Optimizing pre-reforming for quality r-SOC syngas preparation using artificial intelligence (AI) based machine learning (ML)

Vidhisha Agarwal

210107094

Submission Date: April 25, 2024

Final Project submission

Course Name : Applications of AI and ML in chemical engineering

Course Code: CL653

Contents

Executive Summary

Problem: The project addresses the challenge of optimizing the preparation of clean syngas for Solid Oxide Cells (SOCs) in the context of a green hydrogen economy.

Proposed Solution: Leveraging advanced artificial intelligence (AI) and machine learning (ML) techniques to develop a model that evaluates and optimizes the pre-reforming process of syngas for SOCs.

Methodologies: The project employs Design of Experiments (DoE), and regression modelling to assess different syngas constellations, investigate thermochemistry, and develop the AI/ML model.

Expected Outcomes: The project aims to provide guidelines for achieving high-quality syngas suitable for continuous and efficient operation of SOCs, contributing to the advancement of a sustainable hydrogen economy. Additionally, the deployment of the AI/ML model is expected to enhance efficiency, reduce emissions, and foster innovation in renewable energy technologies.

Overall, the project endeavours to address a crucial challenge in renewable energy research and contribute to global efforts towards mitigating climate change and achieving a greener future.

Introduction

Background: Solid oxide cells (SOCs) play a crucial role in the transition to a hydrogen-based economy, as they offer efficient energy conversion and can operate reversibly, serving both as fuel cells and electrolyzes. However, one of the key challenges in utilizing SOCs effectively is ensuring the supply of high-quality syngas, a mixture of hydrogen, carbon monoxide, and other gases, as their fuel source. Traditional methods of syngas preparation often involve complex thermochemical processes prone to inefficiencies and safety hazards.

This project aims to address this challenge by leveraging advanced artificial intelligence (AI) and machine learning (ML) techniques. By optimizing the pre-reforming process of various syngas constellations, we seek to enhance the performance and efficiency of SOCs, contributing to the advancement of a sustainable hydrogen economy.

Specific Problem Statement: The specific problem addressed in this project is the optimization of the pre-reforming process to produce clean syngas suitable for SOCs. The pre-reforming process aims to convert carbonaceous feedstocks into syngas, which can then be

utilized by SOCs. However, achieving the desired fuel outlet conditions, particularly a fuel outlet temperature of 500°C, while maintaining safety and efficiency, presents significant challenges. Factors such as gas composition, flow rates, and catalyst properties influence the pre-reforming process and impact SOC performance.

Reference: Optimising pre-reforming for quality r-SOC syngas preparation using artificial intelligence (AI) based machine learning (ML) - ScienceDirect

Objectives:

- Develop an AI/ML model to evaluate and optimize the pre-reforming process of clean syngas for solid oxide cells.
- Utilize Design of Experiments (DoE) to assess different syngas constellations and identify optimal operating parameters.
- Validate the AI/ML model using empirical data and numerical simulations, ensuring accuracy and reliability.
- Defining the operating parameters for three distinct syngas configurations to achieve 500 °C fuel outlet conditions.
- Provide guidelines for achieving high-quality, pure syngas suitable for continuous and efficient operation of reversible solid oxide cells.

Methodology

Data Source: Since the authentic data for this project is not available on unlicensed platform, AI generated data has been used.

The research paper used for this project provided a set of 4 data (appendix table 1). So, the datapoints used in the project uses this data as a benchmark by considering them in the range near to the sample data. The code (appendix code 1) generates a set of linearly spaced data bound in the intervals that resonate with the sample data. Additionally, 4% noise has been added to the linearly spaced data to generate the datapoints that can be used for development of model and analysing its performance.

Data Preprocessing: Techniques to be used for cleaning and preparing data for analysis.

1.Null value detection: Analysing null values in the Data Frame, replacing them with the mean of each column, and then confirming that the replacements were successful. But the data frame does not have any null values as the data is synthetically generated.

