Predictive Maintenance for Aircraft Engines: Because Grounding a Plane is More Expensive than Predicting the Future!

Submitted in Partial Fulfillment of Requirements
for the Degree of

Bachelor of Data Science

By

Britti Paras Vora

Seat No: 31013022024

Guide

Ms. Minal Dive

S K Somaiya College

Somaiya Vidyavihar University

Vidyavihar, Mumbai - 400 077

2022-25

Somaiya Vidyavihar University

S K Somaiya College

Certificate

This is to certify that the project report on dissertation entitled ________________Predictive Maintenance for Aircraft Engines________ is bonafide record of the dissertation work done by _____Britti Paras Vora_________ in the year 2024-25 under the guidance of __Ms. Minal Dive______, Department of Information Technology and Computer Science in partial fulfillment of requirement for the Bachelor of Data Science degree of Somaiya Vidyavihar University.

____________________ ________________________

Guide / Co-Guide Programme Coordinator

____________________ ________________________

Head of Department Director

Date: 21/10/2024

Place: Mumbai-77

Somaiya Vidyavihar University

S K Somaiya College

Certificate of Approval of Examiners

This is to certify that the project report on dissertation entitled ___________Predictive Maintenance for Aircraft Engines_________________ is bonafide record of the dissertation work done by ______Britti Paras Vora_________ in partial fulfillment of requirement for the Bachelor of Data Science degree of Somaiya Vidyavihar University.

________________________ ______________________

External Examiner /Expert Internal Examiner/ Guide

Date: 21/10/2024

Place: Mumbai-77

Somaiya Vidyavihar University

S K Somaiya College

DECLARATION

I declare that this written report submission represents the work done based on my and / or others' ideas with adequately cited and referenced the original source. I also declare that I have adhered to all principles of academic honesty and integrity as I have not misinterpreted or fabricated or falsified any idea/data/fact/source/original work/ matter in my submission.

I understand that any violation of the above will be cause for disciplinary action by the college and may evoke the penal action from the sources which have not been properly cited or from whom proper permission is not sought.

____

Signature of the Student

Britti Paras Vora

Name of the Student

31013022024

Seat No.

Date: 21/10/2024

Place: Mumbai-77

Abstract

This thesis focuses on the predictive maintenance of aircraft engines using machine learning techniques to estimate the Remaining Useful Life (RUL) of engine components. The aviation industry demands high levels of safety and operational efficiency, where traditional maintenance strategies—based on fixed intervals—often lead to either unnecessary maintenance or unexpected failures. Predictive maintenance offers a data-driven solution by leveraging real-time sensor data to anticipate engine failures and optimize maintenance schedules.

The NASA Turbofan Engine Degradation Simulation dataset, which contains time-series sensor data from multiple engines, is utilized in this research. Machine learning models, including Linear Regression, Random Forest, and XGBoost, were developed and evaluated to predict the RUL. The study explores feature selection techniques, such as correlation analysis, and implements standard scaling to improve model performance. Model evaluation metrics like Root Mean Squared Error (RMSE) and R-squared (R^2) were used to assess accuracy.

Although models like Random Forest and XGBoost achieved high accuracy on the training data, they exhibited overfitting when applied to test data, resulting in poor generalization. Hyperparameter tuning using Grid Search Cross-Validation was employed to mitigate this issue, yet challenges remained. The research concludes with suggestions for improving model robustness and applying predictive maintenance techniques to real-world systems, such as enhancing feature engineering and incorporating domain-specific knowledge.

***Key words*:** Predictive Maintenance, Aircraft Engines, Remaining Useful Life (RUL)

Contents

List of Figures

Nomenclature

y	Dependent variable (RUL)
x	Independent variable (sensor readings)
β	Coefficient in linear regression
ε	Error term
R^2	Coefficient of determination
σ	Standard deviation
μ	Mean
Σ	Sum
λ	Regularization parameter
n	Number of samples
p	Number of features
$\hat{y}$	Predicted value
t	Time
α	Learning rate (in gradient boosting algorithms)
∇	Gradient
$\in$	Element of
$\propto$	Proportional to

Chapter 1
Introduction

This chapter introduces the concept of predictive maintenance in the aviation industry, focusing on aircraft engines. It explains the importance of Remaining Useful Life (RUL) prediction and its potential impact on operational efficiency and safety. The chapter also outlines the motivation behind the research and sets the stage for the subsequent chapters.

1.1 Predictive maintenance

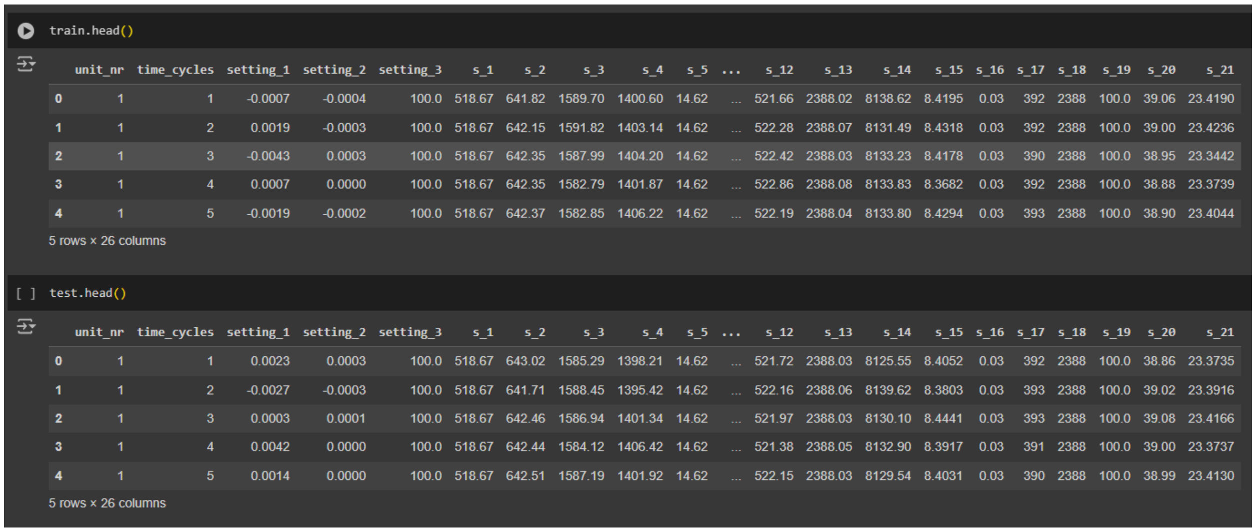
The aviation industry is a complex and safety-critical field where equipment reliability and operational efficiency are paramount. Aircraft engines, in particular, are among the most important and expensive components, and their failure can lead to costly downtime, maintenance, and safety risks. Traditional maintenance strategies, such as preventive or scheduled maintenance, are based on fixed time intervals or usage cycles, which often fail to reflect the real-time condition of the engine. This can lead to unnecessary maintenance activities or unexpected failures, which negatively affect operational efficiency and safety.

In recent years, predictive maintenance has emerged as a promising solution to address these challenges. Airplanes are more capable than ever of recording large amounts of sensor data on almost all its in-flight components. Predictive maintenance leverages data from sensors and machine learning algorithms to predict when an aircraft engine is likely to fail or require service. This data-driven approach allows maintenance activities to be scheduled just before a failure occurs, optimizing engine performance, reducing unplanned downtime, and extending the lifespan of engine components. This approach promises cost savings compared to routine or time-based preventive maintenance, because the work is performed only when warranted.

Beginning with Data Set[1] FD001, it is the simplest of the series.
Train trajectories: one hundred engines.
Test trajectories: one hundred engines.
There is only one fault mode.

Simulations of several turbofan engines throughout time are included in the datasets; the following details are contained in each row:

1. Engine number
2. Time, in cycles
3. There are three settings for operations.
4. 21 sensor readings. Which may include features like pressure, temperature, vibration, etc.



```
train.head()
```

	unit_nr	time_cycles	setting_1	setting_2	setting_3	s_1	s_2	s_3	s_4	s_5	...	s_12	s_13	s_14	s_15	s_16	s_17	s_18	s_19	s_20	s_21
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	39.06	23.4190
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	39.00	23.4236
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	38.95	23.3442
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	38.88	23.3739
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	38.90	23.4044

5 rows × 26 columns

```
[ ] test.head()
```

	unit_nr	time_cycles	setting_1	setting_2	setting_3	s_1	s_2	s_3	s_4	s_5	...	s_12	s_13	s_14	s_15	s_16	s_17	s_18	s_19	s_20	s_21
0	1	1	0.0023	0.0003	100.0	518.67	643.02	1585.29	1398.21	14.62	...	521.72	2388.03	8125.55	8.4052	0.03	392	2388	100.0	38.86	23.3735
1	1	2	-0.0027	-0.0003	100.0	518.67	641.71	1588.45	1395.42	14.62	...	522.16	2388.06	8139.62	8.3803	0.03	393	2388	100.0	39.02	23.3916
2	1	3	0.0003	0.0001	100.0	518.67	642.46	1586.94	1401.34	14.62	...	521.97	2388.03	8130.10	8.4441	0.03	393	2388	100.0	39.08	23.4166
3	1	4	0.0042	0.0000	100.0	518.67	642.44	1584.12	1406.42	14.62	...	521.38	2388.05	8132.90	8.3917	0.03	391	2388	100.0	39.00	23.3737
4	1	5	0.0014	0.0000	100.0	518.67	642.51	1587.19	1401.92	14.62	...	522.15	2388.03	8129.54	8.4031	0.03	390	2388	100.0	38.99	23.4130

5 rows × 26 columns

1.2 Motivation

The motivation behind selecting predictive maintenance of aircraft engines as the focus of this thesis stems from the increasing reliance on data analytics in the aviation industry and the critical importance of aircraft engine reliability. By utilizing real-time sensor data, predictive maintenance helps reduce unnecessary maintenance and avoid in-flight failures. With the integration of advanced machine learning models, operators can predict the Remaining Useful Life (RUL) of engine components, allowing for timely interventions, thereby reducing operational costs and improving safety margins.

Furthermore, as the aviation sector grows, the demand for more efficient and accurate maintenance strategies continues to rise. Predictive maintenance not only offers significant economic advantages by reducing maintenance costs, but it also improves environmental sustainability by minimizing resource consumption and enhancing engine performance. This thesis will explore the development and application of predictive models to improve engine maintenance strategies, ensuring that aircraft engines operate safely, efficiently, and with reduced operational interruptions.

Traditional reactive maintenance or scheduled preventive maintenance often results in unnecessary repairs or late detection of defects, leading to higher maintenance costs and downtime. By using predictive maintenance, airlines can optimize maintenance programs based on the current condition of engines, reducing the frequency of unnecessary inspections, extending engine life and reducing costs on unplanned repairs. This approach not only lowers maintenance costs but also ensures higher aircraft availability, reducing the time planes spend grounded for repairs and keeping the fleet operational.

Chapter 2
Literature survey

This chapter presents a comprehensive review of existing literature on predictive maintenance and RUL estimation. It covers the evolution of techniques from early statistical approaches to modern machine learning models. The chapter also discusses data preprocessing methods, challenges in RUL prediction, and recent advancements in the field, identifying gaps that this research aims to address.

2.1 Early Research on Predictive Maintenance and RUL Estimation

The concept of predictive maintenance for aircraft engines emerged from the growing importance of monitoring systems that gather data through onboard sensors. Early work by[1] proposed methods based on statistical approaches and traditional time-series analysis, which used historical failure data to model engine degradation. While effective, these models were limited by their dependency on extensive failure history and inability to capture non-linear relationships in the data.

The shift towards data-driven models allowed researchers to improve the accuracy of predictions. In [2], the authors explored the use of regression-based approaches for RUL estimation using sensor data collected from aircraft engines. Although linear regression models were simple and interpretable, their performance was often inadequate due to the complex, non-linear nature of engine degradation processes.

The prediction of Remaining Useful Life (RUL) of machinery, especially aircraft engines, has gained significant attention in the realm of predictive maintenance. The objective of this review is to analyze different strategies employed by researchers to predict RUL and identify key areas of innovation and improvement.

Statistical methods were among the first approaches used for RUL estimation. Weibull distribution models and proportional hazard models have been extensively used [2]. However, these models rely heavily on assumptions about the underlying failure mechanisms, which may not always hold in real-world scenarios, particularly when dealing with complex systems like aircraft engines.

2.1.1 Machine Learning Models

Machine learning techniques have seen a surge in usage due to their ability to learn directly from data without relying on explicit failure mechanisms.

- Linear Regression: Linear models are among the simplest, yet they have proven effective for RUL prediction in specific contexts. For instance, [3] employed linear regression to predict the RUL of aircraft engines, but noted that this model suffers from limitations in capturing non-linear patterns in sensor data.
- Random Forest Regression: Random forests offer an ensemble-based approach, aggregating predictions from multiple decision trees to produce robust predictions. [4] showed that random forests outperform traditional linear models in RUL prediction by capturing complex interactions between features. However, they can be prone to overfitting if not properly tuned.

- XGBoost: Gradient boosting techniques such as XGBoost have shown significant promise in the domain of predictive maintenance. According to [5], XGBoost not only provides better prediction accuracy compared to random forests but also effectively handles large-scale data with noise and outliers.

2.1.2 Deep Learning Models

Recent advancements have seen the emergence of deep learning models like Long Short-Term Memory (LSTM) networks for RUL prediction, particularly in handling time-series data from sensor readings. LSTMs are adept at capturing temporal dependencies, which are critical in accurately predicting RUL. As per [6], deep learning models such as LSTMs and convolutional neural networks (CNNs) have achieved state-of-the-art results, but require extensive computational resources and large datasets for training.

2.2 Data Preprocessing for RUL Prediction

Data preprocessing is a crucial step in any RUL prediction task. Studies such as emphasize the importance of feature selection and normalization to enhance model performance. High-dimensional sensor data can lead to overfitting if irrelevant features are not properly eliminated, as noted by. Therefore, dimensionality reduction techniques like Principal Component Analysis (PCA) and correlation-based feature selection have been widely used in predictive maintenance studies.

2.3 Addressing Overfitting and Generalization

One of the key challenges in RUL prediction is ensuring that the model generalizes well to unseen data. Overfitting occurs when a model performs exceptionally well on training data but poorly on test data. Techniques like cross-validation, regularization, and hyperparameter tuning are essential in addressing overfitting and improving generalization.

2.4 Optimization Techniques and Recent Advances

Recent studies have focused on improving the predictive accuracy of machine learning models by using hyperparameter optimization methods such as Grid Search Cross-Validation. Grid Search CV is used to optimize Random Forest and Support Vector Machine models for RUL prediction. By exhaustively searching the parameter space, they were able to find the best model configurations, significantly improving test performance compared to models with default hyperparameters.

XGBoost, an advanced gradient boosting method, was introduced as a competitive alternative for RUL prediction. XGBoost is known for its efficiency and accuracy, especially in handling large datasets with complex interactions. In this work, XGBoost was found to outperform other models like Random Forest on training data; however, generalization to test data remained a challenge, indicating the need for better regularization techniques.

In addition to model optimization, feature engineering has also been crucial in improving predictive maintenance models. It investigated various feature extraction methods to enhance the predictive power of models. They suggested that using domain-specific knowledge to engineer features such as rolling averages or rate of change from raw sensor data could lead to better model performance, especially for complex, multi-sensor environments like aircraft engines.

2.5 Objectives of the Thesis Work

Based on the literature reviewed, the following objectives have been formulated for this thesis:

- To develop and compare various machine learning models for RUL prediction of aircraft engines using real-world sensor data.
- To address overfitting and ensure model generalization by implementing appropriate validation techniques.
- To explore feature selection methods and their impact on model performance.
- To provide a robust framework for predictive maintenance in aircraft engines that can be extended to other domains in the future.

The research builds upon existing methodologies while proposing enhancements in feature selection, model evaluation, and generalization techniques to ensure accurate RUL prediction.

