

**DEFENCE BIO-ENGINEERING & ELECTRO MEDICAL
LABORATORY (DEBEL)**

**DEFENCE RESEARCH AND DEVELOPMENT
ORGANISATION (DRDO)**

Internship Report on Data Analysis of ILSS parameters and CO₂ capture

Submitted by:

Aditya Saroha

B.E. Computer Science and Engineering

USN: 1SJ22CS003

Internship Duration:

2nd December 2024 – 2nd June 2025

Supervisor:

Shri Amrit Roy, Scientist "E", DEBEL, DRDO

**SJC Institute of Technology,
P.B, No. 20, BB Rd, Chickaballapur,
Karnataka, Pin - 562101**

Acknowledgements

On behalf of myself, Aditya Saroha, I would like to express my sincere gratitude to the following individuals and institutions for their invaluable support and guidance through my internship at the Defence Bio-Engineering & Electro Medical Laboratory (DEBEL):

Dr. R. INDUSHEKAR, Director, DEBEL: I extend my sincere gratitude to The Director, DEBEL for providing me with the opportunity to intern at this prestigious laboratory.

Dr. Alka Chatterjee, Sc "G", DEBEL: I am particularly grateful to Dr. Alka Chatterjee for allowing and facilitating this internship at DEBEL.

Dr. A. Aparna G. Sastry, Sc "F", Group Head (HRD): I am grateful to Dr. Aparna Sastry for the assistance during my internship, which facilitated a smooth and enriching experience.

Mr. Amrit Roy (Supervisor), Sc "E" DEBEL: I am deeply grateful to Mr. Amrit Roy for his unwavering guidance, insightful suggestions, and constant encouragement during my internship at DEBEL. His true expertise in the field was instrumental in shaping the direction and success of my project/tasks. On a personal note, sir already knew what kind of errors we could face during the project/tasks, which demonstrates his true knowledge in the field.

Dr. Manjunatha Kumar B.H., HoD CSE, SJCIT: I would like to express my sincere gratitude to Dr. Manjunatha Kumar B.H., Head of the Department of Computer Science & Engineering, for providing me with the opportunity and guidance to complete this internship at DEBEL. Their constant support and encouragement have been invaluable throughout the course of this work.

Mr. Ajay N., Assistant Professor, Dept. of CSE, SJCIT: I extend my heartfelt thanks to Asst. Prof. Ajay N. for his valuable guidance, constant support, and insightful discussions throughout the duration of this project.

SJC Institute of Technology: I extend my appreciation to SJC Institute of Technology, for providing me with the opportunity to pursue this internship at DEBEL. The college's emphasis on practical learning experiences significantly contributed to our development.

Table of Contents

1	Introduction	3
2	Idea Presentation	3
2.1	Data Analysis and Machine Learning	3
2.2	Methodology	4
2.2.1	Understandinng and Preprocessing the Data	4
2.2.2	Exploratory Data Analysis (EDA)	5
2.2.3	Feature Engineering	6
2.2.4	Model Selection	7
2.2.5	Model Training and Validation	7
2.2.6	Model Interpretation and Explainability	10
2.2.7	Future Work And Improvements	11
3	Simulation of the BOS (Backup Oxygen System)	11
3.1	Idea Presenatation	11
3.2	Objectives of the Simulation	11
3.3	Mathematical Modelling of the Nozzle System	12
3.4	Machine Learning based Flow Estimation	13
3.5	Python Implementation of the above approach	13
3.6	Visualization and Evaluation	15
3.7	Summary and Future Work	15
4	Computational Chemistry and ML Study	16
4.1	Dataset Collection and Exploration	17
4.2	Machine Learning Modelling	18
4.3	Results Intepretation	18
4.4	Python Implementation of the above approach	21
5	Conclusion	24

1 Introduction

This report outlines the work carried out by the student Aditya Saroha (USN: 1SJ22CS003) during the internship at DEBEL-DRDO from December 2, 2024, to June 2, 2025. The internship primarily revolved around exploring data-driven analysis of operational parameters related to subsystems under consideration by the lab, building machine learning models, and simulating and understanding critical performance characteristics. It also later extended into simulation-based studies of the Backup Oxygen System (BOS) used under one of the subsystems under consideration and a computational chemistry task using machine learning for carbon material characterization having high amounts of CO_2 adsorption.

The following sections detail the conceptual underpinnings of the internship tasks, the methodologies adopted for data interpretation and relation mining, simulation efforts toward the BOS system, and a brief outline of the computational chemistry-related work.

2 Idea Presentation

OBOGS refers to the On-Board Oxygen Generating System, which is a central life support system that provides not only oxygen but also physiological protection against various forms of fatigue encountered during high-speed, high-altitude maneuvering of a fighter aircraft. It replaces the LOS (Liquid Oxygen System) by utilizing compressed air extracted from engine, primarily used for various aircraft systems and functions. Separating the air’s components using various technologies like PCA, etc.

The system in whole comprises of a BOS (Backup Oxygen System) which is used when the aircraft is above some specified altitude and it can provide oxygen for 40 minutes, EOS (Emergency Oxygen System) which comes into play when the BOS is exhausted by the pilot, and it can provide oxygen for 7 minutes.

OBOGS systems are equipped with advanced onboard data logging systems that continuously monitor and record a wide range of critical operational parameters such as oxygen concentration levels, flow rate, cabin pressure, valve pressures, pilot breathing rate, and system response times throughout the entire duration of the flight. This data is stored with accurate timestamps for complete post-mission traceability.

Once the aircraft lands and the mission is complete, the recorded data is retrieved and examined thoroughly to assess the performance of the OBOGS under real operational conditions. Every parameter is compared against expected limits to detect any anomalies, unexpected fluctuations, or early signs of performance degradation.

This post-flight data evaluation helps in identifying inefficiencies, minor faults, or gradual system drift that could lead to critical failures if not addressed in time. By detecting such issues before they occur in future missions, the reliability of the system is maintained and the safety of the pilot is ensured, especially during high-speed and high-altitude operations where any disruption in oxygen delivery can have serious consequences.

2.1 Data Analysis and Machine Learning

In the early weeks of the internship, the primary focus was to understand and extract relationships from a 150K-row dataset representing various parameters in an IAF aircraft’s breathing subsystem. Each column of the dataset represented different pressure, flow, or temperature readings at specific subsystems. Understanding the domain-specific meanings (eg. columns like XXXILP7_Barg means Inlet pressure of valve 7) formed the basis of constructing meaningful correlations.

Various statistical, computational and machine learning methods were implemented to detect hidden relationships within the huge data set, including:

- Ridge, Lasso and Linear Regression models to predict the regression line

- Pearson Correlation
- Polynomial Fits (Square & cubic)
- Cross-Correlation and Granger Causality tests for temporal patterns
- Random Forests
- Support Vector Machine

The results, though sometimes counter-intuitive (eg. R2 score showing >0.9 but poor predictive fits), led to important insights about the dataset's complexity and noise characteristics. A dedicated graph plotting script was also developed to automate visual diagnostics between all meaningful variable pairs, rendering nearly 1500–4000 graphs across iterations.

After correcting a bug in the deviation calculation logic, more accurate deviation percent graphs were obtained, showing better validation of derived equations.

2.2 Methodology

In any machine learning project, a systematic approach is essential to ensure reliable results and meaningful insights. The methodology typically involves several key stages, starting from understanding and preprocessing the dataset, followed by exploratory data analysis, model selection, training, evaluation, and interpretation. Each step builds upon the previous to refine the model and extract valuable patterns or predictions from the data. The following subsections outline this general workflow.