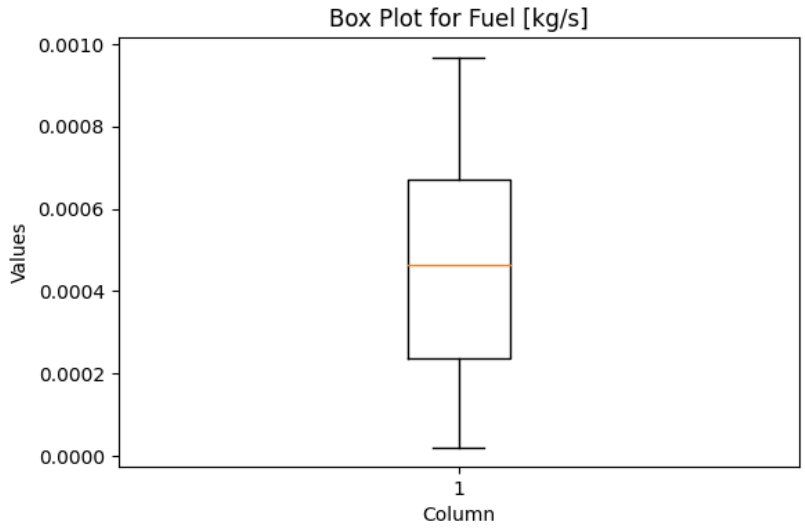
2.Outlier removal: Z-score normalization has been used for outlier removal.

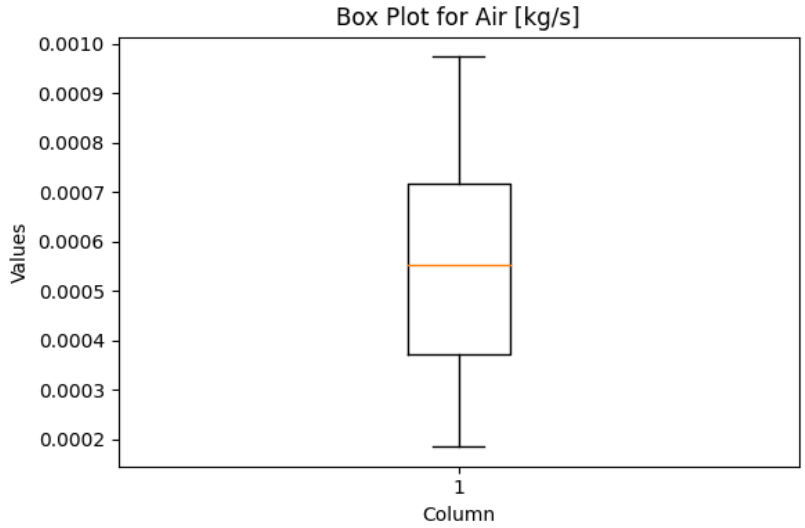
Number of rows removed: 2.

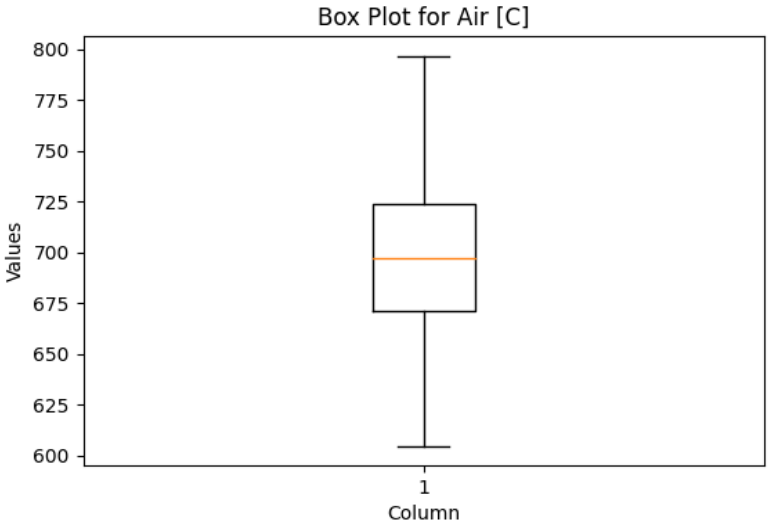
3.Box Plot: Box plots are a versatile tool for understanding the distributional characteristics of a dataset and for identifying potential outliers or differences between groups. They complement