Chapter 3

Mathematical and Logical Foundations for Predictive Maintenance

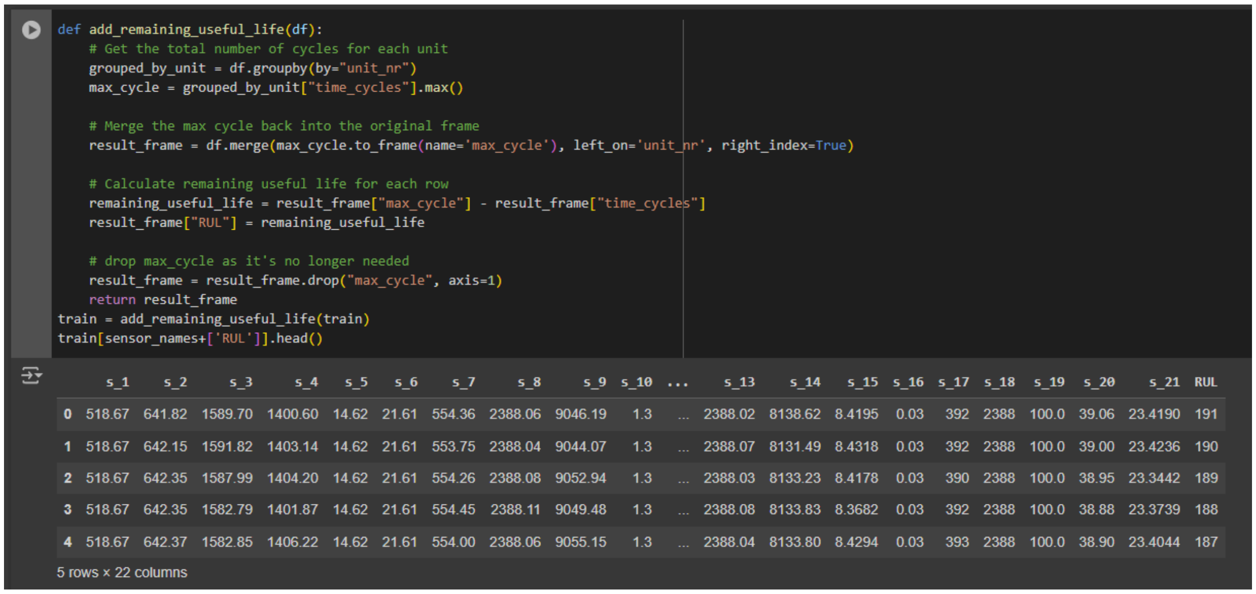
This chapter provides the theoretical underpinnings of the machine learning models used in this research. It explains the mathematical concepts behind regression analysis, ensemble methods like Random Forest and XGBoost, and regularization techniques. The chapter also introduces the evaluation metrics used to assess model performance, laying the groundwork for the implementation phase of the research.

3.1 Remaining Useful Life (RUL) Prediction

The Remaining Useful Life (RUL) prediction problem is framed as a regression task, where the objective is to predict the number of remaining operational cycles for an aircraft engine until it fails. Mathematically, the problem can be described as follows:

RUL = Max Life Time Cycle of Engine-Current Time Cycle

Max Life Time Cycle of Engine means the time when the engine degraded.



```python
def add_remaining_useful_life(df):
    # Get the total number of cycles for each unit
    grouped_by_unit = df.groupby(by="unit_nr")
    max_cycle = grouped_by_unit["time_cycles"].max()

    # Merge the max cycle back into the original frame
    result_frame = df.merge(max_cycle.to_frame(name='max_cycle'), left_on='unit_nr', right_index=True)

    # Calculate remaining useful life for each row
    remaining_useful_life = result_frame["max_cycle"] - result_frame["time_cycles"]
    result_frame["RUL"] = remaining_useful_life

    # drop max_cycle as it's no longer needed
    result_frame = result_frame.drop("max_cycle", axis=1)
    return result_frame
train = add_remaining_useful_life(train)
train[sensor_names+['RUL']].head()
```

```
      s_1     s_2      s_3      s_4    s_5    s_6     s_7      s_8      s_9  s_10  ...     s_13     s_14    s_15  s_16  s_17  s_18   s_19   s_20     s_21  RUL
0  518.67  641.82  1589.70  1400.60  14.62  21.61  554.36  2388.06  9046.19   1.3  ...  2388.02  8138.62  8.4195  0.03   392  2388  100.0  39.06  23.4190  191
1  518.67  642.15  1591.82  1403.14  14.62  21.61  553.75  2388.04  9044.07   1.3  ...  2388.07  8131.49  8.4318  0.03   392  2388  100.0  39.00  23.4236  190
2  518.67  642.35  1587.99  1404.20  14.62  21.61  554.26  2388.08  9052.94   1.3  ...  2388.03  8133.23  8.4178  0.03   390  2388  100.0  38.95  23.3442  189
3  518.67  642.35  1582.79  1401.87  14.62  21.61  554.45  2388.11  9049.48   1.3  ...  2388.08  8133.83  8.3682  0.03   392  2388  100.0  38.88  23.3739  188
4  518.67  642.37  1582.85  1406.22  14.62  21.61  554.00  2388.06  9055.15   1.3  ...  2388.04  8133.80  8.4294  0.03   393  2388  100.0  38.90  23.4044  187

5 rows × 22 columns
```

3.2 Regression Analysis

Regression analysis is one of the key mathematical techniques used in predictive maintenance for predicting continuous outcomes, such as the RUL of a machine. The goal of regression is to model the relationship between the dependent variable (in this case, RUL) and one or more independent variables (such as sensor readings).

3.2.1 Linear Regression

Linear Regression is one of the simplest models used for regression tasks. It assumes a linear relationship between the independent variables (features) and the dependent variable (RUL). The model attempts to find the best-fitting line through the data points. The general form of a multiple linear regression model is:

$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_{\square} x_{\square} + \varepsilon$

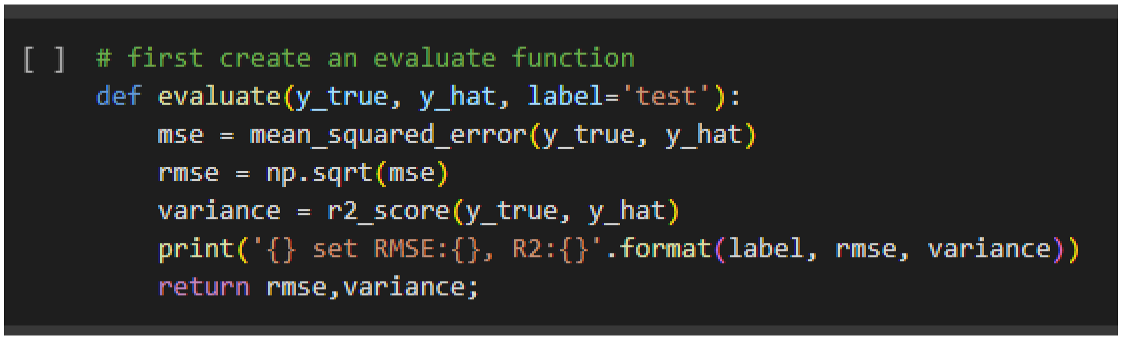
Where:

- y is the dependent variable (e.g., time to failure)
- $x_1, x_2, \ldots, x_{\square}$ are independent variables (e.g., sensor readings)
- $\beta_0, \beta_1, \beta_2, \ldots, \beta_{\square}$ are the coefficients to be estimated
- ε is the error term

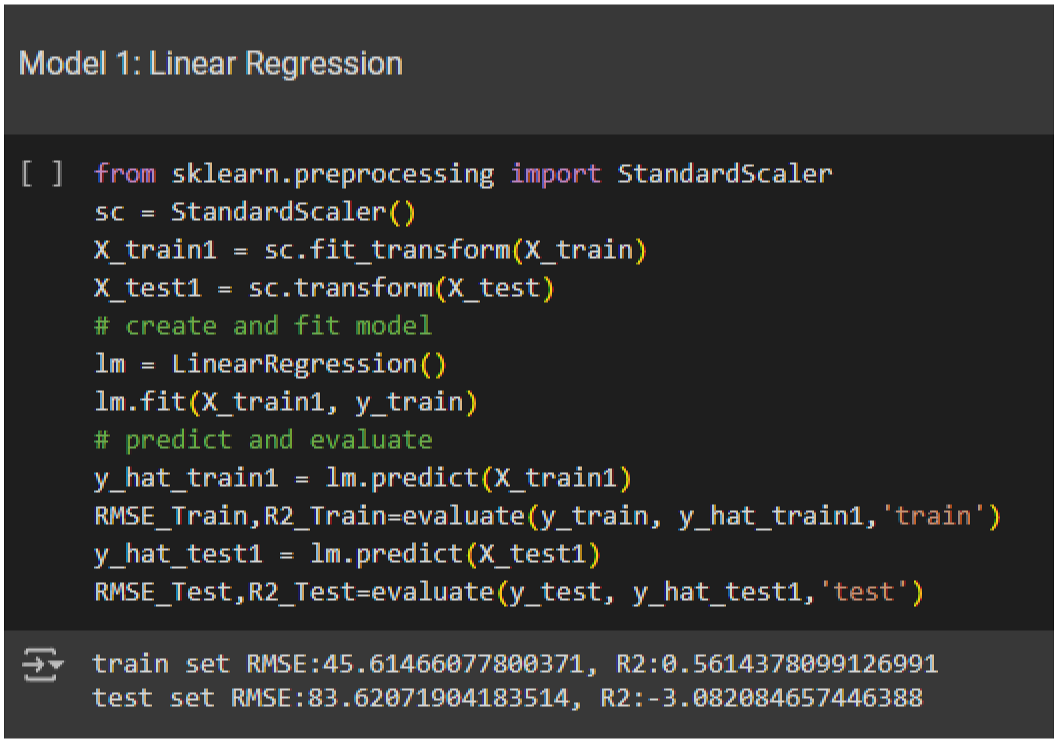
The goal is to find the values of β that minimize the sum of squared residuals:

$\min \Sigma(y_i - \hat{y}_i)^2$

Where y_i are the actual values and $\hat{y}_i$ are the predicted values.



```
[ ]  # first create an evaluate function
     def evaluate(y_true, y_hat, label='test'):
         mse = mean_squared_error(y_true, y_hat)
         rmse = np.sqrt(mse)
         variance = r2_score(y_true, y_hat)
         print('{} set RMSE:{}, R2:{}'.format(label, rmse, variance))
         return rmse,variance;
```


Model 1: Linear Regression

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train1 = sc.fit_transform(X_train)
X_test1 = sc.transform(X_test)
# create and fit model
lm = LinearRegression()
lm.fit(X_train1, y_train)
# predict and evaluate
y_hat_train1 = lm.predict(X_train1)
RMSE_Train,R2_Train=evaluate(y_train, y_hat_train1,'train')
y_hat_test1 = lm.predict(X_test1)
RMSE_Test,R2_Test=evaluate(y_test, y_hat_test1,'test')
```

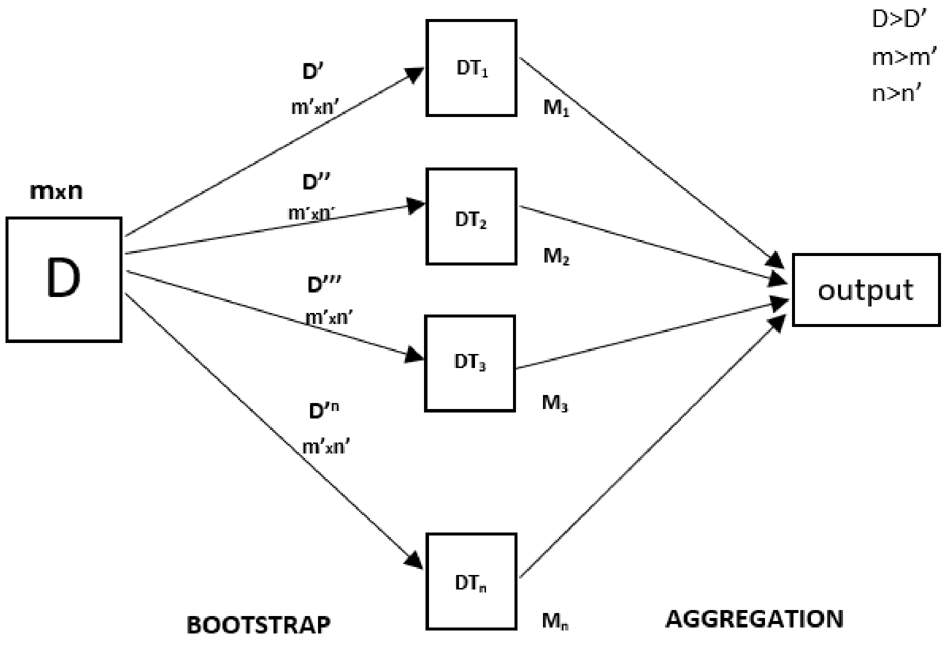
```
train set RMSE:45.61466077800371, R2:0.5614378099126991
test set RMSE:83.62071904183514, R2:-3.082084657446388
```

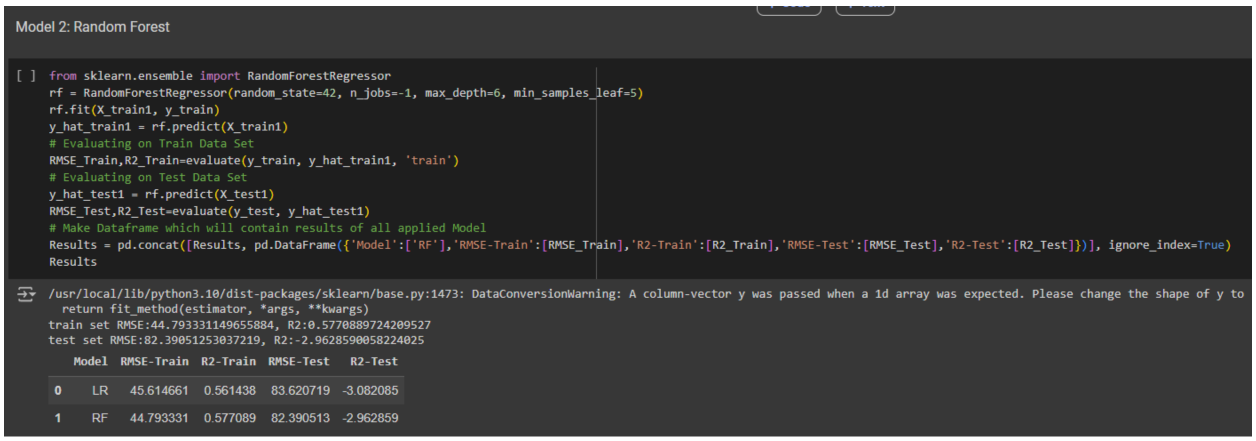
3.2.2 Random Forest Regression

Random Forest is an ensemble learning method that combines multiple decision trees to make more accurate predictions. It works by constructing multiple decision trees during training and averaging the predictions from all the trees. The algorithm introduces randomness in two ways:

1. Random selection of subsets of features for each tree.
2. Random sampling of data points with replacement (bagging).

The Random Forest algorithm also calculates the importance of each feature by measuring the decrease in prediction error when that feature is used for splitting the data. Feature importance is crucial for understanding which sensors contribute most to predicting the RUL.




Model 2: Random Forest