2.2.1 Understanding and Preprocessing the Data

Before applying any machine learning techniques, it was essential to first understand the system thoroughly. Without knowing how the system works and what the data actually represents, it is not possible to apply any method effectively. So, I started by gaining a complete understanding of how the system behaves, how different components interact, and what kind of outputs are expected.

Once the system was understood, I shifted my focus to the dataset. The features in the data had machine-level or internal naming conventions, which were not easy to interpret. To resolve these and assign meaningful context to each feature, I had to collaborate with multiple teams who had worked on different parts of the system. This step helped me clean the dataset and ensure I was working with reliable and interpretable features.

Next, I removed outliers and irrelevant data points—such as boolean values from sensors that did not influence system behavior in any significant way. This reduced noise and unnecessary complexity in the dataset.

One of the most important steps in preprocessing was data normalization. Without normalization, the model training process can become inefficient and unstable. Normalization brings all features to a similar scale, which significantly improves the behavior of optimization algorithms during training. This is especially important when using techniques based on gradient descent.

Gradient descent tends to perform better when the input data is normalized, particularly in the case of convex functions. On normalized data, the algorithm can move steadily towards the local minima with fewer deviations. However, when the data is not normalized or scaled properly, gradient steps become uneven and the convergence path becomes noisy and slower. In some cases, the algorithm may get stuck or take a long time to reach the optimal point. This effect is illustrated clearly in Figure 1, which shows how the presence or absence of normalization affects the convergence trajectory.

A explanation of that is, in the given loss function (this is the loss function of Linear Regression) below:

$$\mathcal{L} = \frac{1}{n} \cdot \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

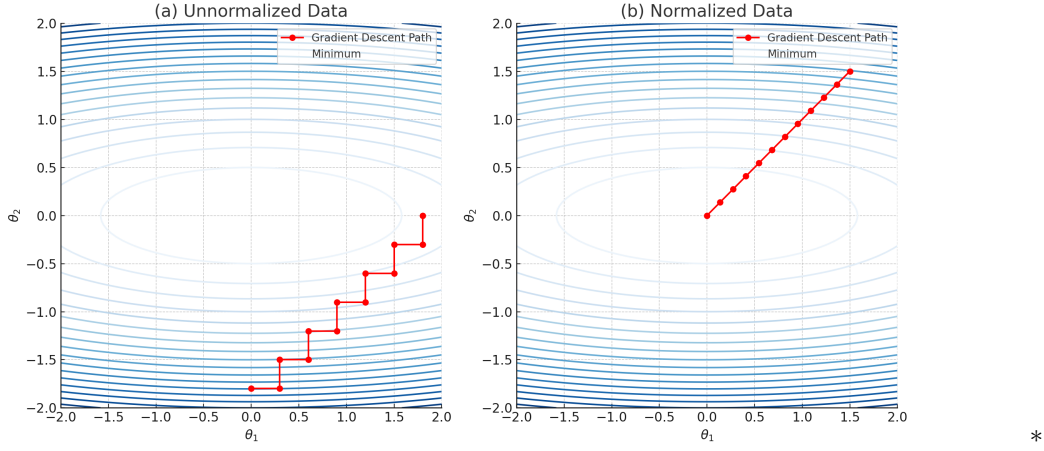


Figure 1: How gradient descent algorithm approaches local minima when data is not Normalized vs Normalized

Since \hat{y}_i is nothing but $w x_i + b$, so our equation becomes

$$\mathcal{L} = \frac{1}{n} \cdot \sum_{i=1}^n ((w \cdot x_i + b) - y_i)^2$$

Now this above equation is dependent on 2 variables w and b , in-order to find the gradient we need to partially differentiate it w.r.t w and b , these equations are given below:

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{2}{n} \cdot \sum_{i=1}^n x_i \cdot (w \cdot x_i + b - y_i) \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial b} = \frac{2}{n} \cdot \sum_{i=1}^n (w \cdot x_i + b - y_i)$$

Now when we update the parameter using the below equation:

$$\theta = \theta - \alpha \cdot \frac{\partial \mathcal{L}}{\partial \theta}$$

If the features are not normalized, then the gradient calculated would be off by a very big factor which will eventually affect the parameters when they are updated using the above equation.

For this dataset, I have selected the most commonly used normalization method (especially via StandardScaler) which is called Z-score normalization or Standardization and is available in *sci-kit learn* library. This method does not bound the values to a specific range like $[0, 1]$, but instead transforms the data so that the mean = 0 and standard deviation = 1.

For each feature x_i in the dataset:

$$x_{\text{scaled}} = \frac{x - \mu}{\sigma}$$

where:

x = original value

μ = mean of the feature

σ = standard deviation of the feature

By following this structured approach in understanding the system, cleaning and interpreting the data, and applying normalization, I ensured that the data was in the best possible form for further analysis.

2.2.2 Exploratory Data Analysis (EDA)

To understand the underlying patterns in the dataset, an exploratory data analysis was conducted. A *heatmap* was generated using the correlation matrix to identify linear relationships between

numerical features and target variables or features, highlighting any multicollinearity. A sample code for selecting the top N features that have the highest Pearson Correlation is given below:

Listing 1: Sample code for finding top N features which correlates with target variable

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv("") # Mention your source file if in excel format use
                        # pd.read_excel()
data_encoded = pd.get_dummies(data, drop_first = True)

N = bool               # Will store your N value
target_feature = ""    # Will store your target variable / feature

correlations = data_encoded.corr()

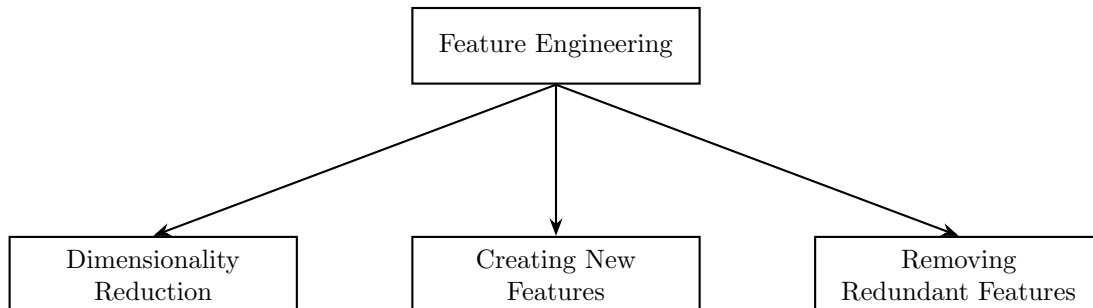
top_features = correlations[target_feature].abs().sort_values(ascending =
                        False).head(N).index

