Nasa AirCraft Turbofan Jet Engine – Predictive Maintenance

Written by	Ayush Gandhi
Document Version	0.1

Document Version Control

Reviews:

Version	Date	Author	Comments

Approval Status:

Version	Review Date	Review by	Approved by	Comments

Predictive_Maintainence

1. Introduction

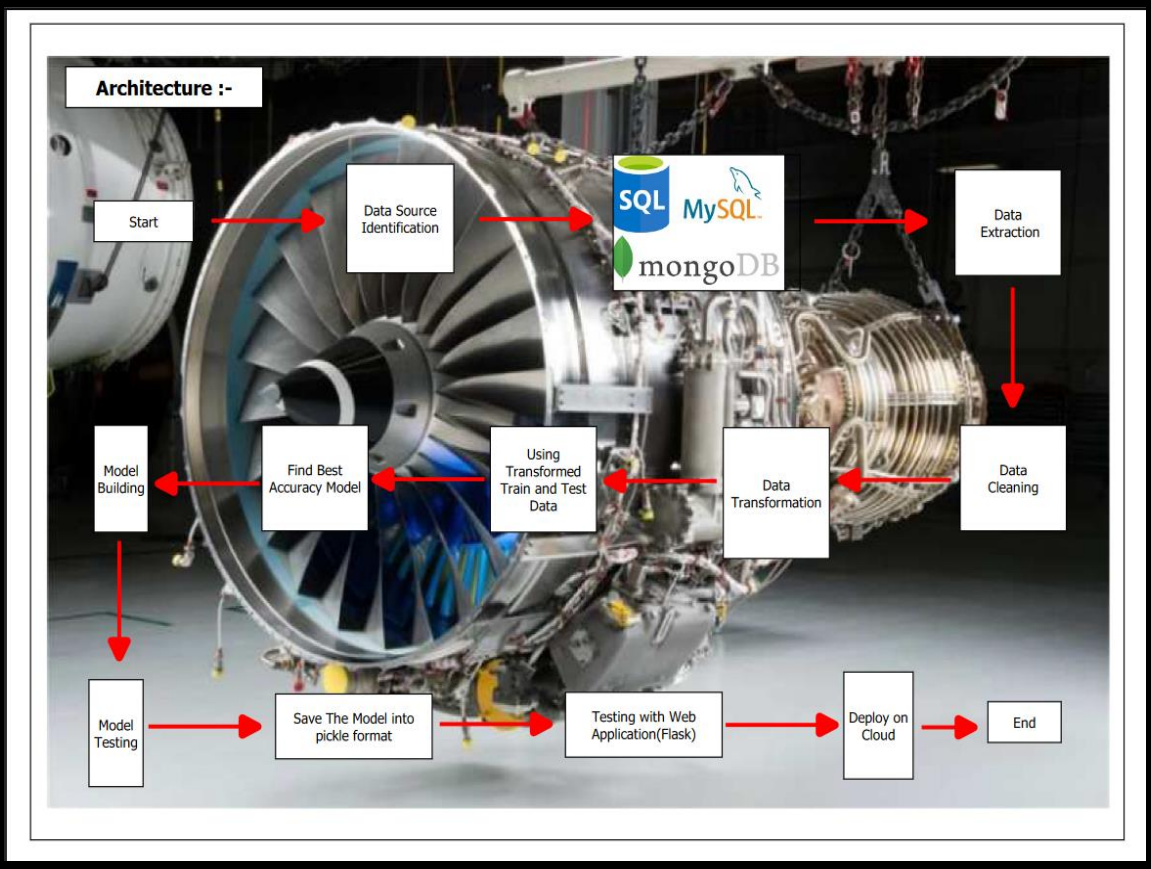
1.1 What is Low-Level Design Document?

The goal of LLD or a low-level design document (LLDD) is to give the internal logical design of the actual program code for Mushroom Classifier. LLD describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer can directly code the program from the document

1.2 Scope

Low-level design (LLD) is a component-level design process that follows a stepby-step refinement process. This process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work.