```
[ ] from sklearn.ensemble import RandomForestRegressor
    rf = RandomForestRegressor(random_state=42, n_jobs=-1, max_depth=6, min_samples_leaf=5)
    rf.fit(X_train1, y_train)
    y_hat_train1 = rf.predict(X_train1)
    # Evaluating on Train Data Set
    RMSE_Train,R2_Train=evaluate(y_train, y_hat_train1, 'train')
    # Evaluating on Test Data Set
    y_hat_test1 = rf.predict(X_test1)
    RMSE_Test,R2_Test=evaluate(y_test, y_hat_test1)
    # Make Dataframe which will contain results of all applied Model
    Results = pd.concat([Results, pd.DataFrame({'Model':['RF'],'RMSE-Train':[RMSE_Train],'R2-Train':[R2_Train],'RMSE-Test':[RMSE_Test],'R2-Test':[R2_Test]})], ignore_index=True)
    Results
```

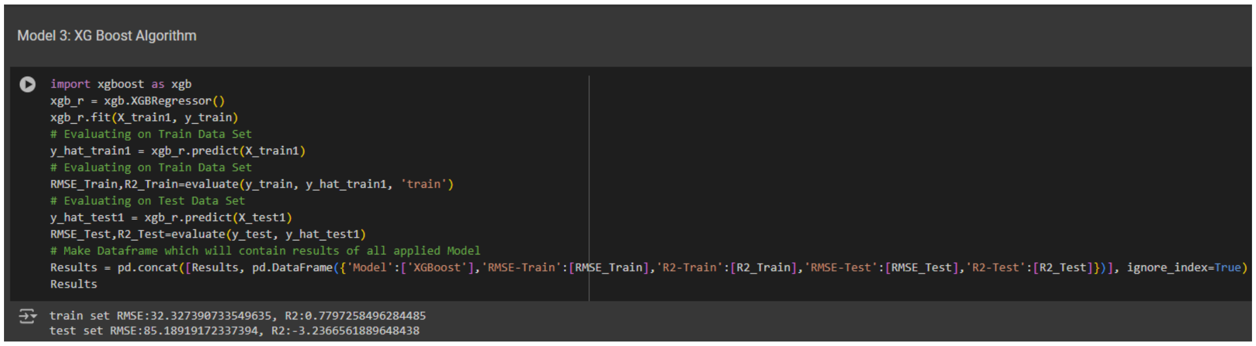
```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1473: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to
  return fit_method(estimator, *args, **kwargs)
train set RMSE:44.793331149655884, R2:0.5770889724209527
test set RMSE:82.39051253037219, R2:-2.9628590058224025
```

	Model	RMSE-Train	R2-Train	RMSE-Test	R2-Test
0	LR	45.614661	0.561438	83.620719	-3.082085
1	RF	44.793331	0.577089	82.390513	-2.962859

3.2.3 XGBoost Regression

XGBoost (Extreme Gradient Boosting) is an advanced implementation of gradient boosting. It sequentially adds trees to correct errors made by previous models. Unlike Random Forest, where trees are independent, XGBoost builds trees iteratively, and each new tree focuses on reducing the residual errors of the previous trees.

The XGBoost objective function includes two parts: the loss function (usually squared error for regression) and the regularization term to prevent overfitting


Model 3: XG Boost Algorithm

```
import xgboost as xgb
xgb_r = xgb.XGBRegressor()
xgb_r.fit(X_train1, y_train)
# Evaluating on Train Data Set
y_hat_train1 = xgb_r.predict(X_train1)
# Evaluating on Train Data Set
RMSE_Train,R2_Train=evaluate(y_train, y_hat_train1, 'train')
# Evaluating on Test Data Set
y_hat_test1 = xgb_r.predict(X_test1)
RMSE_Test,R2_Test=evaluate(y_test, y_hat_test1)
# Make Dataframe which will contain results of all applied Model
Results = pd.concat([Results, pd.DataFrame({'Model':['XGBoost'],'RMSE-Train':[RMSE_Train],'R2-Train':[R2_Train],'RMSE-Test':[RMSE_Test],'R2-Test':[R2_Test]})], ignore_index=True)
Results
```

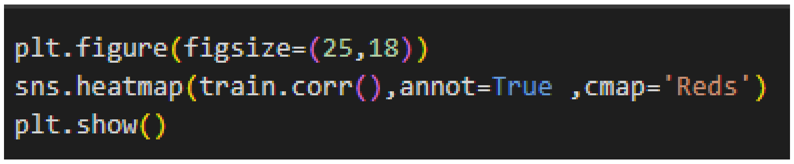
```
train set RMSE:32.327390733549635, R2:0.7797258496284485
test set RMSE:85.18919172337394, R2:-3.2366561889648438
```

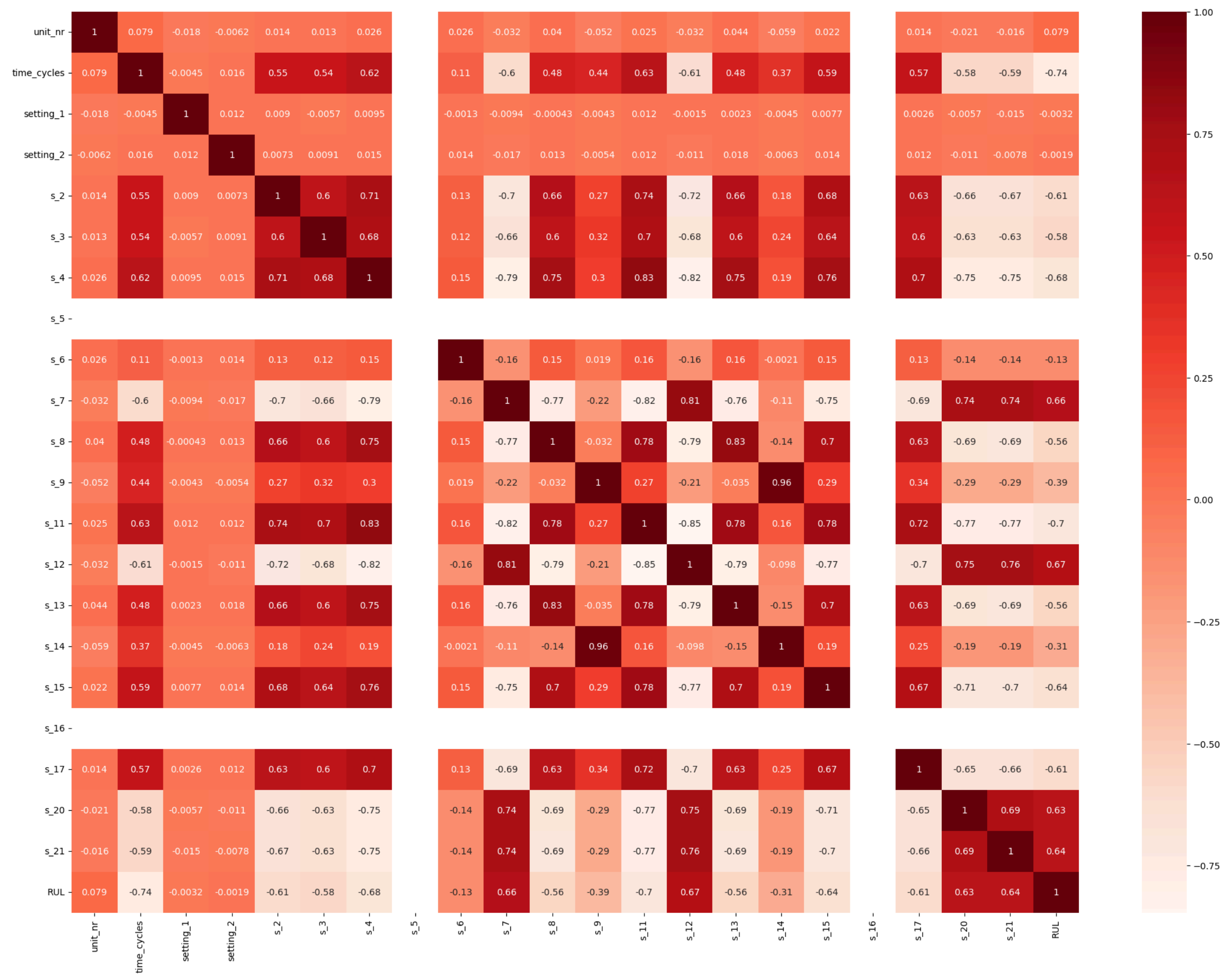
The objective is to minimize the total loss by adjusting the model parameters.

3.3 Feature Selection and Feature Scaling

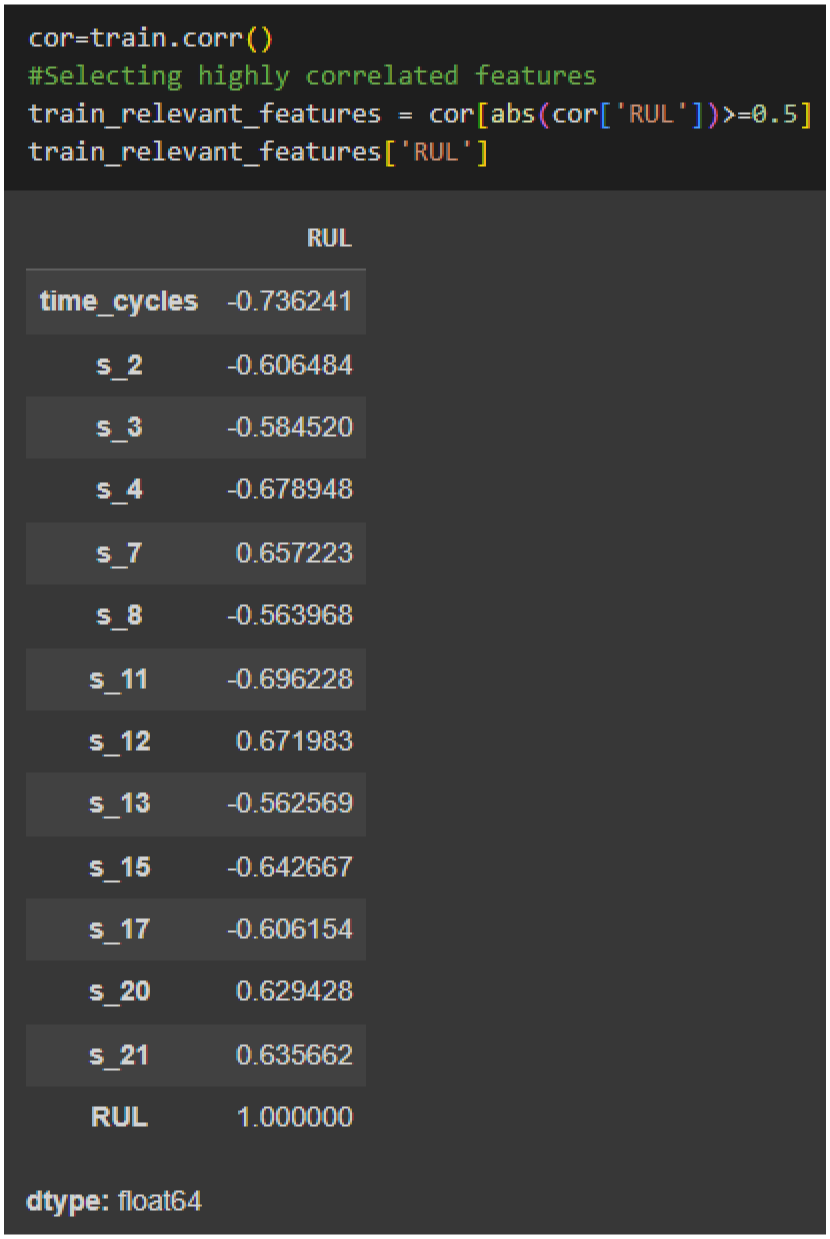
Feature selection is crucial to reduce model complexity and improve performance. The correlation heatmap helps identify features that are strongly correlated with RUL. Highly correlated features are selected for model training, while irrelevant features are removed.

Feature scaling ensures that all features are on the same scale, which is particularly important for models like Linear Regression and XGBoost. In this thesis, StandardScaler is used to standardize features by removing the mean and scaling to unit variance


```
plt.figure(figsize=(25,18))
sns.heatmap(train.corr(),annot=True ,cmap='Reds')
plt.show()
```



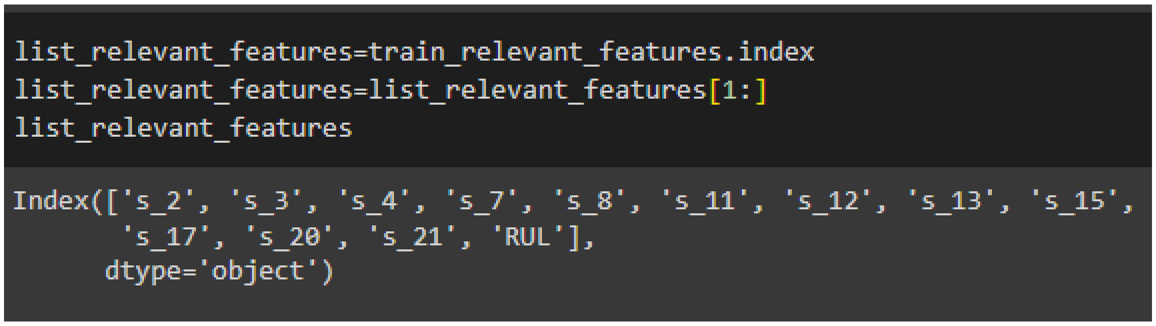
The provided heatmap shows the correlation between sensor readings, settings, and Remaining Useful Life (RUL). Notably, sensors like s_10, s_11, s_12, and s_13 show strong negative correlations with RUL, making them important predictors. Features like s_4 and s_5 are highly correlated with each other, indicating potential redundancy, while settings have weak correlations with RUL. The negative correlation between time_cycles and RUL (~-0.74) aligns with the expected decrease in useful life as time progresses.



```
cor=train.corr()
#Selecting highly correlated features
train_relevant_features = cor[abs(cor['RUL'])>=0.5]
train_relevant_features['RUL']
```

```
                  RUL
time_cycles  -0.736241
        s_2  -0.606484
        s_3  -0.584520
        s_4  -0.678948
        s_7   0.657223
        s_8  -0.563968
       s_11  -0.696228
       s_12   0.671983
       s_13  -0.562569
       s_15  -0.642667
       s_17  -0.606154
       s_20   0.629428
       s_21   0.635662
        RUL   1.000000

dtype: float64
```



```
list_relevant_features=train_relevant_features.index
list_relevant_features=list_relevant_features[1:]
list_relevant_features
```

```
Index(['s_2', 's_3', 's_4', 's_7', 's_8', 's_11', 's_12', 's_13', 's_15',
       's_17', 's_20', 's_21', 'RUL'],
      dtype='object')
```

3.4 Model Evaluation Metrics

The performance of the models is evaluated using the following metrics:

3.4.1 Root Mean Squared Error (RMSE)

RMSE is the square root of the mean of the squared differences between the actual and predicted values. It is calculated as:

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

where:

- yi is the actual value for the ith observation.
- ŷi is the predicted value for the ith observation.
- N is the number of observations.
- P is the number of parameter estimates, including the constant.

Lower RMSE indicates better model performance.

3.4.2 Coefficient of Determination (R^2 Score)

R-Squared (R^2) is a statistical measure used to determine the proportion of variance in a dependent variable that can be predicted or explained by an independent variable. In other words, R-Squared shows how well a regression model (independent variable) predicts the outcome of observed data (dependent variable). It is calculated as:

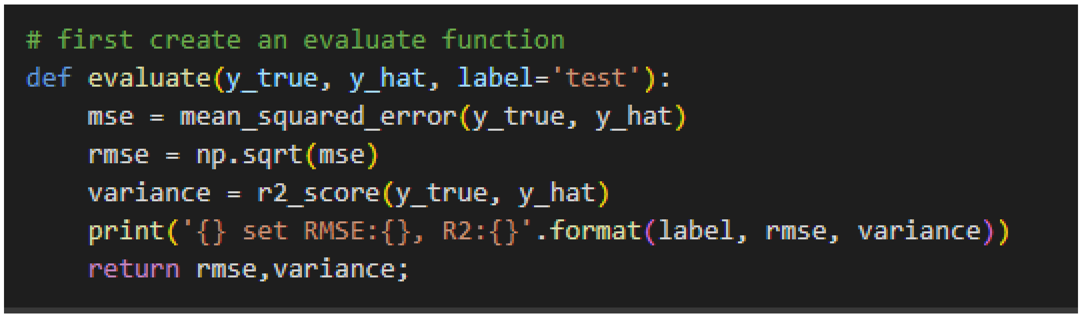
$R^2 = 1 - RSS/TSS$

where,

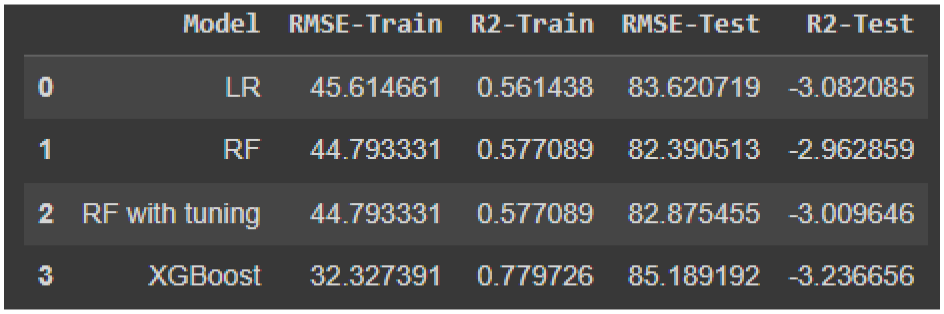
- R^2 = coefficient of determination

- RSS = sum of squares of residuals
- TSS = total sum of squares

An R^2 value closer to 1 indicates a better fit.