plt.figure(figsize = (15, 15))
sns.heatmap(data_encoded[top_features].corr(), annot = True)
plt.title("Top 20 features having highest correlations with target feature"
          )
plt.show()
```

Distribution analysis was performed to examine the spread and skewness of individual features, helping to detect outliers or data imbalance. Pairwise feature relationships were explored using scatter plots and pair plots to visually inspect interactions between variables. Finally, an initial assessment of feature importance was carried out using simple models and statistical metrics to identify which variables may have the most predictive power.

2.2.3 Feature Engineering

Feature selection and engineering play a crucial role in improving model performance and interpretability. This phase focuses on identifying the most relevant variables, transforming them if needed, and removing noisy or redundant information.



Dimensionality Reduction (PCA)

High-dimensional datasets can contain redundant or irrelevant features that reduce model efficiency. Techniques such as *Principal Component Analysis (PCA)* are used to project the data into a lower-dimensional space while preserving as much variance as possible, it played a crucial role in enhancing the model performance for the dataset I was working with.

Creating New Features from Existing Ones

Domain knowledge or exploratory analysis often reveals opportunities to create new features that

better represent the underlying data. These include *polynomial features*, *interaction terms*, *normalized ratios*, or *aggregate statistics* of related parameters, I have not used this in my project.

Removing Redundant Features

Highly correlated features or those with little variance contribute little to model performance and may even introduce noise. Techniques like *correlation thresholding*, *Variance Inflation Factor (VIF)*, or *recursive feature elimination* are employed to systematically drop such features, leading to more robust and generalizable models.

2.2.4 Model Selection

Model selection is one of the most critical stages in a machine learning pipeline, as it directly determines the quality and generalizability of predictions. Given that the task at hand involves predicting continuous numerical values, the problem is framed as a regression problem.

- **Linear Models:** These include *Linear Regression*, *Ridge Regression*, and *Lasso Regression*, which are often effective for problems with well-behaved, linearly separable relationships. Ridge and Lasso also help in mitigating overfitting by introducing regularization.
- **Tree-Based Models:** Algorithms like *Decision Trees*, *Random Forests*, and *XGBoost* were explored for their ability to model complex, non-linear relationships without requiring extensive feature preprocessing.
- **Support Vector Machines (SVM):** Although traditionally used for classification tasks (we have trained SVM for one our tasks which requires to predict at certain altitude whether the temp. will go up a certain limit or not?), SVMs can be adapted for regression using *SVR (Support Vector Regression)*. These models are robust to high-dimensional feature spaces and capable of capturing intricate relationships between variables.

2.2.5 Model Training and Validation

Once the models were selected, the next step involved systematically training and validating them to ensure their predictive performance on unseen data.

The dataset was initially split into training and testing sets, typically in an 80:20 or 70:30 ratio, to assess generalization performance. A sample code on how to do the same is presented below:

Listing 2: Sample code on `train_test_split`

```
from sklearn.model_selection import train_test_split

X = data_encoded["feature 1", "feature 2"]      # List of all of your
                                                # important features

y = data_encoded["target_variable"]            # Your target feature

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,
                                                    random_state = 12)
```

In the above code the `random_state` ensures reproducibility, you'll get the same split every time you run the code.

Training Metrics and Loss Functions Used

Each regression model was trained to minimize an appropriate loss function, depending on its nature:

- **Ridge Regression (L2 Regularization)**

Ridge Regression adds a penalty to the Linear Regression loss to prevent overfitting by shrinking the coefficients, the loss function is given as follows

$$\mathcal{L}_{\text{Ridge}} = \frac{1}{n} \cdot \sum_{i=1}^n (\hat{y}_i - y_i)^2 + \lambda \cdot \sum_{j=1}^p w_j^2$$

where:

λ = regularization strength

w_j = weight of feature j

SecondTerm = penalty on large coefficients

- **Lasso Regression (L1 Regularization)**

Lasso also prevents overfitting but tends to produce sparse models by pushing some coefficients exactly to zero (useful for feature selection).

$$\mathcal{L}_{\text{Lasso}} = \frac{1}{n} \cdot \sum_{i=1}^n (\hat{y}_i - y_i)^2 + \lambda \cdot \sum_{j=1}^p |w_j|$$

Same terms as ridge, but the penalty is the absolute value of weights.

- **Support Vector Regression (SVR)**

SVR tries to fit a line/curve such that all predictions fall within a margin ϵ , while minimizing complexity (i.e. flatness of the model).

$$\mathcal{L}_{\text{SVR}} = \sum_{i=1}^n \max(0, |\hat{y}_i - y_i| - \epsilon)$$

The above loss function ignores small errors within the ϵ margin and penalizes only those errors that lie outside the margin.

- **Decision Tree Regressor**

Decision Trees partition the feature space into regions and assign constant predictions per region to minimize total error, typically uses Mean Squared Error at each split.

$$\mathcal{L}_{\text{Tree}} = \sum_{j=1}^J \sum_{i \in R_j} (\hat{y}_i \cdot R_j - y_i)^2$$

where:

R_j = region j (leaf node)

$\hat{y} \cdot R_j$ = mean target value in region R_j

- **Random Forest Regressor**

An ensemble of multiple Decision Trees where each tree is trained on a bootstrap sample, and the final output is the average of all predictions. Same as Decision Trees, typically MSE across individual trees is averaged:

$$\mathcal{L}_{\text{RF}} = \frac{1}{T} \cdot \sum_{t=1}^T \sum_{j=1}^{J_t} \sum_{i \in R_{jt}} (\hat{y}_i \cdot R_{jt} - y_i)^2$$

where:

T = number of trees

R_{jt} = region j in tree t

- **XGBoost Regressor**

An advanced boosting algorithm that builds trees sequentially, where each new tree attempts to correct the errors of the previous ones. Usually uses regularized objective:

$$\mathcal{L}_{\text{XGB}} = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

where:

$$l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2 \text{ (typically MSE)}$$

$$\Omega(f_k) = \gamma \cdot T + \frac{1}{2} \cdot \lambda \cdot \sum_{j=1}^T w_j^2 \text{ region } j \text{ in tree } t$$

T : number of leaves in tree

γ, λ : regularization parameters

Below is a sample code for training a Random Forest regressor on a similar dataset:

Listing 3: Sample code on training a Random Forest Regressor

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

X, y = make_regression(n_samples = 1000, n_features = 5, noise = 10,
                      random_state = 42)
feature_names = [f'feature_{i}' for i in range(X.shape[1])]
df = pd.DataFrame(X, columns = feature_names)
df['target'] = y

print(df.describe())
sns.pairplot(df)
plt.show()

plt.figure(figsize = (12, 6))
sns.heatmap(df.corr(), annot = True, cmap = "coolwarm_r", fmt = ".2f")
plt.title("Feature Correlation map")
plt.show()

scaler = StandardScaler()
X_scaled = scaler.fit_transform(df[feature_names])

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size
    = 0.3, random_state = 12)

rf = RandomForestRegressor(n_estimators = 100, max_depth = 5, random_state
    = 42)

rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
```

```

r2 = r2_score(y_test, y_pred)

print("\nModel Performance:\n")
print("Mean Squared Error: ", mse)
print("R2 Score: ", r2)

```

2.2.6 Model Interpretation and Explainability

Feature importance analysis provides insights into how individual input features contribute to a model's predictions. By quantifying the influence of each feature, it helps in understanding, validating, and trusting the model's decision-making process. This analysis can guide feature selection, highlight potential data biases, and inform domain-specific reasoning.

In tree-based models like Random Forests or XGBoost, feature importance is typically calculated based on metrics such as Gini importance or the average gain from splits. In contrast, linear models rely on the magnitude and sign of learned coefficients. For black-box models (a models whose internal workings are not yet interpreted by humans), model-agnostic methods like SHAP and LIME offer local and global interpretability by approximating the model behavior around specific predictions.

Feature importance not only enhances model transparency but also serves as a foundation for further optimization and debugging in machine learning workflows.

SHAP Explanations (SHapley Additive exPlanations)

SHAP is a game-theoretic approach to explain the output of any machine learning model. It is based on the Shapley values from cooperative game theory, which fairly distribute the "payout" (model prediction) among the features.