2. Architecture



3. Architecture Description

3.1 Data Description

Predictive Maintenance (PdM) is a proactive maintenance approach that uses data analysis and machine learning algorithms to predict when maintenance should be performed on a machine or system. By analyzing real-time data from sensors and other sources, PdM can help identify potential failures before they occur, minimize downtime, and reduce maintenance costs. The data that we have considered for predictive maintenance is online and it is available on kaggle. We are going to predict the Remaining useful life of NASA's turbofan engine using various machine learning models since PdM has become increasingly important in the aviation industry

3.2 Data ingestion

Data ingestion centralizes data so it's available for processing whereas data preparation cleans, transforms, and/or shapes data to make it ready for analysis. So, we did Exploratory Data Analysis and in this dataset there is no duplicates, missing value but there have lots of outliers or constant features. We remove all the constant features and all "setting" and " sensors" feature are in float value so we scaled the data then we visualize the data on histplot and we also correlate the refine features with predicted value ("RUL) through heatmap we visualized this.
In Data Ingestion, we extracting the train and test dataset through the MySQL workbench.

3.3 Feature engineering

1. Feature Selection: In Dataset have 26 input features, excluding the target feature. We performed a careful selection process to identify the most relevant features for our model.

2. Removal of Constant Sensor Feature: We began by removing constant sensor features that do not provide meaningful information for predicting RUL (Remaining Useful Life). And It helps us improve model efficiency.

3. Correlation Analysis: To better understand the relationship between our remaining features and RUL, we conducted a correlation analysis. This allowed us to gain valuable insights into the predictive power of each feature.

4. Identifying Irrelevant Features: Through this analysis, we discovered that the 'op_setting' feature did not exhibit a significant relationship with the dependent variable RUL. As a result, we made the informed decision to exclude it from our feature set.

5. Leveraging Highly Correlated Features: On the other hand, we found that the 'fuel flow ratio' feature displayed a strong correlation with RUL compared to 'bleed enthalpy.' Recognizing the importance of capturing this relationship, we retained 'fuel flow ratio' as it is likely to be a valuable predictor for our model.

In essence, our Feature Engineering process involved a systematic and data-driven approach to select the most relevant features while discarding those that did not contribute significantly to the prediction of RUL. By making these informed choices, we aimed to build a more robust and accurate predictive model for estimating the remaining useful life of the system.

3.4 Data Pre-processing and Data Transformation

In the data pre-processing phase, we meticulously handled various aspects, including addressing constant features and outliers, ensuring data balance, and calculating the Remaining Useful Life (RUL), which serves as our predictive feature. To facilitate data transformation, we constructed a robust pipeline and saved the object using the pickle format. This transformation process involved converting the pre-processor object into data and subsequently converting it into a numpy array for both the training and testing datasets. Furthermore, we took the step of normalizing the dataset to enhance model performance.

To optimize the performance of our predictive maintenance system, we leveraged the power of grid search cross-validation (Grid Search CV) for hyperparameter tuning. Additionally, we adopted the RobustScaler model and Principal Component Analysis (PCA) for dimensionality reduction and feature scaling, further enhancing the model's robustness.

Once the data was adequately prepared and transformed, we proceeded to the training phase, employing a variety of classifier models, including Logistic Regression, Decision Trees, and Cross-Validation, among others. Following model training, we rigorously assessed their performance by comparing accuracy scores

on the testing dataset. This comprehensive evaluation will provide insights into the effectiveness of our predictive maintenance system across different classifier models, enabling us to make informed decisions and optimizations for the project.

3.5 Predictive Models:

In our pursuit of building an accurate predictive model, we meticulously assessed various machine learning algorithms. We curated a set of diverse models, including Lasso, Linear Regression, Support Vector Machine (SVM), K-Neighbors Regressor, RandomForest Regressor, Gradient Boosting Regressor, and XGBoost Regressor, each with its unique strengths and characteristics. To ensure that our features were on a level playing field, we employed standard scaling, a crucial preprocessing step. Among these formidable contenders, it was the Support Vector Machine (SVM) that emerged as the standout performer, consistently delivering the highest accuracy. This selection underscores the significance of our model choice, as it demonstrates SVM's capability to effectively capture complex patterns in our data and provide us with superior predictive power.