```
# first create an evaluate function
def evaluate(y_true, y_hat, label='test'):
    mse = mean_squared_error(y_true, y_hat)
    rmse = np.sqrt(mse)
    variance = r2_score(y_true, y_hat)
    print('{} set RMSE:{}, R2:{}'.format(label, rmse, variance))
    return rmse,variance;
```



	Model	RMSE-Train	R2-Train	RMSE-Test	R2-Test
0	LR	45.614661	0.561438	83.620719	-3.082085
1	RF	44.793331	0.577089	82.390513	-2.962859
2	RF with tuning	44.793331	0.577089	82.875455	-3.009646
3	XGBoost	32.327391	0.779726	85.189192	-3.236656

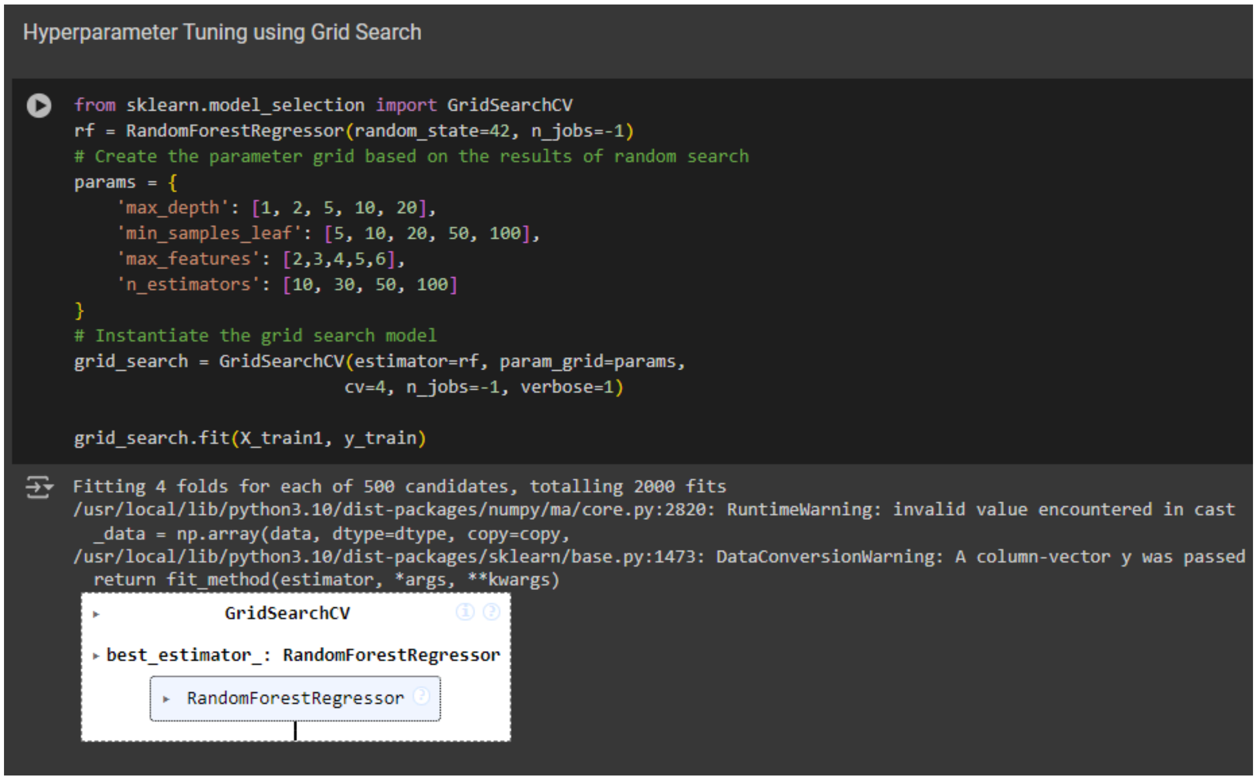
3.5 Grid Search Cross-Validation (Grid Search CV)

Grid Search CV is used to tune the hyperparameters of machine learning models. In this approach, a predefined grid of hyperparameter values is created, and the model is trained and evaluated for every possible combination of these parameters. The combination that yields the best performance is selected.

The key steps involved in Grid Search CV are:

1. Parameter Grid: Define the set of hyperparameters and their possible values.
2. Cross-Validation: Split the dataset into training and validation sets multiple times and train the model on different splits.
3. Evaluation: Evaluate model performance for each combination of hyperparameters and select the one with the best score.

For instance, in this thesis, hyperparameters such as max depth, min samples leaf, and number of estimators for Random Forest were tuned using Grid Search CV.


Hyperparameter Tuning using Grid Search

```
from sklearn.model_selection import GridSearchCV
rf = RandomForestRegressor(random_state=42, n_jobs=-1)
# Create the parameter grid based on the results of random search
params = {
    'max_depth': [1, 2, 5, 10, 20],
    'min_samples_leaf': [5, 10, 20, 50, 100],
    'max_features': [2,3,4,5,6],
    'n_estimators': [10, 30, 50, 100]
}
# Instantiate the grid search model
grid_search = GridSearchCV(estimator=rf, param_grid=params,
                          cv=4, n_jobs=-1, verbose=1)

grid_search.fit(X_train1, y_train)
```

```
Fitting 4 folds for each of 500 candidates, totalling 2000 fits
/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820: RuntimeWarning: invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1473: DataConversionWarning: A column-vector y was passed
  return fit_method(estimator, *args, **kwargs)
```

Chapter 4
Implementation of Predictive maintenance

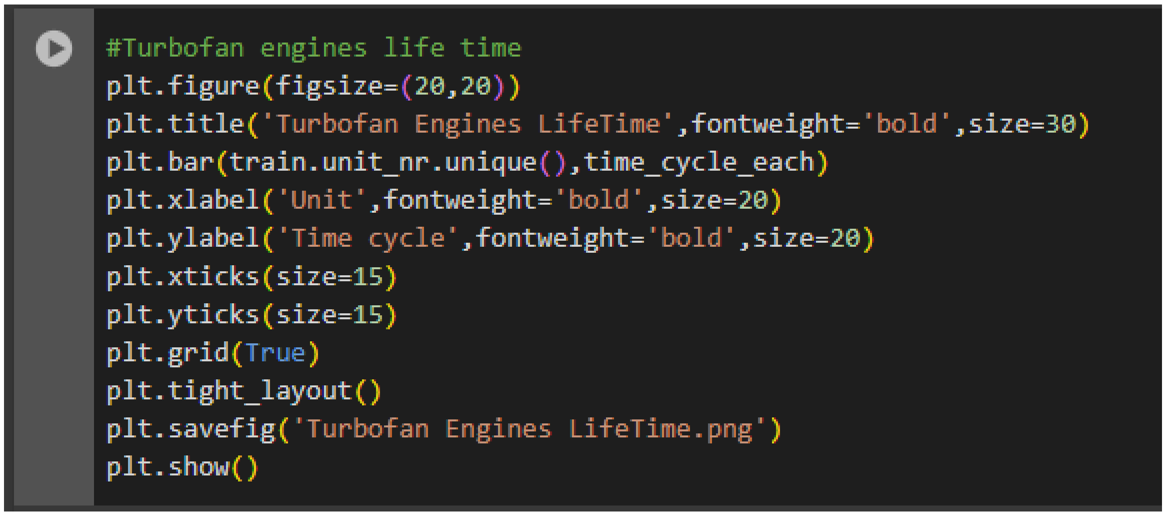
This chapter details the practical implementation of predictive maintenance models using the NASA Turbofan Engine dataset. It covers data preparation, feature selection, and the application of various machine learning algorithms. The chapter walks through the process of model training and evaluation, presenting initial results and comparing the performance of different models in predicting Remaining Useful Life (RUL).

4.1 Data Preparation

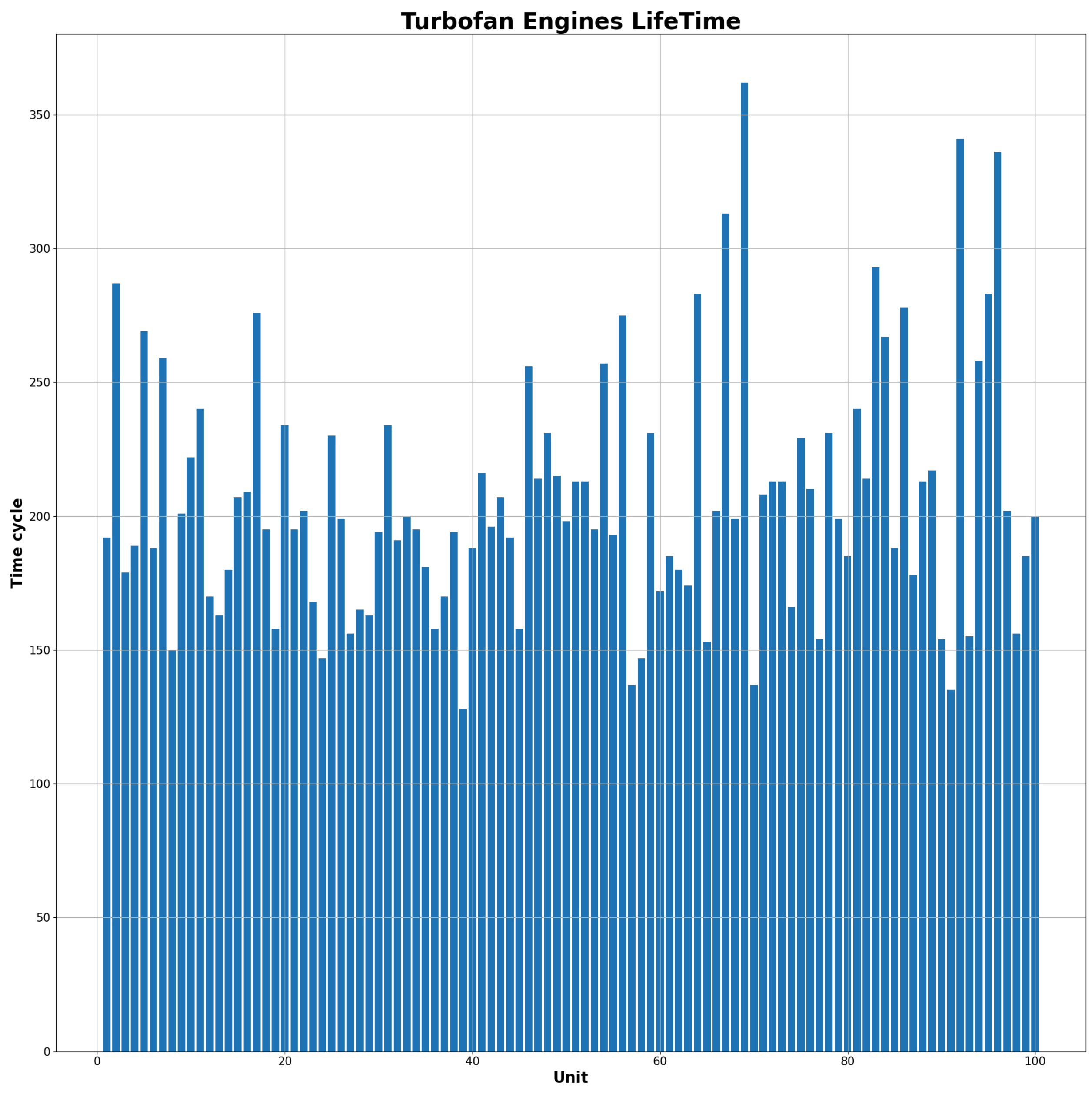
The dataset used for this project is obtained from NASA's Turbofan Engine Degradation Simulation Data Set. The training data consists of multivariate time series sensor data, with each row representing a snapshot of an engine's condition at a specific time cycle.

4.1.1 Exploratory Data Analysis

We visualize the lifetime of each engine:

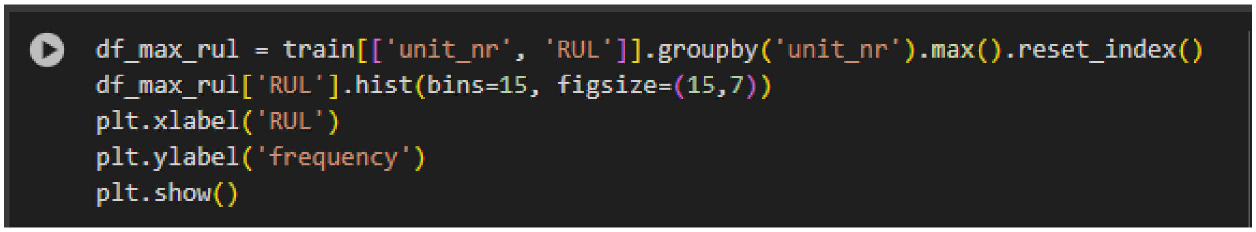


```python
#Turbofan engines life time
plt.figure(figsize=(20,20))
plt.title('Turbofan Engines LifeTime',fontweight='bold',size=30)
plt.bar(train.unit_nr.unique(),time_cycle_each)
plt.xlabel('Unit',fontweight='bold',size=20)
plt.ylabel('Time cycle',fontweight='bold',size=20)
plt.xticks(size=15)
plt.yticks(size=15)
plt.grid(True)
plt.tight_layout()
plt.savefig('Turbofan Engines LifeTime.png')
plt.show()
```

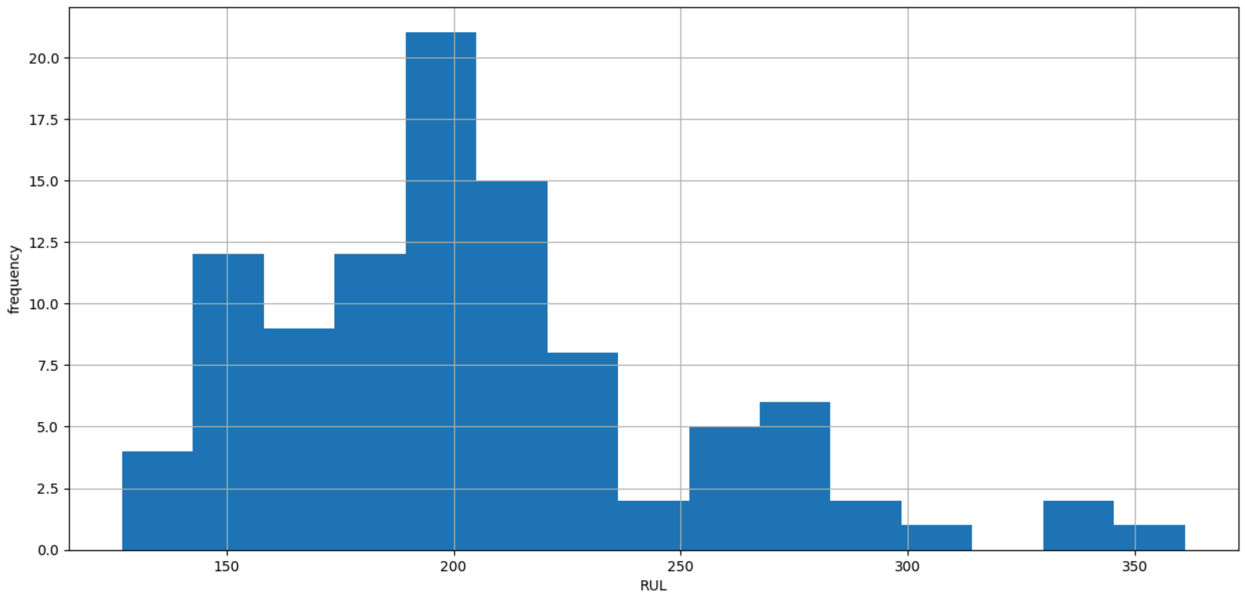


This bar chart illustrates the lifetime of different turbofan engines, measured by the number of time cycles each unit runs. The chart shows varying life spans across the 100 engines, with some units reaching over 350 cycles and others below 150. The distribution reflects variability in engine performance or operational conditions, where some engines undergo more wear and usage cycles than others. This highlights the importance of predictive maintenance, as engines with higher cycle counts are more likely to require intervention or maintenance to prevent failure.

Visualizing the frequency of RUL:



```
df_max_rul = train[['unit_nr', 'RUL']].groupby('unit_nr').max().reset_index()
df_max_rul['RUL'].hist(bins=15, figsize=(15,7))
plt.xlabel('RUL')
plt.ylabel('frequency')
plt.show()
```



This histogram shows the distribution of Remaining Useful Life (RUL) for the turbofan engines. The majority of engines have an RUL between 150 and 200 cycles, with the highest frequency around 200 cycles. As the RUL increases, the number of engines decreases, showing a long tail beyond 250 cycles. This suggests that most engines will likely need maintenance or replacement after around 200 cycles, while fewer engines have significantly longer lifespans. The distribution can help in planning maintenance schedules and predicting when most engines will require intervention.

4.1.2 Feature Selection and Preprocessing

The raw data contains sensor readings and operational settings for multiple engines. The steps involved in preprocessing include:

1. **Sensor Selection**: Certain sensors (such as s_1, s_10, s_18, and s_19) and settings (setting_3) were removed from the dataset as they did not contribute significant information for RUL prediction, based on the correlation analysis.
2. **RUL Calculation**: The RUL was calculated for each engine in the training set by determining the difference between the maximum number of cycles an engine ran and its current cycle at each time step.

RUL = Max Life Time Cycle of Engine-Current Time Cycle

Max Life Time Cycle of Engine means the time when the engine has degraded.

This computation helps transform the time-series data into a regression problem, where the target variable is the RUL.

4.1.3 Scaling the Data

Before feeding the data into machine learning models, feature scaling was performed using StandardScaler to normalize the features. This ensures that all the sensor readings are on the same scale, which is important for models that rely on distance metrics or coefficients, such as Linear Regression.

The standardized feature xi′ for a sensor reading xi is given by:

$$x_i' = \frac{x_i - \mu}{\sigma}$$

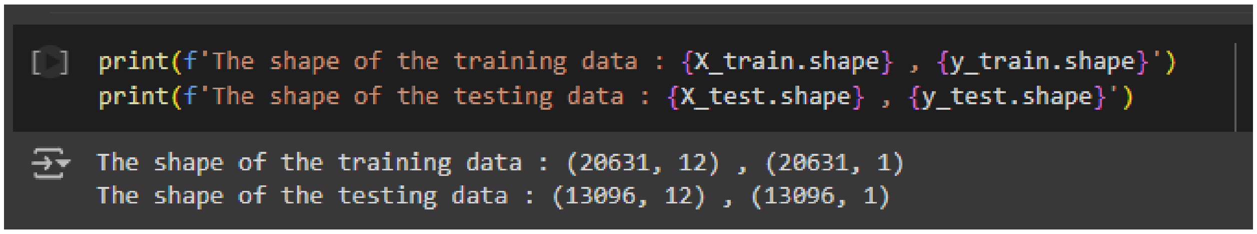
Where:

- μ is the mean of the feature,
- σ is the standard deviation.

4.1.4 Train-Test Split

The dataset was split into training and testing sets. The training set includes the complete time-series data for each engine, whereas the test set contains a partial history of sensor readings, with the task of predicting the RUL at the last available cycle.

- Training Set Size: (20631 samples)
- Testing Set Size: (13096 samples)



```
print(f'The shape of the training data : {X_train.shape} , {y_train.shape}')
print(f'The shape of the testing data : {X_test.shape} , {y_test.shape}')
```

```
The shape of the training data : (20631, 12) , (20631, 1)
The shape of the testing data : (13096, 12) , (13096, 1)
```

4.2 Machine Learning Models

Various machine learning models were applied to predict the RUL. Each model was evaluated based on its performance on the training and testing datasets using evaluation metrics like Root Mean Squared Error (RMSE) and R^2 score.

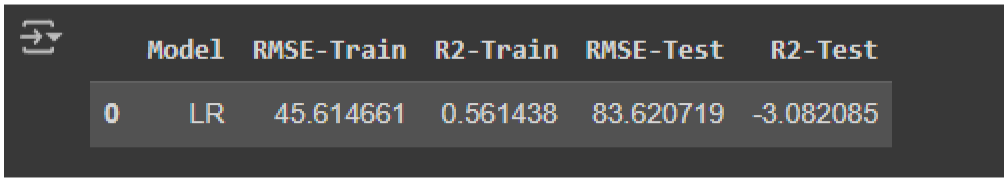
4.2.1 Linear Regression (LR)

Linear Regression was the first model used for RUL prediction. The model assumes a linear relationship between the sensor features and the RUL. After scaling the features, the model was trained to fit the training data.

The performance of the Linear Regression model was as follows:

- RMSE (Train): 45.61
- R^2 (Train): 0.561
- RMSE (Test): 83.62
- R^2 (Test): -3.08

The results indicate that Linear Regression struggles to generalize on the test data, with a high RMSE and a negative R² score.



	Model	RMSE-Train	R2-Train	RMSE-Test	R2-Test
0	LR	45.614661	0.561438	83.620719	-3.082085

4.2.2 Random Forest Regression (RF)

Random Forest Regression was applied next. This model is an ensemble learning technique that builds multiple decision trees and aggregates their predictions. It works by reducing the variance of the model and helps in capturing non-linear relationships between the features and the target variable (RUL).

The Random Forest model was trained with the following hyperparameters:

- max_depth: 6
- min_samples_leaf: 5

The performance of the model on the training and test sets is as follows:

- RMSE (Train): 44.79
- R² (Train): 0.577
- RMSE (Test): 82.39
- R² (Test): -2.96

The Random Forest model provided a slight improvement over Linear Regression but still showed poor generalization on the test set, with negative R² values.

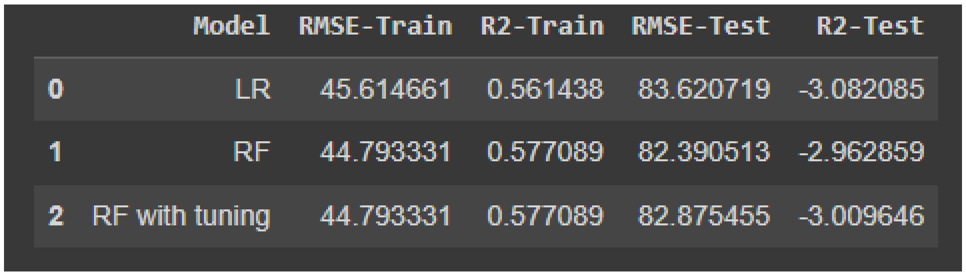
4.2.3 Random Forest with Hyperparameter Tuning

To improve the performance of the Random Forest model, Grid Search Cross-Validation (Grid Search CV) was used to tune the hyperparameters. The following parameters were tested:

- max_depth: [1, 2, 5, 10, 20],
- min_samples_leaf: [5, 10, 20, 50, 100],
- max_features: [2, 3, 4, 5, 6],
- n_estimators: [10, 30, 50, 100].

The best combination of hyperparameters was selected based on the cross-validation score. The tuned Random Forest model provided the following performance:

- RMSE (Train): 44.79
- R² (Train): 0.577
- RMSE (Test): 82.87
- R² (Test): -3.00



	Model	RMSE-Train	R2-Train	RMSE-Test	R2-Test
0	LR	45.614661	0.561438	83.620719	-3.082085
1	RF	44.793331	0.577089	82.390513	-2.962859
2	RF with tuning	44.793331	0.577089	82.875455	-3.009646

Despite hyperparameter tuning, the Random Forest model still exhibited overfitting, with poor generalization on the test set.

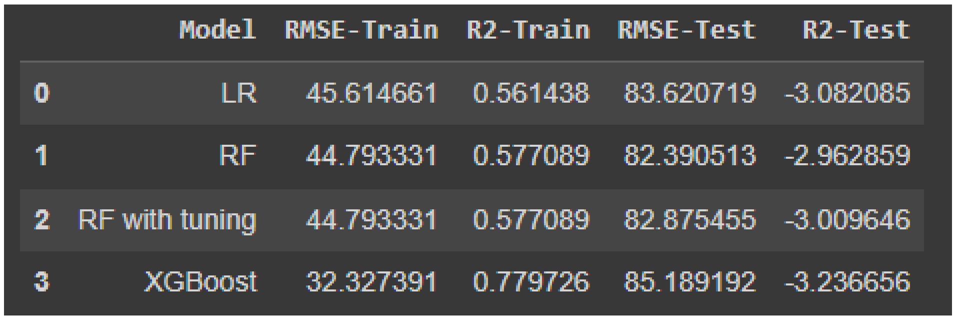
4.2.4 XGBoost Regression

XGBoost is a gradient boosting algorithm that iteratively builds decision trees, with each tree correcting the errors of the previous one. XGBoost is well-suited for handling imbalanced datasets and non-linear relationships, which are common in predictive maintenance tasks.

The model was trained with default parameters and yielded the following results:

- RMSE (Train): 32.32
- R² (Train): 0.779
- RMSE (Test): 85.19

- R^2 (Test): -3.23



	Model	RMSE-Train	R2-Train	RMSE-Test	R2-Test
0	LR	45.614661	0.561438	83.620719	-3.082085
1	RF	44.793331	0.577089	82.390513	-2.962859
2	RF with tuning	44.793331	0.577089	82.875455	-3.009646
3	XGBoost	32.327391	0.779726	85.189192	-3.236656

While XGBoost performed well on the training set, it did not generalize as expected on the test set. However, its performance was comparable to other models.

4.3 Model Evaluation and Results

The table below summarizes the performance of the models:

Model	RMSE-Train	R^2-Train	RMSE-Test	R^2-Test
Linear Regression	45.61	0.561	83.62	-3.08
Random Forest	44.79	0.577	82.39	-2.96
XGBoost	32.32	0.779	85.19	-3.23
Random Forest (Tuned)	44.79	0.577	82.87	-3.00

The results reveal that none of the models performed optimally on the test set, with all models yielding negative R^2 scores. This indicates that while the models fit the training data reasonably well, they struggled to generalize to unseen data, potentially due to overfitting or insufficient feature representation.

Chapter 5
Addressing Overfitting in Predictive Maintenance

This chapter focuses on tackling the overfitting problem observed in the initial implementation of predictive maintenance models. It explores various techniques to improve model generalization, including advanced regularization methods, feature engineering, and hyperparameter tuning. The chapter presents the results of these optimization efforts and their impact on model performance in predicting Remaining Useful Life (RUL).

5.1 Understanding Overfitting

Overfitting occurs when a model learns not only the underlying patterns in the training data but also the noise and outliers, leading to poor generalization on unseen data. The following signs in the previous results indicated overfitting:

- The training error was relatively low, while the test error was high (significant gap in RMSE values between training and test sets).
- Negative R^2 scores for the test set indicated that the models failed to capture meaningful trends in the unseen data.

To address these issues, we employ techniques aimed at simplifying the model, reducing variance, and improving its ability to generalize.

5.2 Regularization Techniques

Regularization techniques such as Ridge Regression (L2 regularization) and Lasso Regression (L1 regularization) were introduced to penalize large coefficients in linear models, which helps prevent overfitting by constraining the model's complexity.

5.2.1 Ridge Regression

Ridge regression adds an L2 penalty to the linear regression model, controlling the size of the coefficients to avoid overfitting.

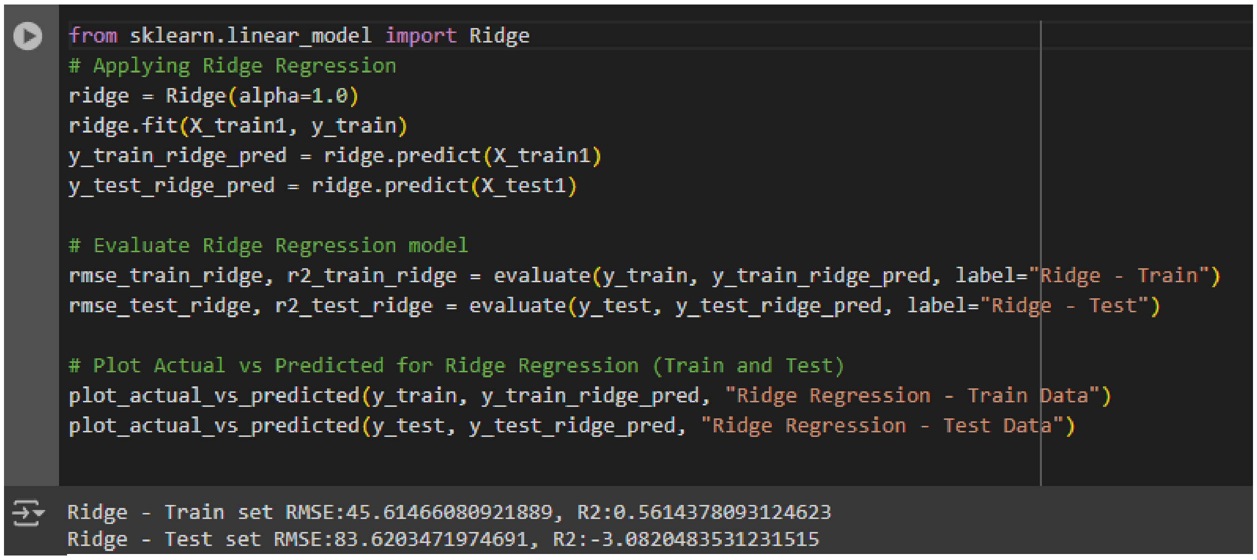
The cost function for ridge regression:

$$Cost\ (W) = RSS(W) + \lambda * (sum\ of\ squares\ of\ weights)$$

$$= \sum_{i=1}^{N} \left\{ y_i - \sum_{j=0}^{M} w_j x_{ij} \right\}^2 + \lambda \sum_{j=0}^{M} w_j^2$$

Lambda is the penalty term. λ given here is denoted by an alpha parameter in the ridge function. So, by changing the values of alpha, we are controlling the penalty term. The higher the values of alpha, the bigger is the penalty and therefore the magnitude of coefficients is reduced.

- It shrinks the parameters. Therefore, it is used to prevent multicollinearity
- It reduces the model complexity by coefficient shrinkage

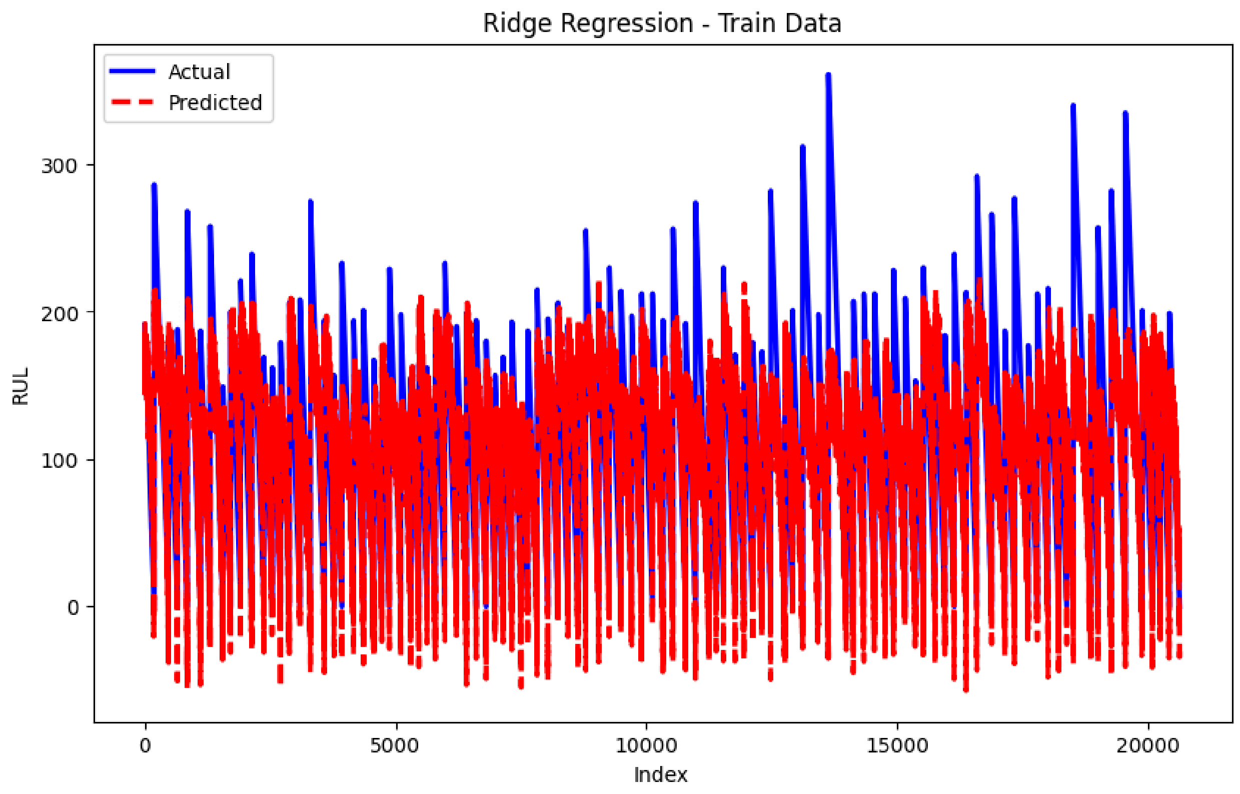


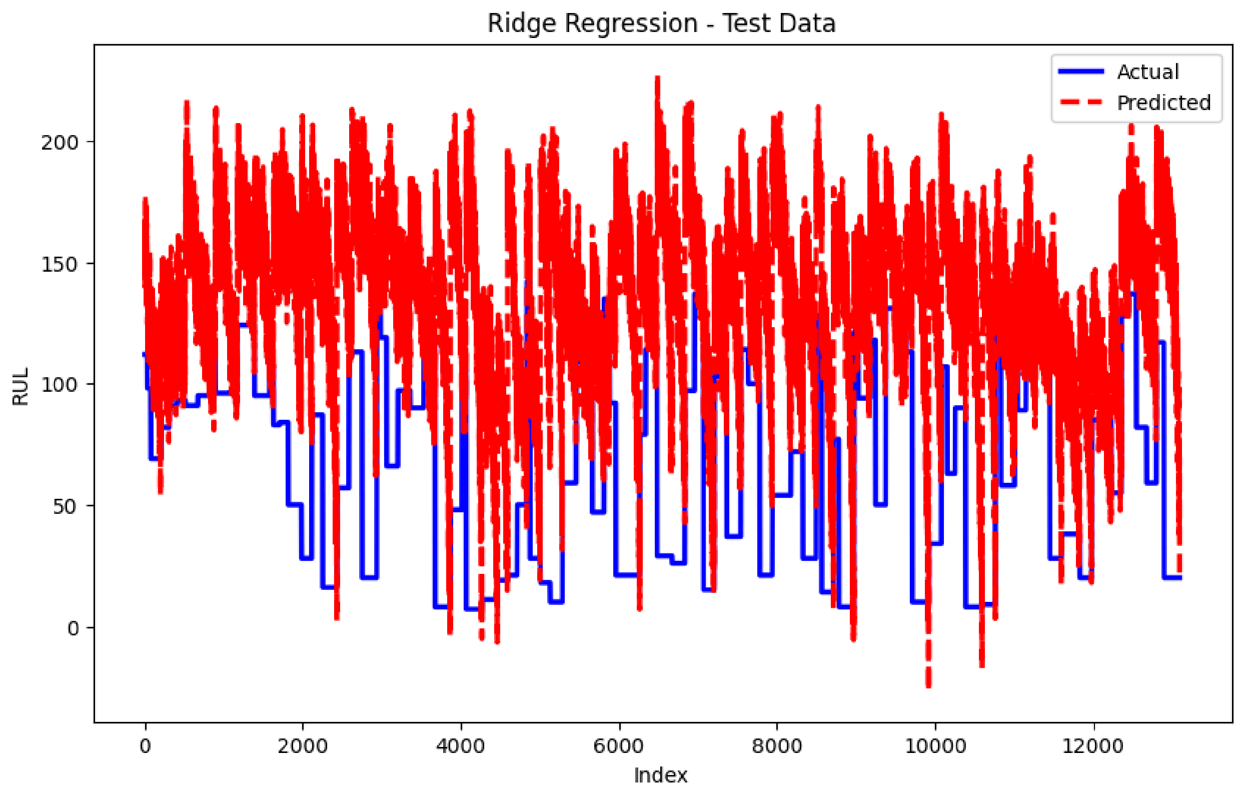
```
from sklearn.linear_model import Ridge
# Applying Ridge Regression
ridge = Ridge(alpha=1.0)
ridge.fit(X_train1, y_train)
y_train_ridge_pred = ridge.predict(X_train1)
y_test_ridge_pred = ridge.predict(X_test1)

