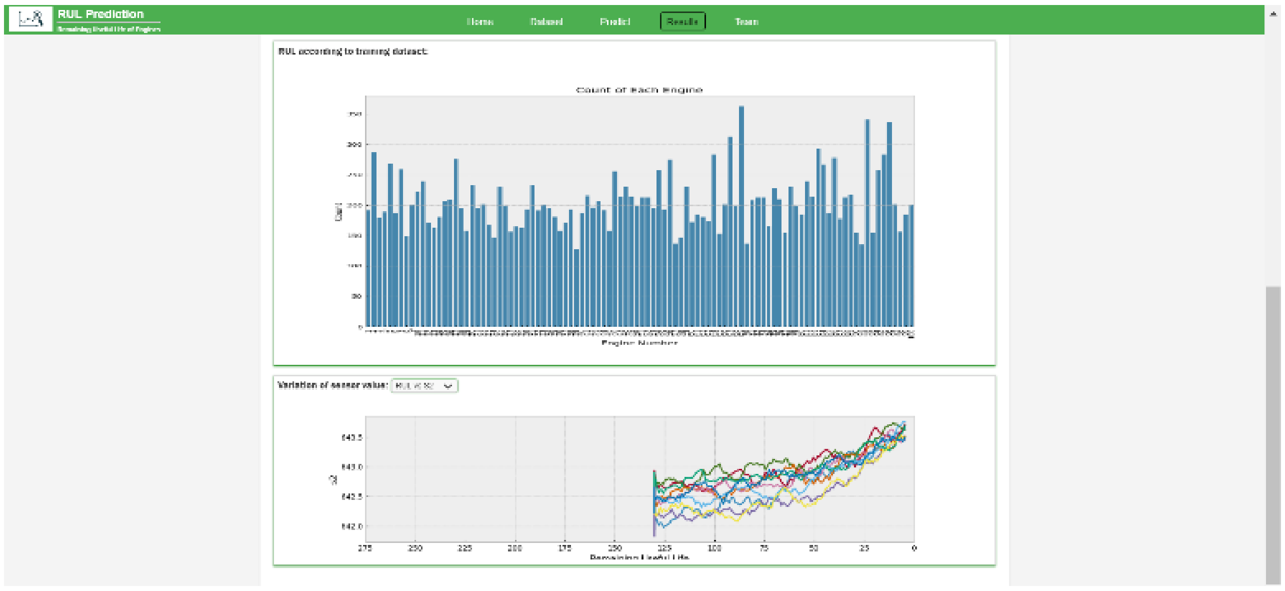


Figure 19.7 : Visualizations of result

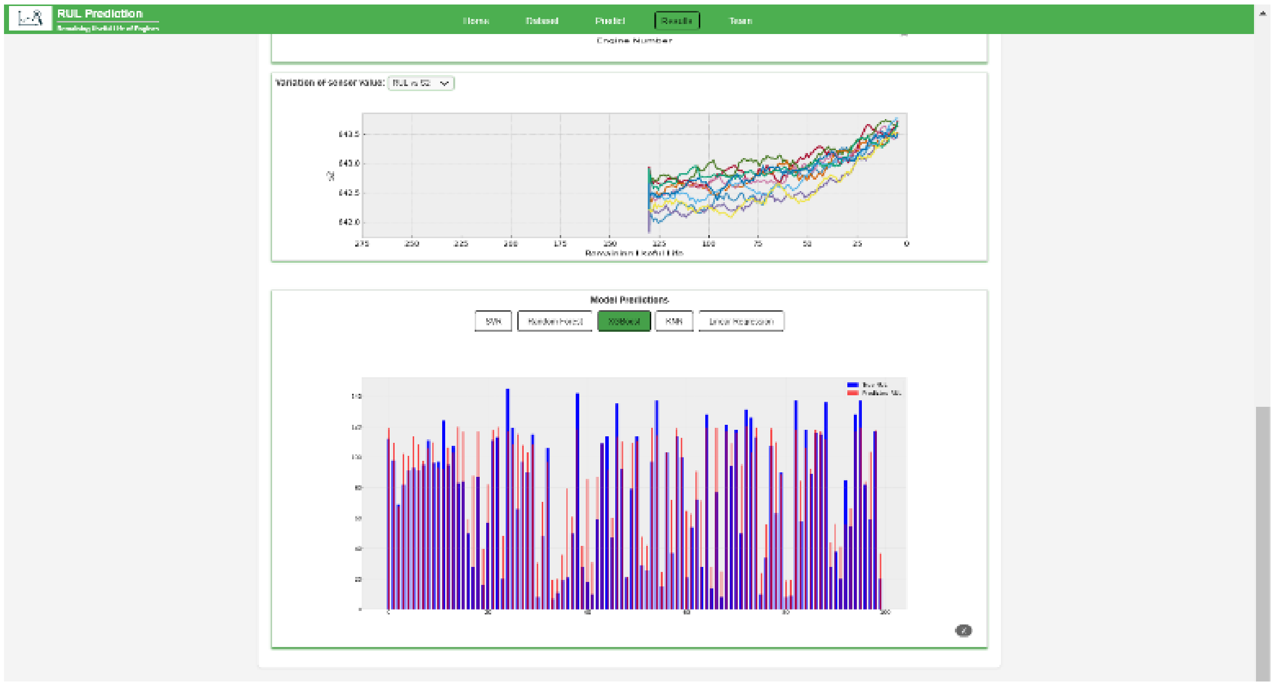
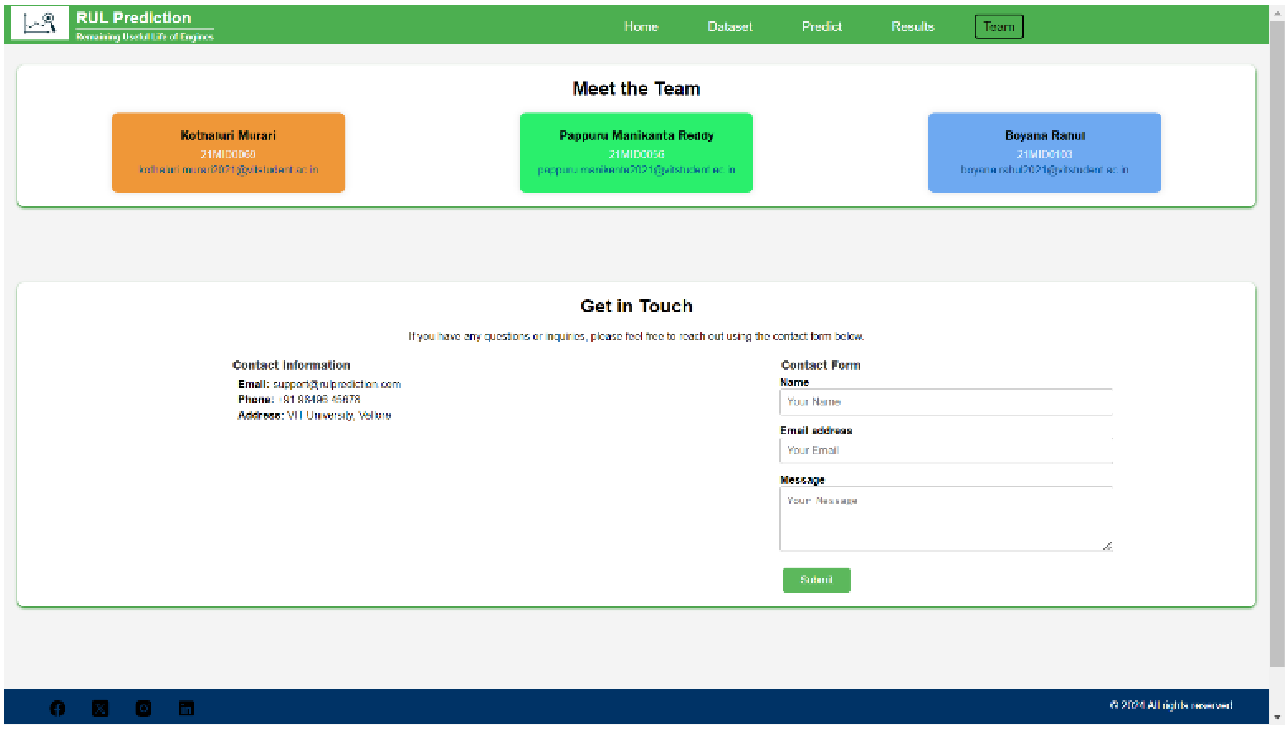


Figure 19.8: Visualizations of result



Meet the Team

Kothaluri Murari
21MID0069
kothaluri.murari2021@vitstudent.ac.in

Pappuru Manikanta Reddy
21MID0056
pappuru.manikanta2021@vitstudent.ac.in

Boyana Rahul
21MID0103
boyana.rahul2021@vitstudent.ac.in

Get in Touch

If you have any questions or inquiries, please feel free to reach out using the contact form below.