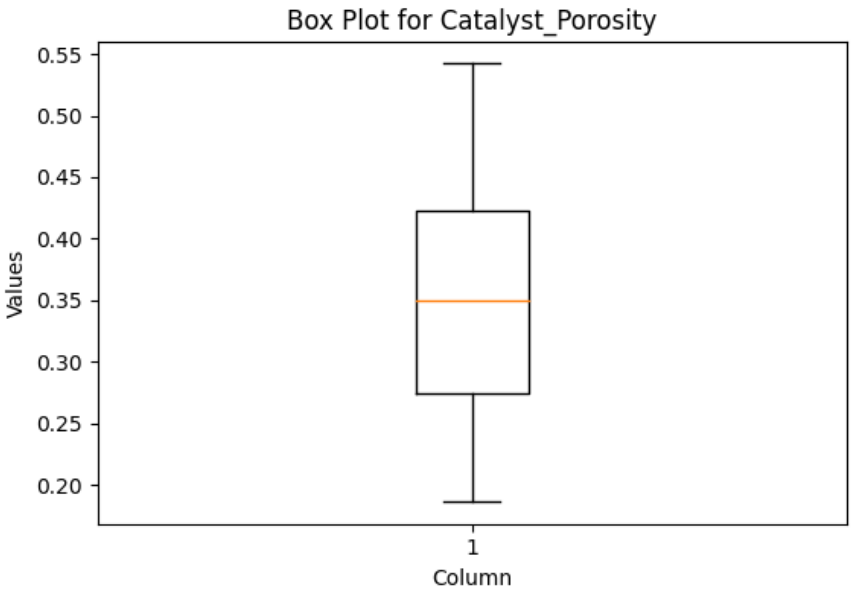
other statistical measures and visualization techniques, providing a clear and intuitive summary of the data's distribution.

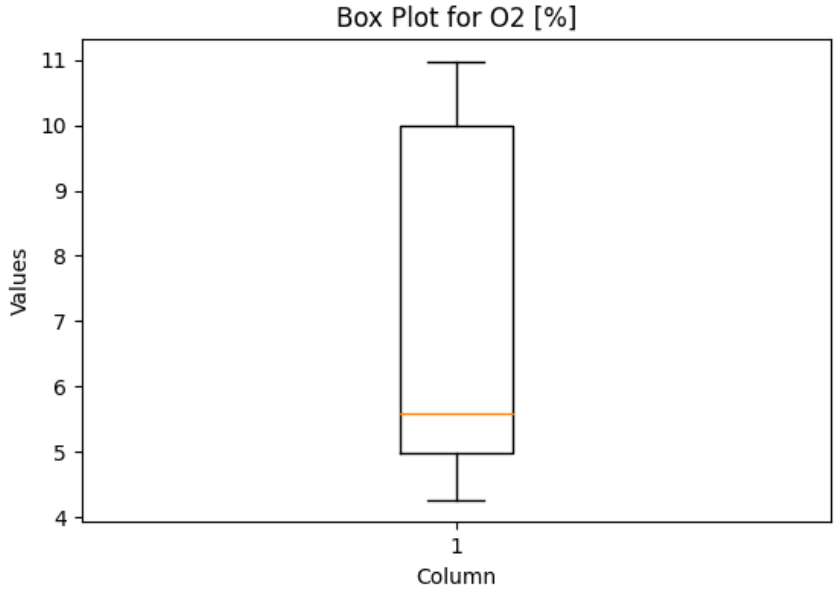
The following are the box plots for the data set.