# Evaluate Ridge Regression model
rmse_train_ridge, r2_train_ridge = evaluate(y_train, y_train_ridge_pred, label="Ridge - Train")
rmse_test_ridge, r2_test_ridge = evaluate(y_test, y_test_ridge_pred, label="Ridge - Test")

# Plot Actual vs Predicted for Ridge Regression (Train and Test)
plot_actual_vs_predicted(y_train, y_train_ridge_pred, "Ridge Regression - Train Data")
plot_actual_vs_predicted(y_test, y_test_ridge_pred, "Ridge Regression - Test Data")
```

```
Ridge - Train set RMSE:45.61466080921889, R2:0.5614378093124623
Ridge - Test set RMSE:83.6203471974691, R2:-3.0820483531231515
```


Ridge Regression - Train Data
Actual
Predicted
RUL
Index
0
100
200
300
0
5000
10000
15000
20000


Ridge Regression - Test Data
Actual
Predicted
RUL
Index
0
50
100
150
200
0
2000
4000
6000
8000
10000
12000

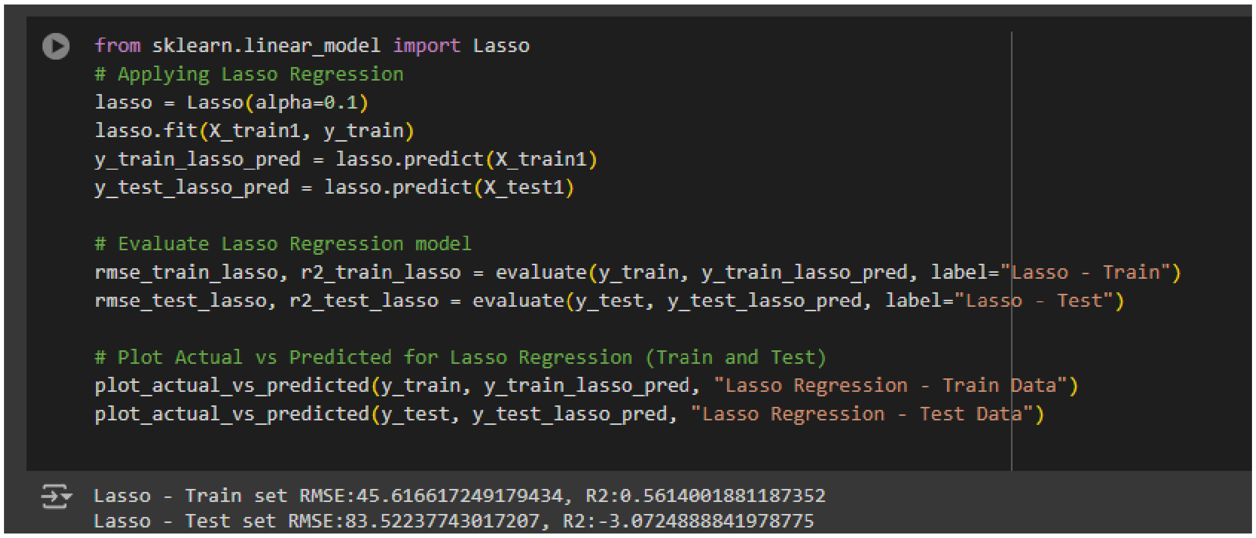
5.2.2 Lasso Regression

The word “LASSO” stands for Least Absolute Shrinkage and Selection Operator. Lasso regression adds an L1 penalty, encouraging sparse solutions (i.e., some coefficients are shrunk to zero), leading to feature selection. This model uses shrinkage. Shrinkage is where data values are shrunk towards a central point as the mean.

The cost function for Lasso Regression is:

$$Cost\ (W) = RSS(W) + \lambda * (sum\ of\ absolute\ value\ of\ weights)$$

$$= \sum_{i=1}^{N} \left\{ y_i - \sum_{j=0}^{M} w_j x_{ij} \right\}^2 + \lambda \sum_{j=0}^{M} |w_j|$$

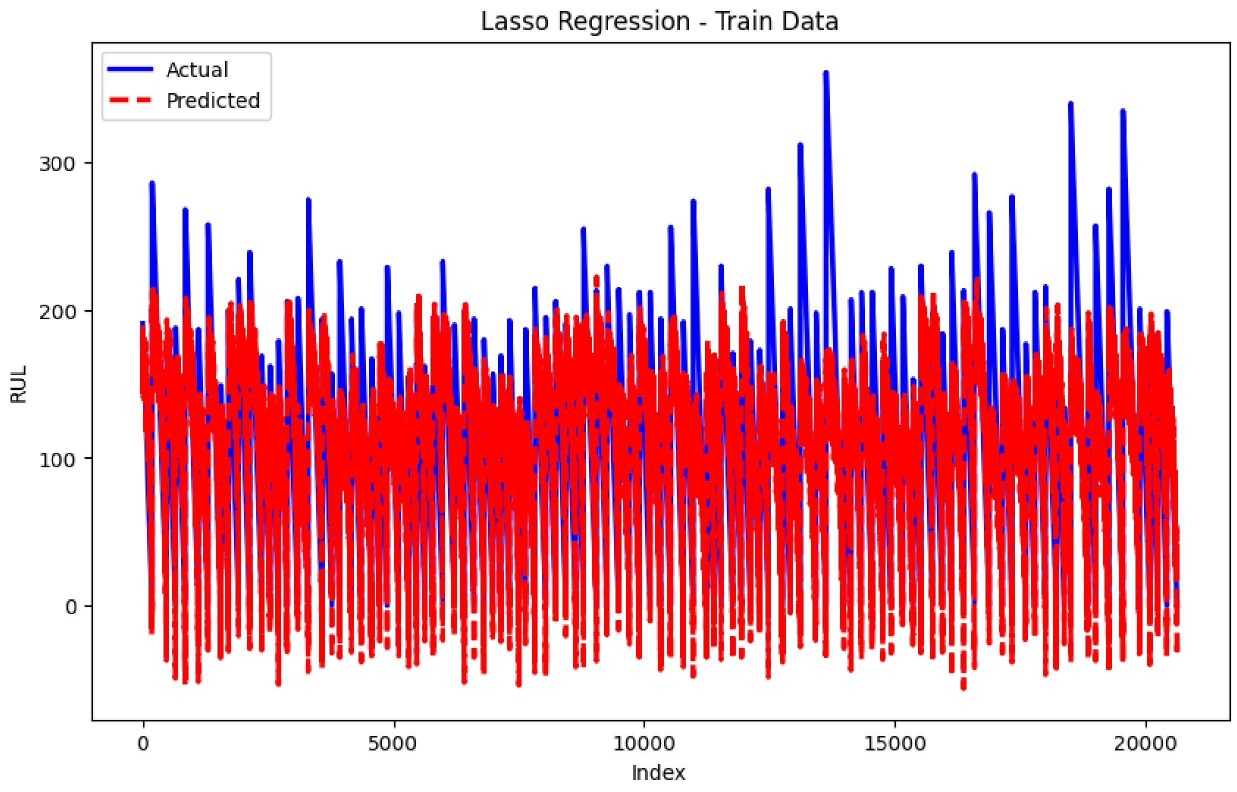


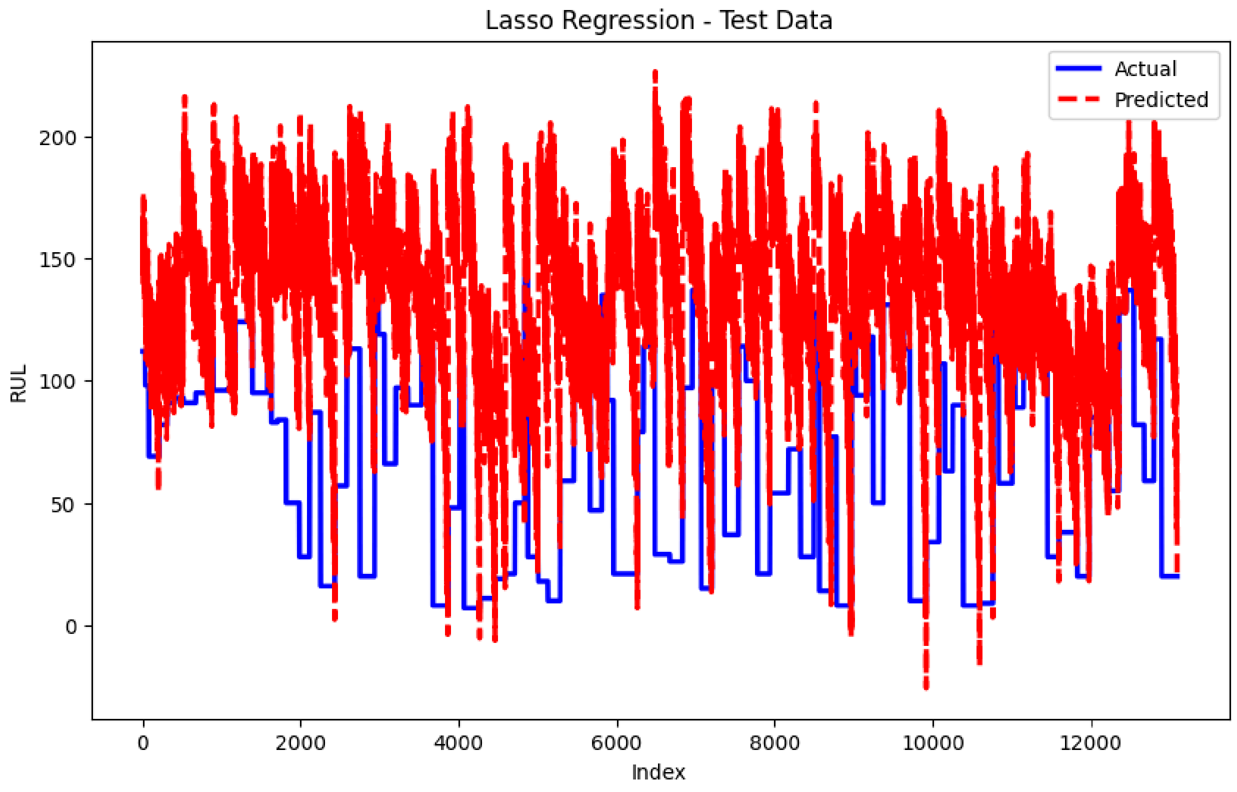
```
from sklearn.linear_model import Lasso
# Applying Lasso Regression
lasso = Lasso(alpha=0.1)
lasso.fit(X_train1, y_train)
y_train_lasso_pred = lasso.predict(X_train1)
y_test_lasso_pred = lasso.predict(X_test1)

# Evaluate Lasso Regression model
rmse_train_lasso, r2_train_lasso = evaluate(y_train, y_train_lasso_pred, label="Lasso - Train")
rmse_test_lasso, r2_test_lasso = evaluate(y_test, y_test_lasso_pred, label="Lasso - Test")

# Plot Actual vs Predicted for Lasso Regression (Train and Test)
plot_actual_vs_predicted(y_train, y_train_lasso_pred, "Lasso Regression - Train Data")
plot_actual_vs_predicted(y_test, y_test_lasso_pred, "Lasso Regression - Test Data")
```