3.6 Model Evaluation

Model evaluation is a critical step in the machine learning workflow, as it provides insights into how well a model is likely to perform in practice. It guides decisions about whether to deploy a model, refine it, or explore alternative approaches. Ultimately, the effectiveness of model evaluation determines the model's practical utility and impact on solving real-world problems.
Once, model is tested, model with highest accuracy score will be store in database & will predict output based on user given data.

1. Performance Metrics: In this, we used common metrics includes R2 score, mean squared error, mean absolute error.
2. Train-Test Split : To assess a model's performance ,the dataset is often divided into two subset: a training and testing set. The model is trained on training set and its performance is evaluated on testing set.

	Model	MAE	RMSE	R2
0	Lasso	33.775373	43.271669	0.461694
1	LinearRegression	33.922029	43.253501	0.462146
2	SupportVectorMachine	31.393672	41.888889	0.495548
3	KNeighborsRegressor	35.359056	45.976951	0.392282
4	RandomForestRegressor	40.069879	52.835141	0.197458
5	GradientBoostingRegressor	35.723366	47.193165	0.359705
6	XGBRegressor	46.518579	60.226749	-0.042799

3. Hyper-parameter Tuning with Pre-processing: Model Evaluation often involves hyper parameter tuning, where different combinations of hyper-parameters are tested to find the optimal settings that result in the best performance. So, we used in our model Hyper-parameter tuning in SVM.

4. Comparative Analysis: we used multiple models like linear regression, Support Vector Machine, KNN Regressor, Random Forest Regressor, Gradient Boosting Regressor, XGB Regressor. In among of these SVM gives the best accuracy with using of hyper-parameter tuning.

```
# Define the SVR hyperparameter search space
param_dist = {
    'C': [3, 5],
    'epsilon': [0.1, 0.2],
    'kernel': ['rbf'],
    'degree': [3 , 5],
    'gamma': ['auto'],
    'coef0': [0.0, 0.1],
    'shrinking': [True],
    'tol': [0.001, 0.01],
    'cache_size': [200, 500]
}
```

	Model	MAE	RMSE	R2
0	Lasso	12.183683	15.168114	0.417514
1	LinearRegression	11.795560	14.784870	0.446577
2	SupportVectorMachine	5.558247	10.882363	0.700174
3	KNeighborsRegressor	6.525596	11.855887	0.644131
4	RandomForestRegressor	6.724953	11.622220	0.658020
5	GradientBoostingRegressor	6.988650	11.136907	0.685984
6	XGBRegressor	7.707941	12.106965	0.628898

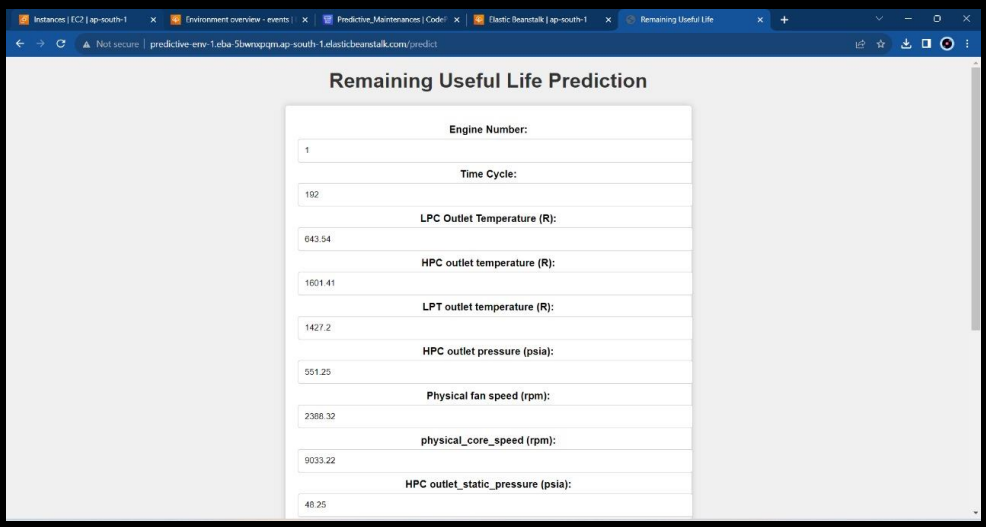
Model evaluation is a critical step in the machine learning workflow, as it provides insights into how well a model is likely to perform in practice. It guides decisions about whether to deploy a model is likely to perform in practice. It guides decisions about whether to deploy a model, refine it, or explore alternative approaches . Ultimately the effectiveness of model evaluation determines the model's practical utility and impact on solving real-world problems.