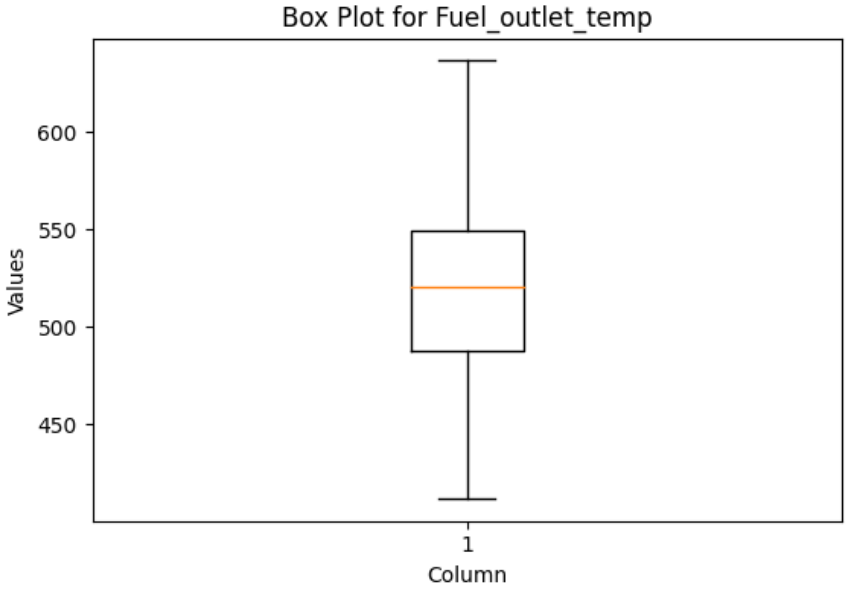




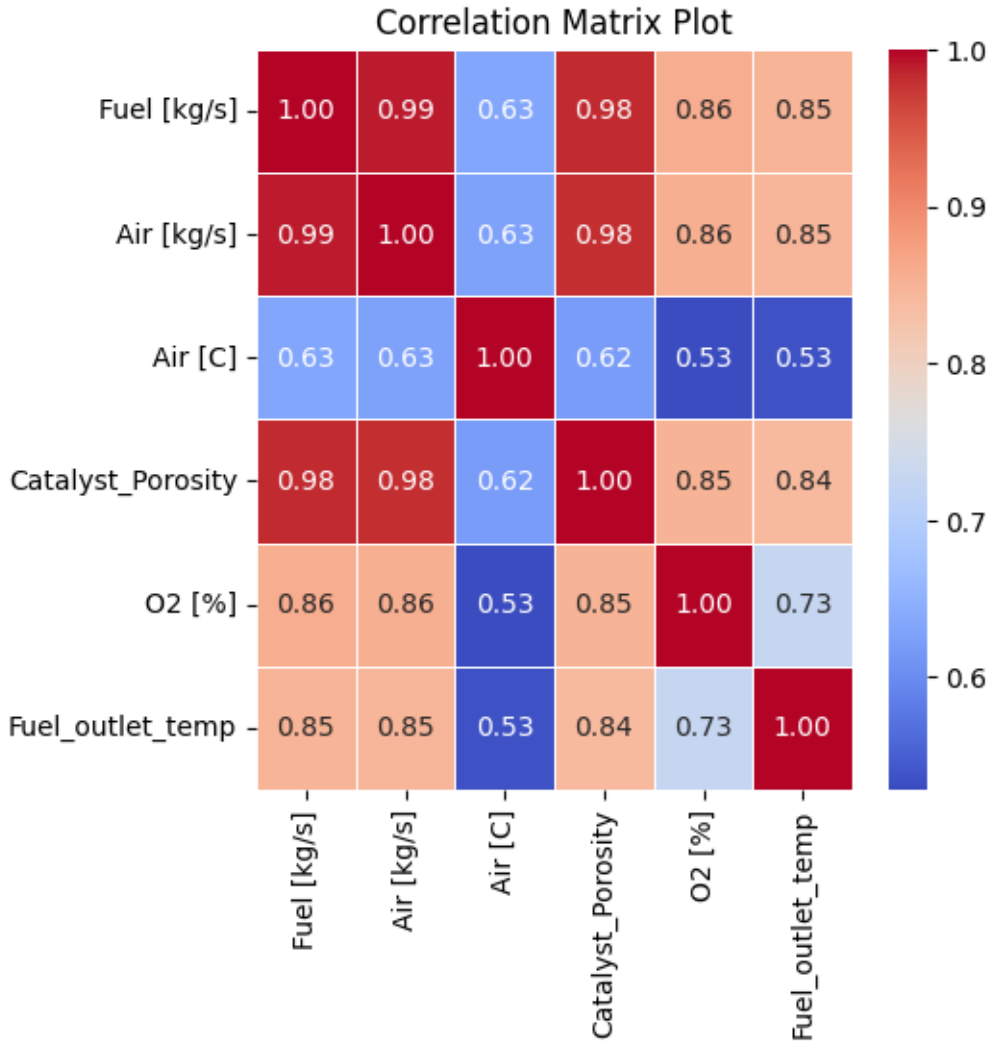



Box Plot for Catalyst_Porosity
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
Values
1
Column


Box Plot for O2 [%]
11
10
9
8
7
6
5
4
Values
1
Column


Box Plot for Fuel_outlet_temp
600
550
500
450
Values
1
Column

4. Correlation matrix: Based on the correlation matrix, here's a summary of the correlations between "Fuel_outlet_temp" and other variables:



Fuel [kg/s]: - Positively correlated with "Fuel_outlet_temp" with a correlation coefficient of approximately 0.85. This indicates that higher fuel flow rates are associated with higher outlet temperatures.

Air [kg/s]: Positively correlated with "Fuel_outlet_temp" with a correlation coefficient of approximately 0.85. Similar to fuel flow rates, higher air flow rates are associated with higher outlet temperatures.

Air [C] (temperature): Positively correlated with "Fuel_outlet_temp" but to a lesser extent compared to flow rates, with a correlation coefficient of approximately 0.73. Higher inlet air temperatures are associated with higher outlet temperatures, but the correlation is not as strong as with flow rates.

Catalyst Porosity: Positively correlated with "Fuel_outlet_temp" with a correlation coefficient of approximately 0.84. Higher catalyst porosity is associated with higher outlet temperatures.

O2 [%] (oxygen concentration): Positively correlated with "Fuel_outlet_temp" with a correlation coefficient of approximately 0.73. Higher oxygen concentrations are associated with higher outlet temperatures.

From the above relations we can say that all variables have positive correlations with "Fuel_outlet_temp", indicating that increasing any of these variables is likely to lead to higher outlet temperatures. However, the strength of correlation varies, with fuel and air flow rates having the strongest correlations, followed by catalyst porosity, oxygen concentration, and finally, inlet air temperature.

5. Separating Features and Target Variable: The features (independent variables) are extracted from the dataset by dropping the column 'Fuel_outlet_temp'. These features represent the input variables used to predict the target variable. The target variable ('Fuel_outlet_temp') is extracted and assigned to the variable ‘y’.