Given a model $f(x)$, the SHAP value for feature i in instance x is defined as:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|! \cdot (|N| - |S| - 1)!}{|N|!} [f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S)]$$

where:

N is the set of all features

S is a subset of features not containing feature i

$f_S(x_S)$ is the model trained or evaluated using only the features in set S

This equation computes the *marginal contribution* of feature i across all possible subsets of features, ensuring a fair and additive distribution:

$$f(x) = \phi_0 + \sum_{i=1}^n (\phi_i)$$

Here, ϕ_0 is expected model output (the base value), and the sum of all ϕ_i values gives the actual prediction.

Below is an example SHAP beeswarm plot for a related dataset:

Visualizations of Learned Relationships

Visual analysis was used to interpret the model's behavior:

- SHAP summary plots to identify feature impact distribution
- SHAP dependence plots to visualize nonlinear interactions
- Feature vs. prediction scatter plots to observe learned trends

These visualizations help in verifying the model's reasoning aligns with domain expectations and detecting any unintended biases or spurious correlations.

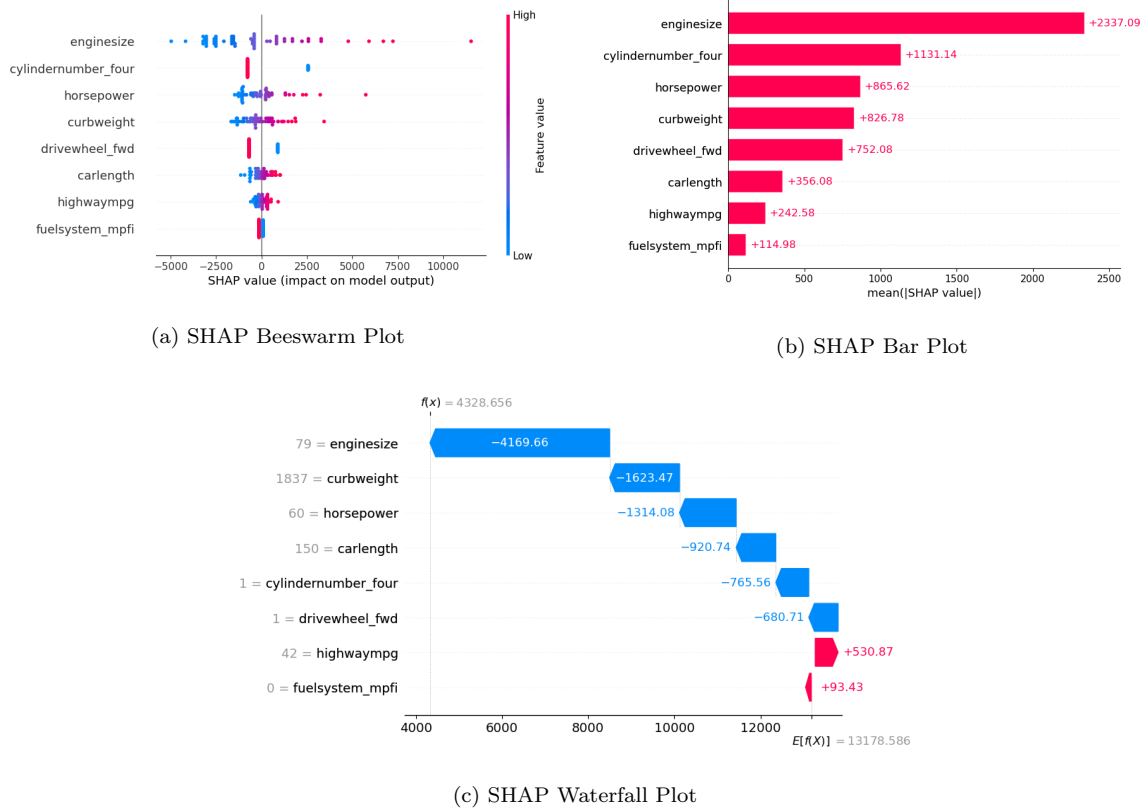


Figure 2: SHAP plots: (a) Beeswarm, (b) Bar, and (c) Waterfall

2.2.7 Future Work And Improvements

Future improvements to this study could involve exploring additional models such as *Gradient Boosting Machines*, or *Neural Networks/Deep Neural Networks* with multiple layers to capture more complex relationships in the data. Incorporating domain knowledge during feature engineering or model interpretation could improve both accuracy and explainability. Furthermore, setting up feedback loops and integrating online learning mechanisms would enable the model to adapt continuously to new data, enhancing its robustness in dynamic environments.

3 Simulation of the BOS (Backup Oxygen System)

3.1 Idea Presenatation

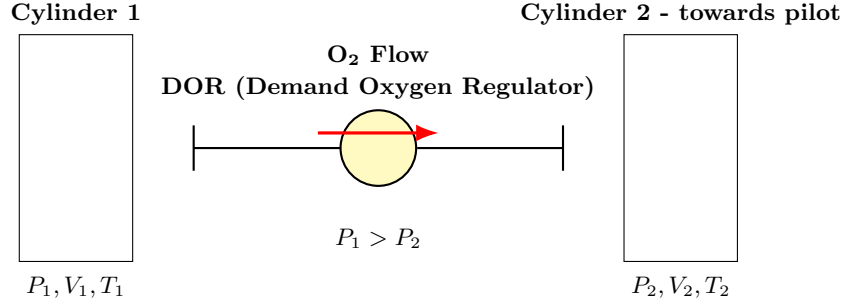
The *Backup Oxygen System (BOS)* used in high-performance situations where aircraft is critical to maintaining pilot health and performance during flight. In particular, the Backup Oxygen System ensures continued delivery of oxygen in emergency or failure scenarios. To understand its dynamic behavior, a simulation model was developed that mimics real-time pressure changes and flow conditions within the pipeline and nozzle system feeding into the pilot's mask. This simulation also serves as a sandbox to evaluate how well basic physical principles and machine learning (ML) models can estimate system performance under fluctuating pressure conditions.

3.2 Objectives of the Simulation

- To simulate real-time pressure variations across two oxygen supply lines
- To model the pressure dynamics across a nozzle regulating flow to the pilot

- To estimate flow rate as a function of upstream and downstream pressures
- To validate the accuracy of pressure-based flow estimation using a simple ML model (Linear Regression)
- To visualize system response and identify where physical estimates and ML predictions diverge

Below is a generalized diagram of the BOS which is undergoing simulation:



To approximate realistic conditions, artificially generated pressure data (with added noise) was used as a substitute for actual flight test data.

3.3 Mathematical Modelling of the Nozzle System

The BOS contains two critical pressure points:

- P_1 : Upstream pressure (before the nozzle)
- P_2 : Downstream pressure (after the DOR, towards the pilot mask)

To simulate the internal nozzle behavior, intermediate pressure values were defined:

- P' : Estimated internal pressure (before the nozzle)
- P'' : Estimated pressure (after the DOR, towards the pilot mask)

Nozzle inlet pressure P' :

$$p' = p_1 - k_1 \cdot |p_1 - p_2|^\gamma$$

Nozzle outlet pressure / pressure drop P'' :

$$p'' = p_2 + k_2 \cdot |p_1 - p_2|^\eta$$

Estimated Flow Rate is given by (intuition by Ohm's law for fluids):

$$Q = \frac{p' - p''}{R}$$

where:

- k_1, k_2 are empirical constants to be tuned,
- $\gamma, \eta \in (0.8, 1.0)$ are smoothing exponents based on experimentation
- R is a resistance factor mimicking flow constriction across the nozzle

This formulation captures the fact that flow occurs due to a pressure differential and the system behaves non-linearly when the pressure difference is large or rapidly changing. Also, these equations are not physically exact but were engineered to model realistic flow behavior using pseudo-empirical rules and exponents, which can later be updated after experimental calibration.