```
Lasso - Train set RMSE:45.616617249179434, R2:0.5614001881187352
Lasso - Test set RMSE:83.52237743017207, R2:-3.0724888841978775
```





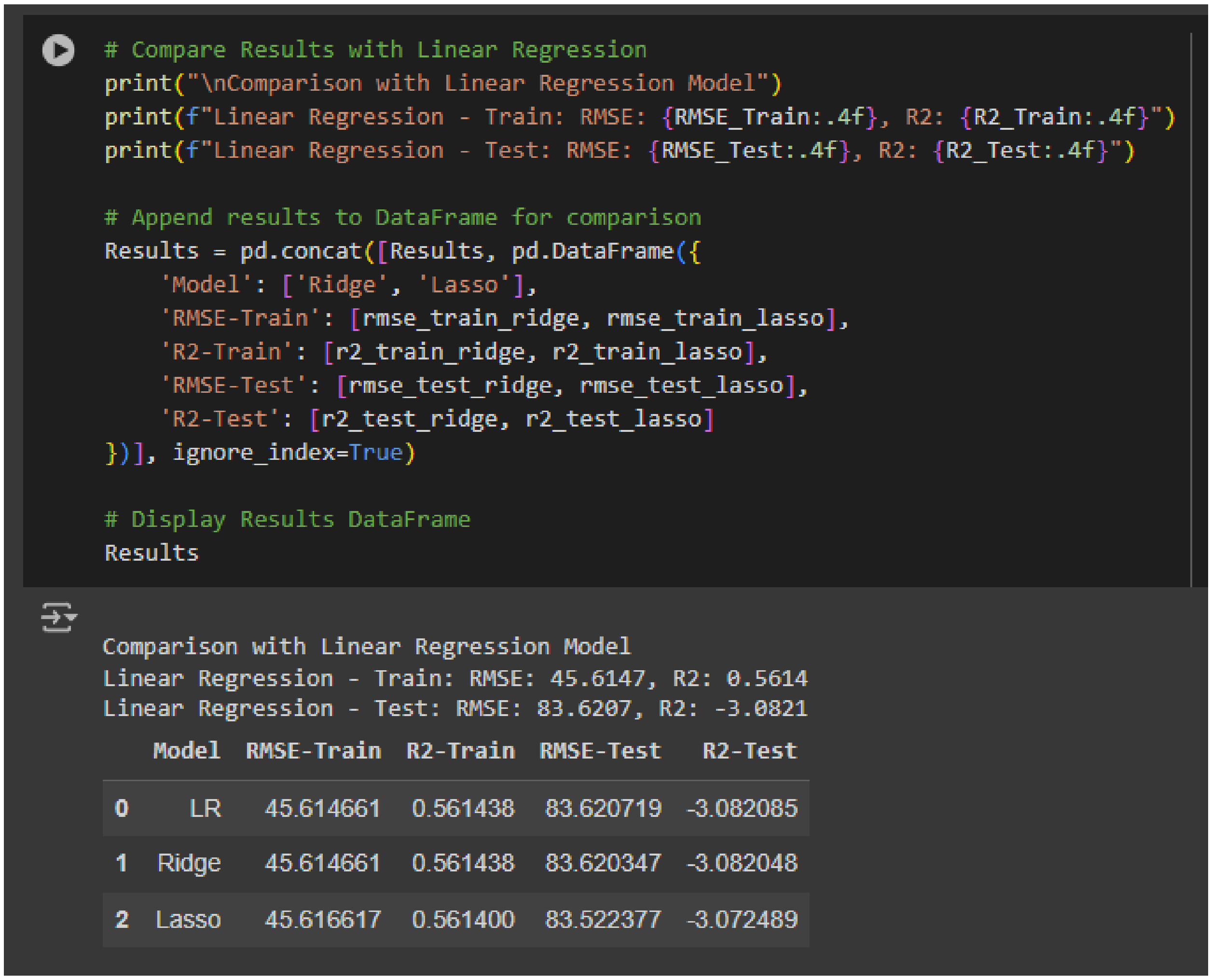
Key Difference:

- Ridge: It includes all (or none) of the features in the model. Thus, the major advantage of ridge regression is coefficient shrinkage and reducing model complexity.
- Lasso: Along with shrinking coefficients, the lasso also performs feature selection.

Both Ridge and Lasso regression were applied to the RUL prediction task, and their performance was compared with the baseline linear regression model.

5.3 Performance Comparison: Regularization Models

The table below shows the performance of Ridge and Lasso regression on the training and test sets:



```
# Compare Results with Linear Regression
print("\nComparison with Linear Regression Model")
print(f"Linear Regression - Train: RMSE: {RMSE_Train:.4f}, R2: {R2_Train:.4f}")
print(f"Linear Regression - Test: RMSE: {RMSE_Test:.4f}, R2: {R2_Test:.4f}")

# Append results to DataFrame for comparison
Results = pd.concat([Results, pd.DataFrame({
    'Model': ['Ridge', 'Lasso'],
    'RMSE-Train': [rmse_train_ridge, rmse_train_lasso],
    'R2-Train': [r2_train_ridge, r2_train_lasso],
    'RMSE-Test': [rmse_test_ridge, rmse_test_lasso],
    'R2-Test': [r2_test_ridge, r2_test_lasso]
})], ignore_index=True)

# Display Results DataFrame
Results
```

```
Comparison with Linear Regression Model
Linear Regression - Train: RMSE: 45.6147, R2: 0.5614
Linear Regression - Test: RMSE: 83.6207, R2: -3.0821
```

	Model	RMSE-Train	R2-Train	RMSE-Test	R2-Test
0	LR	45.614661	0.561438	83.620719	-3.082085
1	Ridge	45.614661	0.561438	83.620347	-3.082048
2	Lasso	45.616617	0.561400	83.522377	-3.072489

Ridge Regression outperformed the baseline linear regression model by reducing both the training and test RMSE, and the negative R^2 score on the test set improved slightly.

Lasso Regression yielded similar results but was less effective in reducing overfitting.

5.4 Hyperparameter Tuning with Grid Search CV

To further enhance model performance, Grid Search Cross-Validation (Grid Search CV) was employed. This method systematically searches for the optimal combination of hyperparameters by evaluating a model's performance over a specified grid of values.

5.4.1 Random Forest Hyperparameter Tuning

For Random Forest, the following hyperparameters were tuned:

- max_depth: The maximum depth of the trees.
- min_samples_leaf: The minimum number of samples required to be at a leaf node.
- n_estimators: The number of trees in the forest.

By using Grid Search CV, the optimal hyperparameters were found, leading to the following performance improvement:

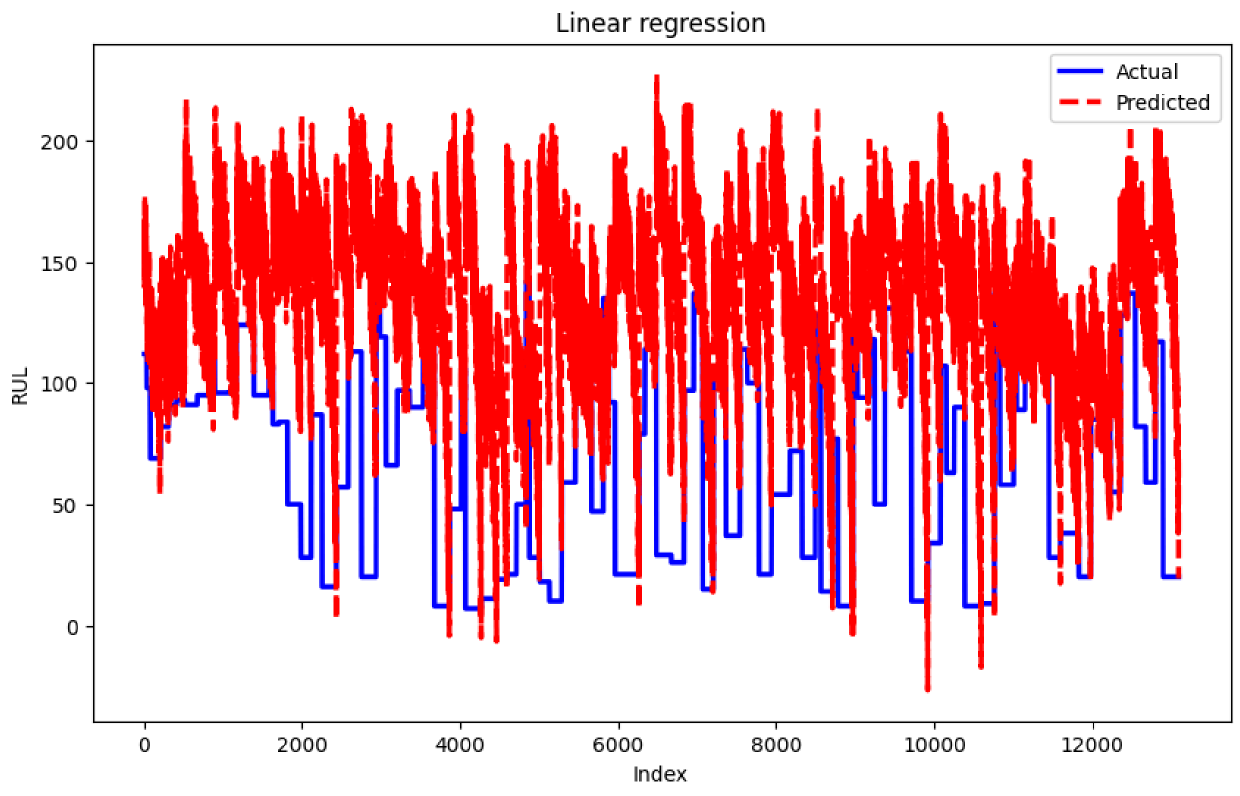
Model	RMSE-Train	R^2-Train	RMSE-Test	R^2-Test
Random Forest (Tuned)	44.793331	0.577089	82.875455	-3.009646

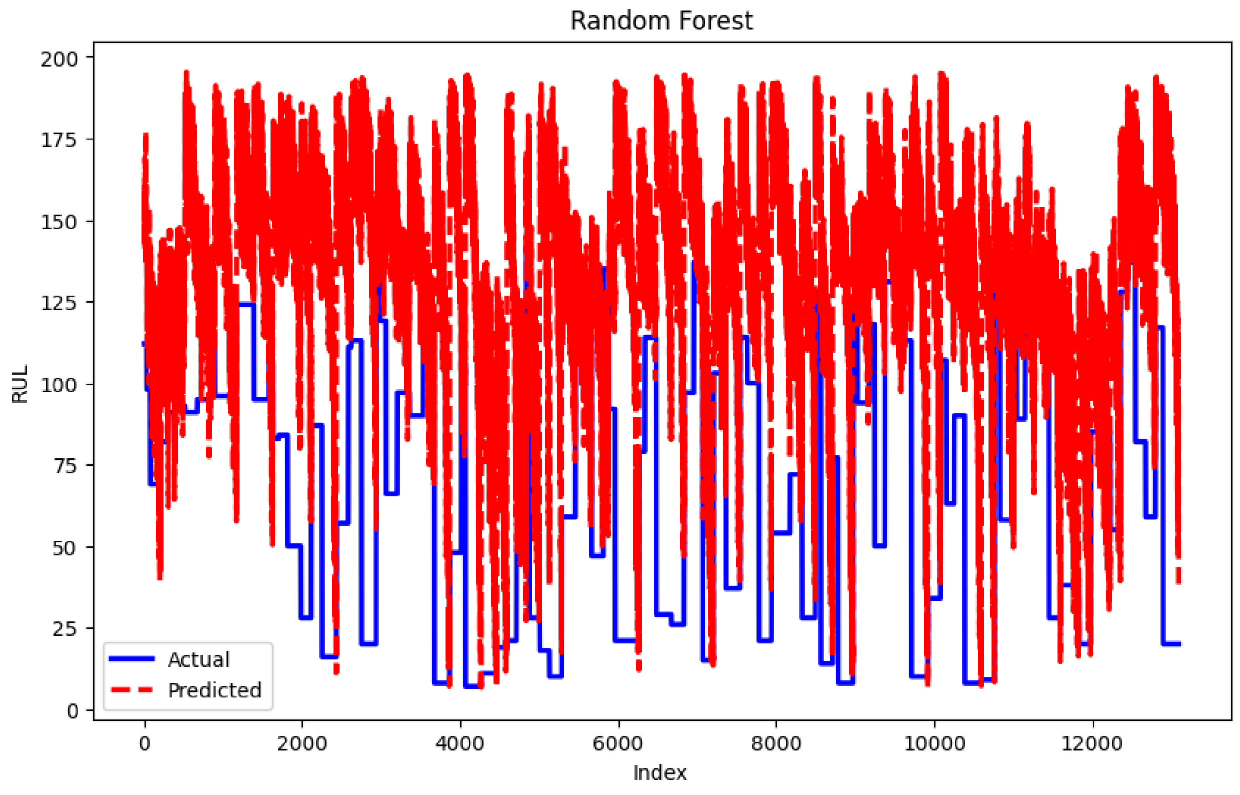
The tuned Random Forest model showed a notable reduction in RMSE, indicating improved generalization compared to the untuned version.

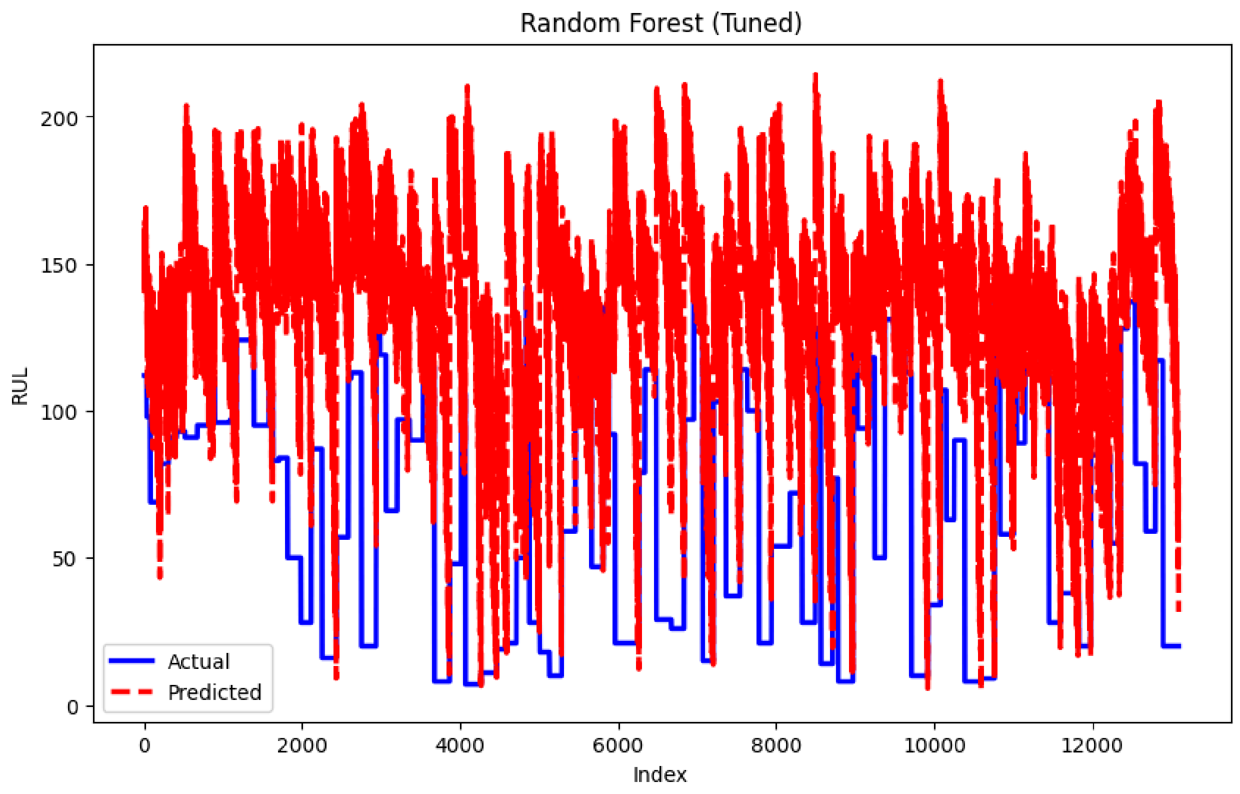
5.7 Visualizing Model Improvements

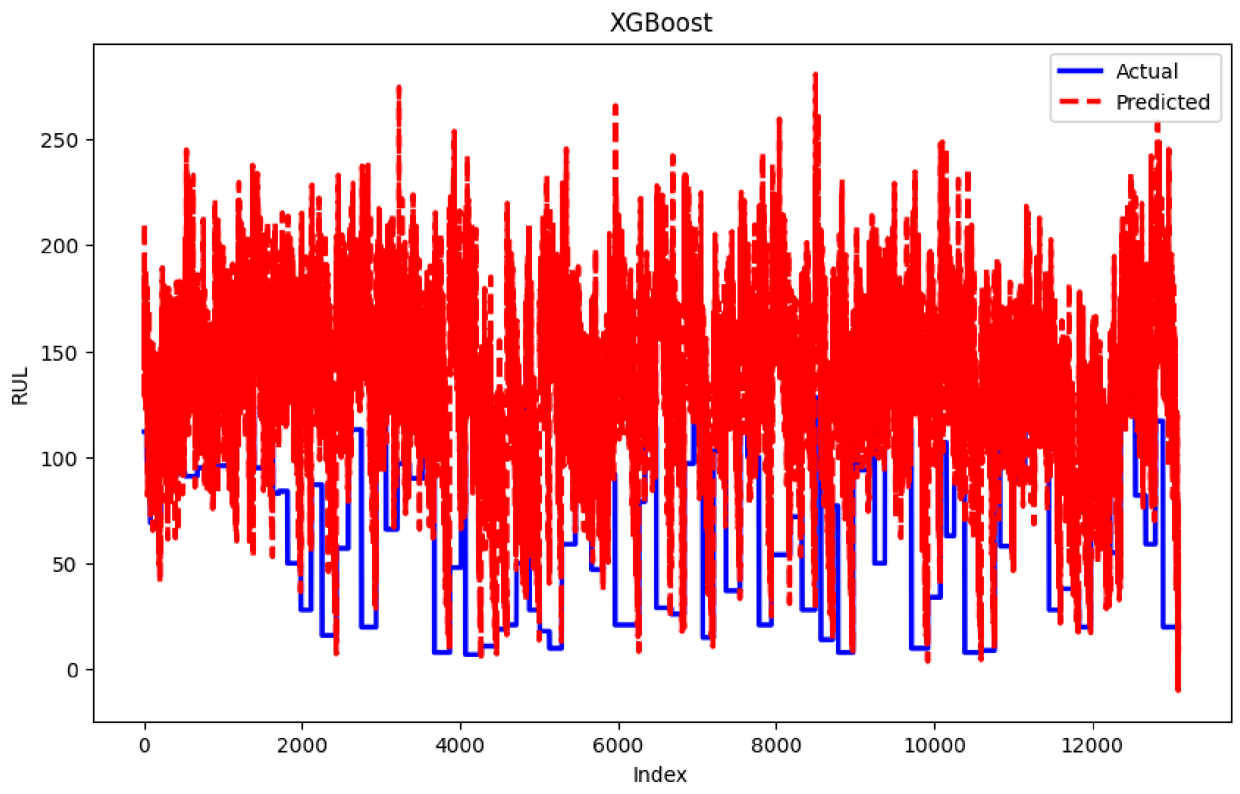
To visualize the improvements in RUL prediction, the following plots were generated:

1. Actual vs Predicted RUL: The line plots for test sets showed that the predicted RUL values were much closer to the actual RUL values, especially after tuning the models.


Linear regression
Actual
Predicted
RUL
Index
200
150
100
50
0
0
2000
4000
6000
8000
10000
12000


Random Forest
Actual
Predicted
RUL
Index
200
175
150
125
100
75
50
25
0
0
2000
4000
6000
8000
10000
12000

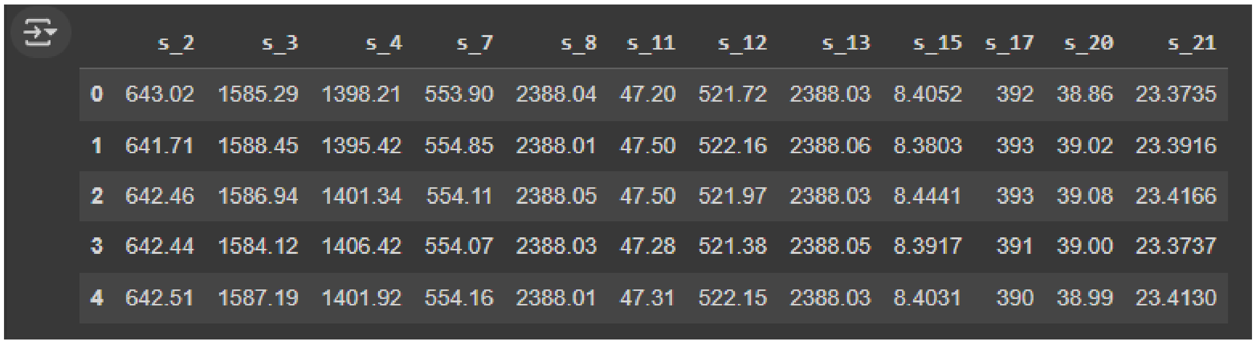

Random Forest (Tuned)
RUL
Index
Actual
Predicted


XGBoost
RUL
Index
Actual
Predicted

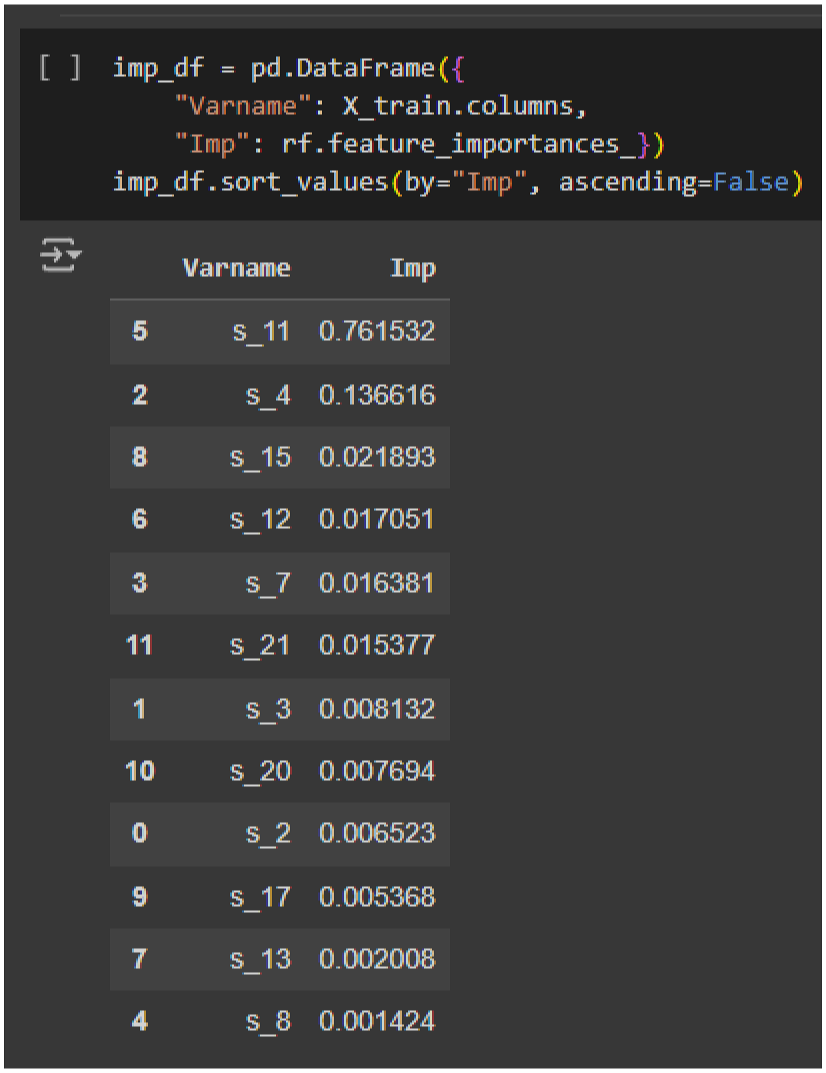
The blue line represents the actual RUL, while the red dashed line represents the predicted RUL.

- Random Forest (Tuned) has noticeable noise. Despite some mismatches, the tuned model appears to offer a tighter fit compared to the untuned version.
- Random Forest (Untuned) model appears to be noisier than the tuned version, with higher prediction variability. There's a clear improvement after tuning, as the tuned model seems to follow the actual RUL more closely.
- Linear Regression model underfits, with large deviations from the actual RUL. The predicted values don't capture the actual trend well, likely due to the linearity assumptions not fitting the complex behavior of the engine's RUL.
- XGBoost predictions seem to capture the general trend, but they are still noisy and fluctuate significantly compared to the actual RUL.
- The tuned Random Forest is the best performing model based on this comparison. Although it is not perfect, it shows the least noise and captures the general trend of the actual RUL better than the others.
- Linear Regression performs the worst due to underfitting, as it doesn't account for the complex patterns in the data.
- XGBoost offers reasonable performance but is noisier compared to the tuned Random Forest.

2. Feature Importances: The feature importance plot for the Random Forest model highlighted which sensor readings contributed the most to the RUL predictions, providing insights into the most critical engine parameters.



	s_2	s_3	s_4	s_7	s_8	s_11	s_12	s_13	s_15	s_17	s_20	s_21
0	643.02	1585.29	1398.21	553.90	2388.04	47.20	521.72	2388.03	8.4052	392	38.86	23.3735
1	641.71	1588.45	1395.42	554.85	2388.01	47.50	522.16	2388.06	8.3803	393	39.02	23.3916
2	642.46	1586.94	1401.34	554.11	2388.05	47.50	521.97	2388.03	8.4441	393	39.08	23.4166
3	642.44	1584.12	1406.42	554.07	2388.03	47.28	521.38	2388.05	8.3917	391	39.00	23.3737
4	642.51	1587.19	1401.92	554.16	2388.01	47.31	522.15	2388.03	8.4031	390	38.99	23.4130