6.Splitting the Data: The dataset is split into training and testing sets using the ‘train_test_split’ function from scikit-learn. This splitting ensures that a portion of the data is held out as a separate test set, allowing for unbiased evaluation of the model's performance. In this case, 80% of the data is allocated for training, while the remaining 20% is reserved for testing. The random_state parameter ensures reproducibility of the split, meaning the same split will be generated each time the code is run with the same random state value.

7.Data scaling:

For the Support Vector Machine (SVM) model, I've utilized standardization to preprocess the data. This choice was made because SVM is sensitive to the scale of features, and standardization ensures that all features contribute equally to the decision boundary. By standardizing the data to have a mean of 0 and a standard deviation of 1, I enhance the SVM's performance and make it less sensitive to the scale of individual features.

In the case of the K-Nearest Neighbors (KNN) regression model, I opted for normalization of the data. This decision stems from KNN's reliance on distance computations between data points, which can be heavily influenced by the scale of features. Normalization rescales the features to a fixed range, ensuring that all features contribute equally to the distance metric. By normalizing the data, I prevent features with larger scales from dominating the distance computation, resulting in a more balanced influence of all features on the predictions.

Lastly, for the Multiple Linear Regression model, I again chose standardization for data preprocessing. While multiple regression is less sensitive to feature scale compared to SVM and KNN, standardization still offers benefits. It ensures that the coefficients associated with each feature represent the change in the target variable for a one-standard-deviation change in the corresponding feature, making the coefficients more interpretable and facilitating model comparison. Additionally, standardization can improve the convergence speed of optimization algorithms used to fit the model.