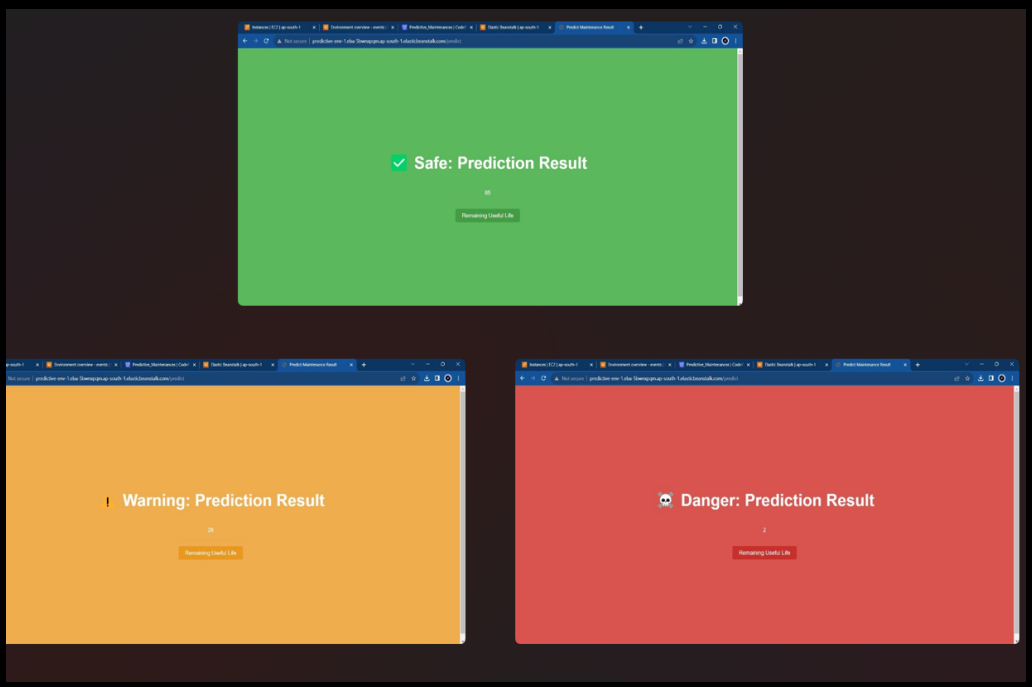
3.7 Data from User

As a trained model, it is capable of predicting the RUL of Engine as a user input sensor data.

3.8 Model Deployment:

We deploy created model on AWS server through bean elastic . Here , the created model run successfully and as we check it is giving accuracy as per the model.





4. Unit Test Cases

Test Case Description	Pre-Requisite	Expected Result
Verify whether the application URL is accessible to the user	1. Application URL should be defined	Application URL should be accessible to the user
Verify whether the application loads completely for the user when the URL is accessed	1. Application URL is accessible. 2. Application is deployed	The application should load completely for the user when the URL is accessed
Verify whether user is able to see input fields on logging in	1. Application is accessible 2. User is log in the application	User should be able to see input fields on logging in
Verify whether user is able to edit all input fields	1. Application is accessible 2. User is log in to application	User should be able to edit all input fields
Verify whether user gets submit button to submit the inputs	1. Application is accessible 2. User is log in to application	User should get submit button to submit the inputs
Verify whether user is able to get the output on submitting the results	1. Application is accessible 2. User is log in to application 3. User has submitted the results	User should get the output after submit has been sent.