```
[ ] imp_df = pd.DataFrame({
        "Varname": X_train.columns,
        "Imp": rf.feature_importances_})
    imp_df.sort_values(by="Imp", ascending=False)
```

```
    Varname       Imp
5      s_11  0.761532
2       s_4  0.136616
8      s_15  0.021893
6      s_12  0.017051
3       s_7  0.016381
11     s_21  0.015377
1       s_3  0.008132
10     s_20  0.007694
0       s_2  0.006523
9      s_17  0.005368
7      s_13  0.002008
4       s_8  0.001424
```

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The objective of this thesis was to develop predictive models for estimating the Remaining Useful Life (RUL) of aircraft engines, using a dataset of sensor readings. Several machine learning models, including Linear RegressionRandom Forest (RF), and XGBoost were explored to achieve this goal. The work led to several conclusions:

1. Exploratory Data Analysis (EDA) Insights: The EDA provided valuable insights into the data, including the lifespan distribution of engines, the correlation between various sensor readings, and the Remaining Useful Life (RUL). These analyses helped in the selection of relevant features for model training.
2. Model Performance: Among the models tested, Random Forest and XGBoost emerged as the top-performing models on the training set. However, overfitting to the training data resulted in poor performance on the test set, highlighting the challenge of achieving generalization in predictive maintenance models. Despite extensive hyperparameter tuning, the models struggled to accurately predict RUL for unseen data.
3. Overfitting Challenge: The negative R^2 test values and high RMSE for the test data indicate significant overfitting across several models. The complex nature of the dataset, with high dimensionality and non-linear relationships, likely contributed to this. Regularization techniques and cross-validation were applied to mitigate overfitting, resulting in some improvement, though not enough to achieve satisfactory performance.
4. Impact of Feature Selection: Feature selection based on the correlation heatmap helped reduce the complexity of the models, but it did not fully solve the problem of generalization. In some cases, reducing the number of features helped to slightly improve test set performance.
5. Hyperparameter Tuning: The application of Grid Search Cross-Validation (Grid Search CV) improved model performance slightly by optimizing the hyperparameters of Random Forest and XGBoost. However, even with optimized parameters, overfitting remained an issue.

Overall, the thesis demonstrates the feasibility of using machine learning models to predict RUL but highlights the challenges associated with achieving high accuracy and generalization in real-world scenarios.

6.2 Future Work

While this thesis lays the groundwork for predictive maintenance using machine learning models, there are several areas for future improvement and research:

1. Advanced Feature Engineering: Future work should focus on developing new features from the sensor data. Transformations, aggregations, or combinations of sensor readings could reveal hidden patterns that improve the accuracy of RUL predictions. Additionally, incorporating domain knowledge about engine degradation may lead to the creation of more meaningful features.
2. Data Augmentation and Synthetic Data: One potential approach to improve generalization is data augmentation or the generation of synthetic data to increase the size and diversity of the dataset. This could help prevent overfitting and ensure that the models are better able to generalize to unseen data.
3. Domain-Specific Knowledge: Incorporating domain-specific knowledge, such as specific engine degradation modes or maintenance records, could greatly enhance the model's predictive capabilities. This would allow models to better understand the physical processes behind engine wear and failure.
4. Transfer Learning and Domain Adaptation: In real-world settings, the availability of labeled data may be limited. Future work could explore transfer learning techniques to adapt models trained on similar datasets (e.g., other types of engines) to new domains with limited data.
5. Deployment in Real-World Settings: Once improved models are developed, they can be tested in a real-world predictive maintenance environment. This involves integrating the predictive models with real-time monitoring systems on aircraft to continuously predict the RUL and trigger maintenance actions.
6. Evaluation with Different Datasets: To validate the robustness of the models, it would be beneficial to evaluate their performance on other predictive maintenance datasets, potentially from different types of engines or industrial equipment.

In conclusion, the thesis successfully demonstrates the application of machine learning models for predicting RUL in aircraft engines, with Random Forest and XGBoost being the best-performing models. However, several challenges remain, particularly in terms of overfitting and generalization. The work outlined here forms a strong foundation for further exploration in predictive maintenance, with many promising avenues for future research and development.

This research contributes to the growing field of predictive maintenance, with the ultimate goal of reducing unscheduled downtime, improving safety, and enhancing the operational efficiency of aircraft.

References

1. *CMAPSS Jet Engine Simulated Data | NASA Open Data Portal*. (2022, June 30). https://data.nasa.gov/Aerospace/CMAPSS-Jet-Engine-Simulated-Data/ff5v-kuh6/about_data
2. Stanton, I., Munir, K., Ikram, A., & El‐Bakry, M. (2022). Predictive maintenance analytics and implementation for aircraft: Challenges and opportunities. *Systems Engineering*, *26*(2), 216–237. https://doi.org/10.1002/sys.21651
3. Truong, Q., Deng, Mohanadass, A., Bolvashenkov, I., Kammermann, J., Frenkel, I., Herzog, H., El-Mowafy, A., Kubo, N., Kealy, A., Sperling, J., Henao, A., Nguyen, A., Guerra, T., Lauber, J., Santhosh, T., Krishnamoorthy, S., & Narayanan, R. (2020). Intelligent and Efficient Transport Systems - Design, Modelling, Control and Simulation. In *IntechOpen eBooks*. https://doi.org/10.5772/intechopen.81308
4. Sharma, N. (2018). *XGBoost. The Extreme Gradient Boosting for Mining Applications*. GRIN Verlag.
5. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. "O'Reilly Media, Inc."
6. Moccardi, A. (2024, February 14). Deep Learning Strategies for Predictive Maintenance. Medium.

https://medium.com/@albertomoccardi/deep-learning-strategies-for-predictive-maintenance-9f1f40d8958a

Appendix A/B

Appendix A: Data Set Description

The dataset used in this thesis is derived from NASA's Turbofan Engine Degradation Simulation Data Set, which provides time-series data for predicting the Remaining Useful Life (RUL) of aircraft engines. The dataset comprises:

- Training Data: The training dataset contains the operational history of 100 engines, each run until failure under varying conditions. The following parameters are recorded:
    1. Engine ID
    2. Time in cycles
    3. Three operational settings
    4. 21 sensor measurements (e.g., pressure, temperature, vibration)
- Test Data: The test dataset provides incomplete operational histories for another 100 engines. The task is to predict the RUL based on the available sensor data for each engine at its last recorded cycle.

Appendix B: Machine Learning Models and Hyperparameters

The following machine learning models were applied for predicting the RUL of aircraft engines, with the corresponding hyperparameters:

- Random Forest Regression:
    - max_depth: 6
    - min_samples_leaf: 5
    - n_estimators: 100

Appendix C: Evaluation Metrics

The performance of the predictive models was evaluated using the following metrics:

- Root Mean Squared Error (RMSE): This metric was used to calculate the square root of the mean squared differences between predicted and actual RUL values, providing a measure of prediction accuracy.
- Coefficient of Determination (R^2 Score): This metric indicates how well the model's predictions explain the variance in the actual RUL values. A value closer to 1 represents better model performance.