3.4 Machine Learning based Flow Estimation

To validate the simplified physics-based model and test learnability, a linear regression model was applied. The flow Q was modeled as a function of the upstream and downstream pressures:

$$Q_{\text{ML}} = \alpha \cdot P_1 + \beta \cdot P_2 + \chi$$

where:

α, β : Learned Coefficients

χ : is intercept term

After this the model was trained on the simulated pressure pairs and corresponding Q values using the *Linear Regression* module from *sklearn*. This approach helped assess how well a simple linear model could replicate the flow predictions generated from the above model.

3.5 Python Implementation of the above approach

Listing 4: BOS Simulation

```
- [x] 1. Simulation task of pilot breathing tank.

# Python Code - Approach one, and not too efficient -> consult with Amrit
# sir once on the accuracy part

import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Note: later we should replace this with actual data from the flight tests
.
# Right now, using artificial data generation
def generate_pipe_pressures(n_samples):
    time = np.linspace(0, 20, n_samples)

    # Pipe 1 pressure fluctuates sinusoidally(Sine functions), with added
    # random noise(to simulate real world pressures of pilot breathing)
    pressure_1 = 5 + 1.5 * np.sin(0.4 * time) + 0.3 * np.random.randn(
        n_samples)

    # Pipe 2 pressure
    pressure_2 = 3 + 0.8 * np.cos(0.3 * time) + 0.2 * np.random.randn(
        n_samples)

    return time, pressure_1, pressure_2

def nozzle_model_basic(p1, p2):
    # Improve this significantly with
    # the help from sir, ...
    p_prime_list = []
    p_double_prime_list = []
    flow_estimations = []

    for i in range(len(p1)):
        p_1 = p1[i]
        p_2 = p2[i]

        # drop before nozzle
        p_ = p_1 - 0.1 * (abs(p_1 - p_2))*0.95
        pp = p_2 + 0.05 * (abs(p_1 - p_2))*0.85
```

```

        # Partially wrong, due to artificial data
        q = (p_ - pp) / 2.05

        p_prime_list.append(p_)
        p_double_prime_list.append(pp)
        flow_estimations.append(q)

    return p_prime_list, p_double_prime_list, flow_estimations

# ML to test modeling pressure diff
def estimate_flow_with_ml(p1, p2, flows):
    # shape for model input
    X = np.column_stack((p1, p2))
    y = np.array(flows)

    # Linear model
    model = LinearRegression()
    model.fit(X, y)

    predicted_flows = model.predict(X)

    return predicted_flows, model

# simulate and visualize resut
def visualize_all(t, p1, p2, p_prime, p_dprime, q_real, q_pred):
    plt.figure(figsize=(16, 12))

    plt.subplot(4, 1, 1)
    plt.plot(t, p1, color='blue', label='p1 (Pipe 1)')
    plt.plot(t, p2, color='green', label='p2 (Pipe 2)')
    plt.legend()
    plt.title('Raw Pipe Pressures')
    plt.grid(True)

    plt.subplot(4, 1, 2)
    plt.plot(t, p_prime, label="p' (Before nozzle)", color='orange')
    plt.plot(t, p_dprime, label="p'' (After nozzle)", color='red')
    plt.legend()
    plt.title('Nozzle Internal Pressures')
    plt.grid(True)

    plt.subplot(4, 1, 3)
    plt.plot(t, q_real, label='Flow Rate (Simulated)', color='purple')
    plt.plot(t, q_pred, label='Flow Rate (Predicted ML)', linestyle='--',
             color='brown')
    plt.legend()
    plt.title('Flow Rate Comparison')
    plt.grid(True)

    plt.subplot(4, 1, 4)
    err = np.array(q_real) - np.array(q_pred)
    plt.plot(t, err, label='Prediction Error', color='gray')
    plt.legend()
    plt.title('ML Prediction Error Over Time')
    plt.grid(True)

    plt.tight_layout()
    plt.show()

```

```

# utility to print model coefs
def print_model_details(model):
    print("Flow estimation model coefficients:")
    print(f"Coef for p1: {model.coef_[0]:.4f}")
    print(f"Coef for p2: {model.coef_[1]:.4f}")
    print(f"Intercept: {model.intercept_[0]:.4f}")
    print()

# Additional debug info if needed (disabled by default)
# Addition of this is to debug some errors which are being generated
# unexpectedly, consult with sir once, also mention in mail
def debug_print_additional_details(time, p1, p2, q):
    for i in range(0, len(time), max(len(time)//10,1)):
        print(f"t={time[i]:.2f}, p1={p1[i]:.2f}, p2={p2[i]:.2f}, Q={q[i]:.2f}")
    print()

# main function for simulation
def run_large_simulation():
    num_samples = 500
    time_vals, press1, press2 = generate_pipe_pressures(num_samples)

    p_prime, p_dprime, flow_values = nozzle_model_basic(press1, press2)

    predicted_flows, flow_model = estimate_flow_with_ml(press1, press2,
        flow_values)

    print_model_details(flow_model)

    # remove this to print random samples
    # debug_print_additional_details(time_vals, press1, press2, flow_values
    )

    visualize_all(time_vals, press1, press2, p_prime, p_dprime, flow_values
        , predicted_flows)

# call the simulator
run_large_simulation()

```

3.6 Visualization and Evaluation

To assess the performance and accuracy of the simulation:

- The *pressure curves* P_1, P_2, P' and P'' were plotted over time
- The *simulated flow* Q was compared with the *predicted flow* Q_{ML}
- The *prediction error* over the time was also visualized

These graphs are presented below:

This allowed a clear visual understanding of the model behavior and highlighted areas where improvements or corrections were needed.

3.7 Summary and Future Work

This simulation served as a first iteration toward modeling the BOS in realistic conditions. Although simplified equations were used but it produced reasonable outputs and provided a foundation for:

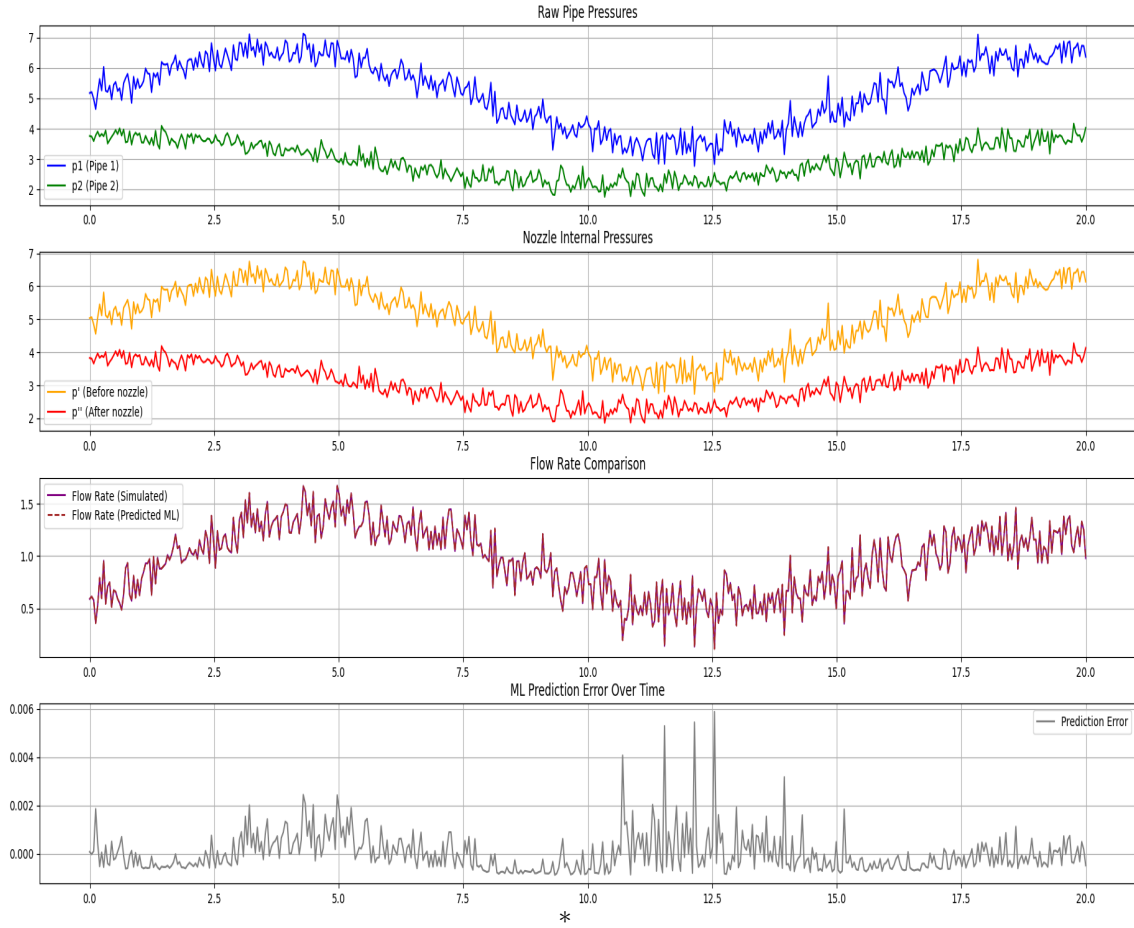


Figure 3: BOS simulation plots

- Replacing synthetic data with real flight sensor data
- Improving pressure drop models with the help from domain experts
- Enhancing regression model using *non-linear* or *neural network* based approaches
- Incorporating *time series memory* via RNN's or similar models

Further iterations will aim to better match real fluid dynamics by integrating actual aircraft breathing profiles and nozzle performance data.

4 Computational Chemistry and ML Study

With the growing emphasis on reducing atmospheric carbon dioxide (CO_2) levels, materials with high CO_2 adsorption capacity have gained significant attention. Among them, porous carbon materials have shown promising characteristics due to their large surface area, tunable pore structures, and functional group diversity. Recent research efforts have focused on using machine learning to predict and optimize these materials based on their synthesis parameters.

In this project, we revisited a previously published paper involving the production and performance of various carbon-based materials. The aim was to rebuild and extend the original machine learning framework which had primarily relied on feature selection and optimization using a Dung Beetle Optimizer (DBO).

4.1 Dataset Collection and Exploration

The dataset was collected from various literature studies, and it consisted the following features:

- S_{BET} stands for the specific surface area obtained by BET method
- V_{total}^b stands for the total pore volume
- V_{total}^c stands for the micropore volume
- C, H, N, O stands for Ultimate Analysis (%)
- adb stands for adsorption rate of CO_2 at 25°C at 1 bar

Preprocessing and Exploration steps included:

- Handling the missing values in the target variable (adb), NaN is replaced by *mean* of the values
- Normalizing the numerical values for better model output
- Visualizing the pairwise relationships and correlation matrices to identify potential linear dependencies
- Calculating the *mean*, *median*, *25% percentile*, *50% percentile*, *75% percentile*, *max* and *min* values

The Heatmap of the dataset is given below:

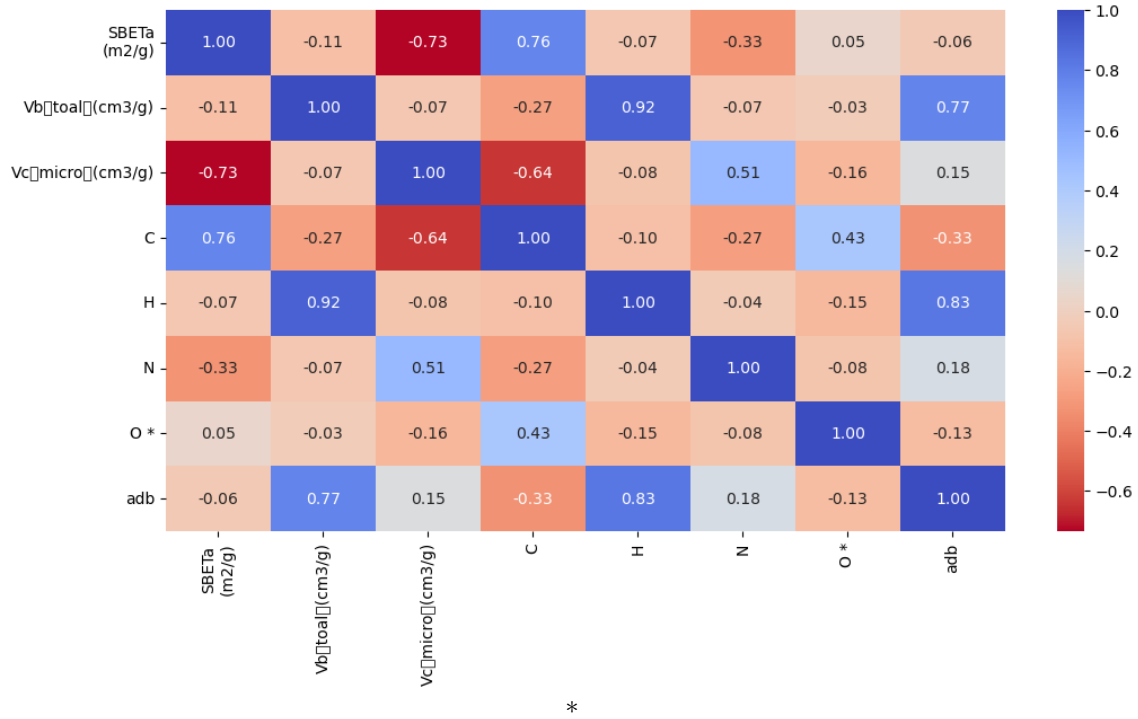


Figure 4: Heatmap of the dataset

From the above graph we can easily infer that, H , *Total Pore Volume* and C have the highest correlations respectively 0.83, 0.77 and 0.33.

4.2 Machine Learning Modelling

In this study we will use two types of ML models namely *HistGradient Boosting Regressor* and *CatBoosting Regressor*.