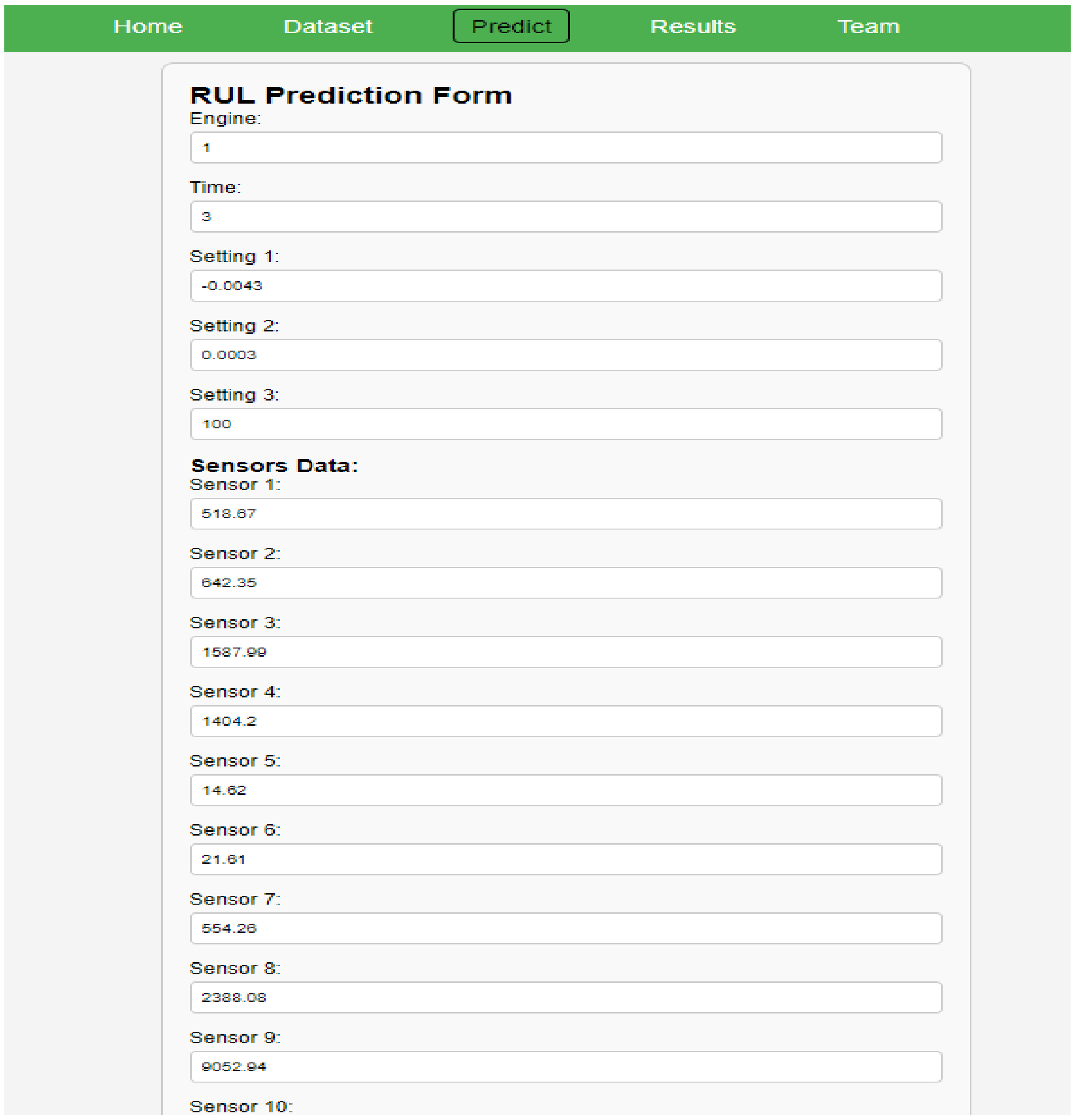
Contact Information
Email: support@rulprediction.com
Phone: +91 98496 45678
Address: VIT University, Vellore

Contact Form
Name
Your Name
Email address
Your Email
Message
Your Message
Submit

© 2024 All rights reserved

Figure 19.9 : Team and contact information page

Chapter 8: Outcome



RUL Prediction Form

Engine: 1

Time: 3

Setting 1: -0.0043

Setting 2: 0.0003

Setting 3: 100

Sensors Data:

Sensor 1: 518.67

Sensor 2: 642.35

Sensor 3: 1587.99

Sensor 4: 1404.2

Sensor 5: 14.62

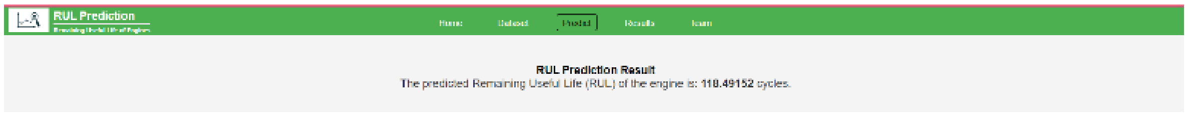
Sensor 6: 21.61

Sensor 7: 554.26

Sensor 8: 2388.08

Sensor 9: 9052.94

Sensor 10:



RUL Prediction Result

The predicted Remaining Useful Life (RUL) of the engine is: **118.49152** cycles.

Figure-20 Predicting RUL from User-Entered Data via Pickle File

.

CONCLUSION

In conclusion, the project successfully developed a robust predictive maintenance system for estimating the Remaining Useful Life (RUL) of aircraft engines, utilizing machine learning models and integrating multiple technologies to ensure efficiency and accuracy. By leveraging the NASA CMAPSS dataset, extensive preprocessing techniques, and various regression models, the system was able to predict RUL with a high degree of accuracy. Among the models tested, XGBoost demonstrated the most reliable performance, with a low RMSE and a high R^2 score, indicating its ability to make precise predictions without overfitting or underfitting. The integration of feature importance through Explainable AI (XAI) also provided insights into the influence of sensor data on RUL predictions.

Particularly XGBoost, which demonstrated impressive performance. The model's RMSE and R^2 scores were comparable to those typically achieved by deep learning models such as LSTM, highlighting its effectiveness in providing accurate predictions without the added complexity of deep learning architectures. XGBoost consistently outperformed other traditional models, offering an optimal balance of accuracy and efficiency while avoiding issues of overfitting or underfitting.

A key strength of the system lies in its comprehensive architecture, which combines PostgreSQL for data management, machine learning models for analysis, and Flask for real-time web deployment. The entire data pipeline, from data storage in PostgreSQL to model execution and result visualization on the web interface, worked seamlessly, ensuring that users can interact with and analyse engine data dynamically. The incorporation of manual data input through the web form further demonstrates the system's flexibility and usability in practical scenarios, proving its readiness for real-world application. This project highlights the potential of integrating machine learning, web technologies, and database systems for predictive maintenance. The system's modular design ensures scalability and adaptability for future enhancements, making it a valuable tool in the field of predictive analytics for machinery. Overall, the project not only achieves its goals of accurate RUL prediction but also sets a foundation for further advancements in the field, offering a significant contribution to maintenance optimization and operational efficiency in industrial applications.

This predictive maintenance system is highly valuable in the aircraft industry, where timely and accurate predictions of engine failure are crucial for safety and operational efficiency. By accurately estimating the Remaining Useful Life (RUL) of engines, airlines can optimize maintenance schedules, reduce unexpected downtime, and avoid costly repairs. This proactive approach minimizes the risk of in-flight engine failures, ensuring safer and more reliable operations. Moreover, by leveraging sensor data and machine learning, the system can continuously monitor engine performance in real time, allowing for dynamic adjustments and more informed decision-making. This technological advancement aligns with the aviation industry's increasing reliance on data-driven solutions to enhance maintenance strategies and overall aircraft lifecycle management.

REFERENCES

[1] R. Khelif, B. Chebel-Morello, S. Malinowski, E. Laajili, F. Fnaiech and N. Zerhouni, "Direct Remaining Useful Life Estimation Based on Support Vector Regression," in IEEE Transactions on Industrial Electronics, vol. 64, no. 3, pp. 2276-2285, March 2017, doi: 10.1109/TIE.2016.2623260.

[2] Ma, J.; Su, H.; Zhao, W.-L.; Liu, B. Predicting the Remaining Useful Life of an Aircraft Engine Using a Stacked Sparse Autoencoder with Multilayer Self-Learning. *Complexity* **2018**, *2018*, 3813029.

[3] Zhao, R.; Yan, R.; Chen, Z.; Mao, K.; Wang, P.; Gao, R.X. Deep learning and its applications to machine health monitoring. Mech. Syst. Signal Process. 2019, 115, 213–237.