The default loss function for *HistGradient Boosting Regressor* is Squared Error, which is given below:

$$L(y, \hat{y}) = \frac{1}{n} \cdot \sum_{i=1}^n (y_i - \hat{y})^2$$

where:

$$\hat{y} \text{ is } w \cdot x_i + b$$

and default loss function for *CatBoosting Regressor* is Root Mean Squared Error, which is given below:

$$L(y_i, \hat{y}_i) = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

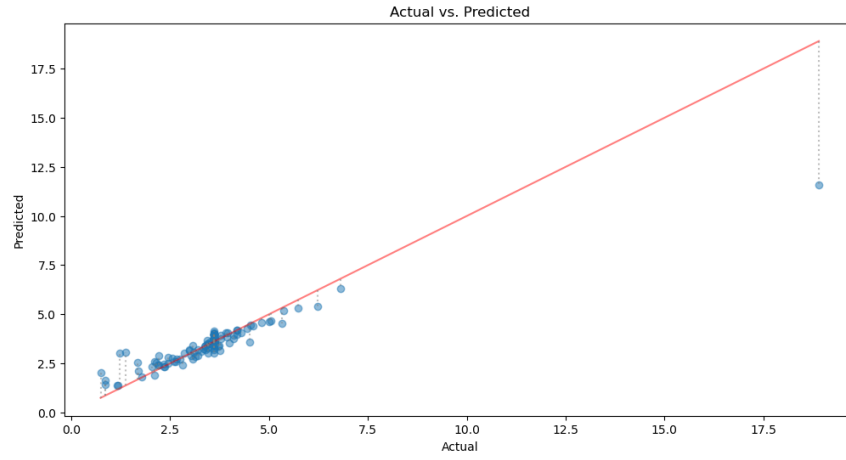
4.3 Results Interpretation

Both the models gave a decent R^2 score on the test dataset, which was around 30% of the entire dataset. R^2 score for HistGradient Boosting was 0.79 and for CatBoost it was around 0.81, but the CatBoosting was able to explain the variance in the dataset at higher values more efficiently than the HistGradient Boosting, which is evident from the below plots:

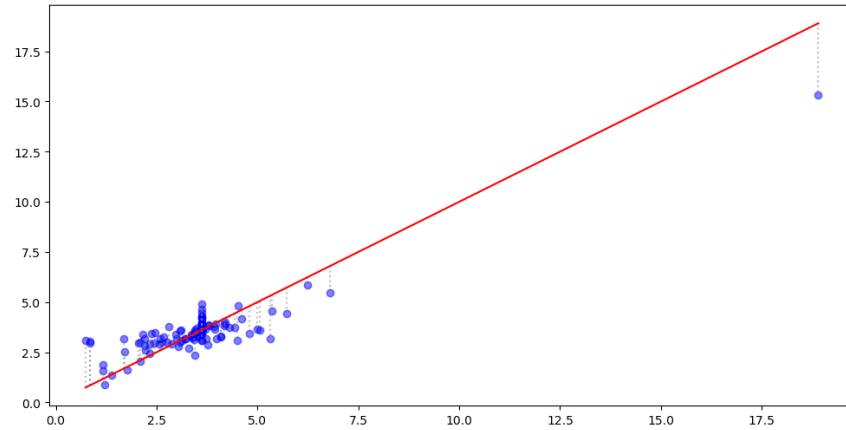
To evaluate the normality assumption of the model residuals, QQ plots and histograms were analyzed for both models under consideration. The QQ plots visually compare the quantiles of the residuals against a theoretical normal distribution. Ideally, residuals that follow a normal distribution will lie along the reference line. Deviations from this line, especially at the extremes, may indicate non-normality due to skewness, outliers, or heavy tails.

To assess the quality of the regression model, a histogram of the residuals (the differences between actual and predicted values) was plotted along with a Kernel Density Estimate (KDE) curve. The residuals are primarily centered around zero, indicating that the model does not exhibit significant bias. The bell-shaped and approximately symmetric distribution of the KDE curve suggests that the errors follow a near-normal distribution. This is a desirable property, as it implies that the model's predictions are generally accurate and the errors are randomly distributed. A lack of skewness or heavy tails further supports the assumption that the model generalizes well to unseen data.

Below is given the QQ plots and histogram of the residuals for both *HistGradientBoosting Regressor* and *CatBoosting Regressor*:

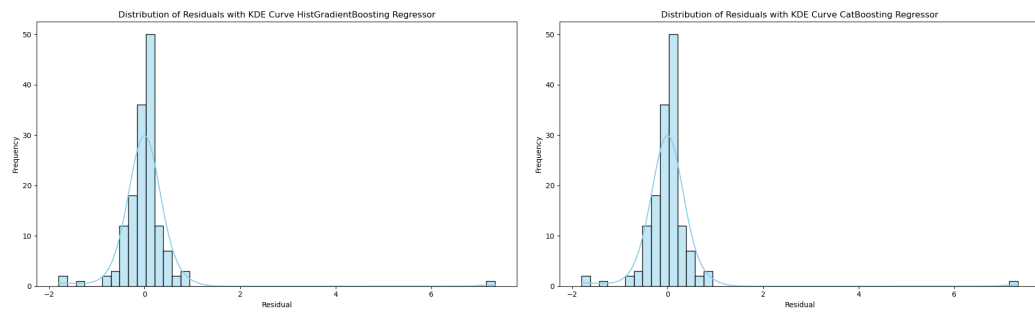


(a) HistGradientBoosting Regressor



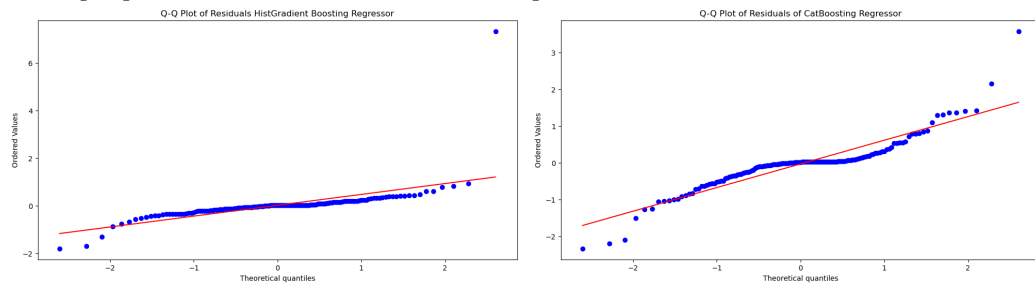
(b) CatBoosting Regressor

Figure 5: Comparison between HistGradientBoosting Regressor and CatBoosting Regressor



(a) Histogram of Residuals with KDE: HistGradient-Boosting Regressor

(b) Histogram of Residuals with KDE: CatBoosting Regressor



(c) QQ Plot: HistGradientBoosting Regressor

(d) Plot: CatBoosting Regressor

Figure 6: QQ plots and residual histograms for model evaluation. The plots indicate how well the residuals follow normality.

To gain insights into the contribution of individual features toward the model’s predictions, we employed *SHAP* (*SHapley Additive exPlanations*) analysis. Two types of SHAP visualizations were generated: a *beeswarm* plot and a *force plot* (*waterfall plot*).

Beeswarm Plot

The beeswarm plot summarizes the impact of each feature across all instances in the dataset. Each point represents a SHAP value for a particular sample and feature. The horizontal position indicates whether the feature increased or decreased the predicted output, while the color represents the feature’s value (blue = low, pink = high).

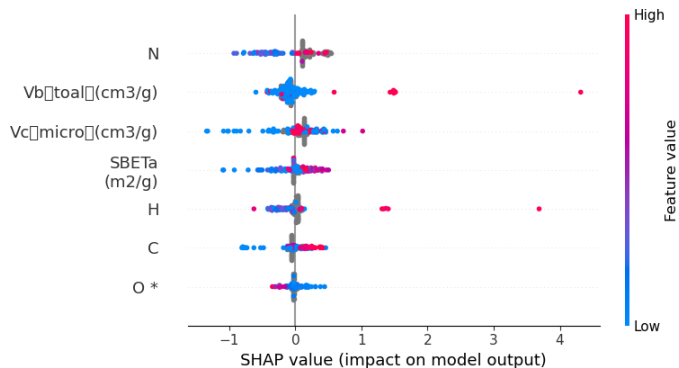


Figure 7: Beeswarm plot

From the plot we can infer that:

- Nitrogen (N) and micropore volume (V_{micro}^c) show strong positive SHAP values when their values are high, suggesting a significant positive contribution to the model output.
- Total pore volume (V_{total}^b) and surface area (S_{BET}) influence the predictions both positively and negatively depending on their value, reflecting more complex nonlinear relationships.
- Carbon (C) and Hydrogen (H) have a relatively narrower spread of SHAP values, indicating more moderate influence.

This plot helps identify which features are most influential and how their values affect the prediction outcomes.

SHAP Force Plot (Waterfall Plot)

The force plot shows a single prediction explanation, illustrating how each feature pushed the prediction from the base value $E[f(x)]$ toward the final model output $f(x)$. Features shown in red contributed positively (pushed the prediction higher), while those in blue pulled the prediction downward.

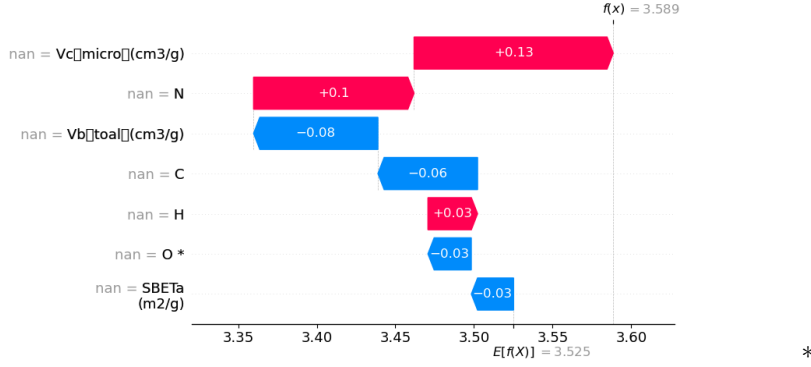


Figure 8: SHAP Force plot (Waterfall plot)

From the plot we can infer that:

- Micropore volume (V_{micro}^c) and Nitrogen content had the strongest positive contributions.
- Total pore volume (V_{total}^b), Carbon, and surface area (S_{BET}) slightly pulled the prediction down.

This visualization offers a detailed, case-specific view of feature contributions, which is especially useful for understanding individual predictions in critical applications.

Also, just by looking at the heatmap, it initially seems like we are contradicting the findings from the paper "Research on the production and characteristics of carbon materials with high CO_2 adsorption performance: Based on machine learning and dung beetle optimizer methods." However, it's only after training the model and analyzing the SHAP force plots that the true importance of the micropore volume (V_{micro}^c) becomes clear. This highlights the limitation of relying solely on correlation heatmaps and the value of model interpretability techniques like SHAP for uncovering deeper insights.

4.4 Python Implementation of the above approach

Listing 5: CO_2 adsorption ML (code compiled from Jupyter Notebook)

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

df.head()
df.describe()
print(df.columns)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

correlations = df.corr()
plt.figure(figsize = (12,6))
sns.heatmap(df.corr(), annot = True, cmap = "coolwarm_r", fmt = ".2f")

top_features = correlations["adb"].abs().sort_values(ascending=False).head(5).index
sns.heatmap(df[top_features].corr(), annot = True)

from sklearn.model_selection import train_test_split
```

```

X = df[["SBETa\\n(m2/g)", "Vb\\rtoal\\r(cm3/g)", "Vc\\rmicro\\r(cm3/g)", "C", "H",
      "N", "O *"]]
y = df["adb"]

y = y.fillna(y.mean())

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,
      random_state = 12)

from sklearn.ensemble import HistGradientBoostingRegressor
model = HistGradientBoostingRegressor()

model.fit(X, y)
y_pred = model.predict(X_test)

from sklearn.metrics import mean_absolute_error, mean_squared_error,
      r2_score
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("MAE: ", mae)
print("MSE: ", mse)
print("R2 score: ", r2)

import matplotlib.pyplot as plt

plt.figure(figsize = (12, 6))
plt.scatter(y_test, y_pred, alpha = 0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r',
      alpha = 0.5)
for actual, predicted in zip(y_test, y_pred):
    plt.plot([actual, actual], [actual, predicted], color = "grey",
      linestyle = ":", alpha = 0.5)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs. Predicted')
plt.show()

import seaborn as sns

residuals = y_test - y_pred

plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_pred, y=residuals, alpha = 0.5)
plt.axhline(0, color='red', linestyle='--', alpha = 0.5)
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs. Predicted")

plt.figure(figsize = (10, 6))
sns.histplot(residuals, bins = 50, kde = True, color = "skyblue", edgecolor
      = "black")
plt.xlabel("Residual")
plt.ylabel("Frequency")
plt.title("Distribution of Residuals with KDE Curve HistGradientBoosting
      Regressor")
plt.tight_layout()

```

```

plt.show()

import scipy.stats as stats
import matplotlib.pyplot as plt

plt.figure(figsize = (12, 6))
stats.probplot(residuals, dist="norm", plot=plt)
plt.title("Q-Q Plot of Residuals HistGradient Boosting Regressor")

import shap
import matplotlib.pyplot as plt
shap.initjs()
# Create the SHAP explainer
explainer = shap.Explainer(model.predict, X_test)

# Calculate SHAP values
shap_values = explainer(X_test)
shap.summary_plot(shap_values, X_test)
shap.summary_plot(shap_values, X_test, plot_type="bar")

shap.plots.waterfall(shap_values[0])

from catboost import CatBoostRegressor

model = CatBoostRegressor(verbose = 0)
model.fit(X_train, y_train)

y_pred2 = model.predict(X_test)

plt.figure(figsize = (12, 6))
plt.scatter(y_test, y_pred2, color = "blue", alpha = 0.5)
for actual, predicted in zip(y_test, y_pred2):
    plt.plot([actual, actual], [actual, predicted], color = "gray",
             linestyle = ":", alpha = 0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color
        = "red", linestyle = "-")

print("Mean Absolute Error: ", mean_absolute_error(y_test, y_pred2))
print("Mean Squared Error: ", mean_squared_error(y_test, y_pred2))
print("R2 score: ", r2_score(y_test, y_pred2))

residuals2 = y_test - y_pred2
plt.figure(figsize = (10, 6))
sns.histplot(residuals, bins = 50, kde = True, color = "skyblue", edgecolor
            = "black")
plt.xlabel("Residual")
plt.ylabel("Frequency")
plt.title("Distribution of Residuals with KDE Curve CatBoosting Regressor")
plt.tight_layout()
plt.show()

plt.figure(figsize = (12, 6))
stats.probplot(residuals2, dist="norm", plot=plt)
plt.title("Q-Q Plot of Residuals of CatBoosting Regressor")

```

5 Conclusion

To conclude, this project involved a diverse set of tasks that often required me to revisit fundamental concepts and develop a deeper understanding of the machine learning models being applied. I successfully generated insights for the OBOGS (On-Board Oxygen Generating System), conducted simulation for the BOS (Backup Oxygen System), and contributed to the ongoing research in the chemistry-based study. These experiences not only enhanced my technical proficiency but also deepened my appreciation for interdisciplinary problem-solving.

Looking ahead, we can explore more advanced machine learning approaches and experiment with deep neural networks (DNNs) to further improve prediction performance and reveal more complex patterns in the data.

On a personal note, I would like to thank Shri. Amrit Roy, SC "E" for his kind supervision of this project, and for giving me the opportunity to lead it.

For any questions/comments, I am available at adityasaroha456@gmail.com.