[4] Wang, Y.; Zhao, Y. Multi-Scale Remaining Useful Life Prediction Using Long Short-Term Memory. *Sustainability* **2022**, *14*, 15667. https://doi.org/10.3390/su142315667.

[5] Ellefsen, A.L.; Bjørlykhaug, E.; Æsøy, V.; Ushakov, S.; Zhang, H. Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture. Reliab. Eng. Syst. Saf. 2018, 183, 240–251.

[6] Ayodeji, A.; Wang, Z.; Wang, W.; Qin, W.; Yang, C.; Xu, S.; Liu, X. Causal augmented ConvNet: A temporal memory dilated convolution model for long-sequence time series prediction. ISA Trans. 2021, 123, 200–217.

[7] Sharanya, S., Venkataraman, Revathi and Murali, G.. "Predicting remaining useful life of turbofan engines using degradation signal based echo state network" International Journal of Turbo & Jet-Engines, vol. 40, no. s1, 2023, pp. s181-s194. https://doi.org/10.1515/tjj-2022-0007.

[8] Zhang, Y.; Xin, Y.; Liu, Z.-W.; Chi, M.; Ma, G. Health status assessment and remaining useful life prediction of aero-engine based on BiGRU and MMoE. Reliab. Eng. Syst. Saf. 2022, 220, 108263. https://doi.org/10.1016/j.ress.2021.108263

[9] A. Saxena, K. Goebel, D. Simon and N. Eklund, "Damage propagation modeling for aircraft engine run-to-failure simulation," *2008 International Conference on Prognostics and Health Management*, Denver, CO, USA, 2008, pp. 1-9, doi: 10.1109/PHM.2008.4711414.

[10] J. Xu and L. Xu, "Health management based on fusion prognostics for avionics systems," in *Journal of Systems Engineering and Electronics*, vol. 22, no. 3, pp. 428-436, June 2011, doi: 10.3969/j.issn.1004-4132.2011.03.010.

[11] Wang, G.; Lyu, Z.; Li, X. An Optimized Random Forest Regression Model for Li-Ion Battery Prognostics and Health Management. Batteries 2023, 9, 332. https://doi.org/10.3390/batteries9060332

[12] V. Mathew, T. Toby, V. Singh, B. M. Rao and M. G. Kumar, "Prediction of Remaining Useful Lifetime (RUL) of turbofan engine using machine learning," *2017 IEEE International Conference on Circuits and Systems (ICCS)*, Thiruvananthapuram, India, 2017, pp. 306-311, doi: 10.1109/ICCS1.2017.8326010.

[13] Deng, S., Zhou, J. Prediction of Remaining Useful Life of Aero-engines Based on CNN-LSTM-Attention. *Int J Comput Intell Syst* **17**, 232 (2024). https://doi.org/10.1007/s44196-024-00639-w

[14]Ordóñez, C.; Lasheras, F.S.; Roca-Pardiñas, J.; Juez, F.J.D.C. A hybrid ARIMA–SVM model for the study of the remaining useful life of aircraft engines. J. Comput. Appl. Math. 2018, 346, 184–191. https://doi.org/10.1016/j.cam.2018.07.008.

[15]Mutunga, Jackline Mwende et al. “Health-Index Based Prognostics for a Turbofan Engine using Ensemble of Machine Learning Algorithms.” (2019).

[16]Kang Z, Catal C, Tekinerdogan B. Remaining Useful Life (RUL) Prediction of Equipment in Production Lines Using Artificial Neural Networks. *Sensors (Basel)*. 2021;21(3):932. Published 2021 Jan 30. doi:10.3390/s21030932.

[17]Youness G, Aalah A. An Explainable Artificial Intelligence Approach for Remaining Useful Life Prediction. Aerospace. 2023; 10(5):474. https://doi.org/10.3390/aerospace10050474.

[18]Protopapadakis, G, Apostolidis, A, & Kalfas, AI. "Explainable and Interpretable AI-Assisted Remaining Useful Life Estimation for Aeroengines." Proceedings of the ASME Turbo Expo 2022: Turbomachinery Technical Conference and Exposition. Volume 2: Coal, Biomass, Hydrogen, and Alternative Fuels; Controls, Diagnostics, and Instrumentation; Steam Turbine. Rotterdam, Netherlands. June 13–17, 2022. V002T05A002. ASME. https://doi.org/10.1115/GT2022-80777.