Model Architecture: Based on the analysis, a regression model approach is recommended for capturing the nonlinear relationships between the input parameters and the output (fuel outlet temperatures). To fully incorporate interactions between parameters and the nonlinear connections between variables and the output, it's suggested to consider cubic and quadratic factors. A multi-regression model is proposed, where the real oxygen content is represented as dummy variables based on chosen factor levels (low, medium, and high), allowing for the consideration of categorical variables like gas composition. This approach avoids data fragmentation by using all the data in a single regression, resulting in more accurate estimates compared to using separate models for different subsets of data. The following models are suggested for consideration:

1. Support Vector Machine (SVM): SVM can capture nonlinear relationships in the data by mapping the input features into a higher-dimensional space. It can be effective for regression tasks, especially when there are complex relationships between the input variables and the output.
2. K-Nearest Neighbor (KNN) Algorithm: KNN is a non-parametric algorithm that can capture complex relationships between variables without making assumptions about the underlying data distribution. It considers the local neighborhood of data points to make predictions, making it suitable for capturing nonlinear relationships.
3. Multi-Regression Model: The proposed multi-regression model allows for the incorporation of cubic and quadratic factors, as well as the consideration of categorical variables through dummy variables. By using all the data in a single regression, this model can provide more accurate estimates and avoid the need for data fragmentation.

Each of these models has its strengths and weaknesses, and the choice between them should be based on factors such as the complexity of the relationships in the data, computational efficiency, and interpretability of the results.

Tools and Technologies: For the development of the project, a combination of programming languages, libraries, and tools can be utilized. Here's a list of some commonly used ones:

1. Python: A versatile programming language with extensive libraries for data manipulation, analysis, and machine learning.
2. NumPy: A fundamental package for numerical computing in Python, providing support for arrays, matrices, and mathematical functions.
3. pandas: A powerful data analysis and manipulation library for Python, offering data structures like DataFrame for handling structured data.
4. scikit-learn: A popular machine learning library in Python, providing a wide range of algorithms for classification, regression, clustering, and more, along with tools for model selection and evaluation.
5. matplotlib: A comprehensive plotting library for Python, used for creating static, interactive, and animated visualizations in various formats.
6. seaborn: Built on top of matplotlib, seaborn provides a high-level interface for drawing attractive and informative statistical graphics.
7. Jupyter Notebook / JupyterLab: Interactive computing environments that allow you to create and share documents containing live code, equations, visualizations, and narrative text.
8. GitHub: A platform for version control using Git, facilitating collaboration, code review, and project management for software development projects.
9. SciPy: A library for scientific computing in Python, providing functions for optimization, integration, interpolation, and more.
10. GridSearchCV: A utility in scikit-learn for hyperparameter tuning using grid search with cross-validation.

These tools and technologies provide a comprehensive ecosystem for data preprocessing, analysis, modeling, and visualization in the context of machine learning and data science tasks.

Implementation Plan

Development Phases: Breakdown of the project into phases/stages with timelines.

Here's a breakdown of the project into phases/stages along with estimated timelines:

Phase 1: Data Exploration and Preprocessing

- Timeline: 1-2 weeks
- Tasks:

Explore the dataset to understand its structure, features, and distributions.

Handle missing values, outliers, and data inconsistencies.

Perform exploratory data analysis (EDA) to gain insights into relationships between variables.

Preprocess the data, including feature scaling, encoding categorical variables, and splitting into training and testing sets.

Phase 2: Model Selection and Development

- Timeline:2-3 weeks
- Tasks:

Select appropriate regression models based on project requirements and analysis recommendations.

Develop and implement the selected models, including SVM, KNN regression, and multi-regression with quadratic and cubic factors.

Tune hyperparameters for SVM and KNN regression using GridSearchCV.

Train the models on the training data and evaluate their performance using appropriate metrics.

Phase 3: Model Evaluation and Refinement

- Timeline: 1-2 weeks
- Tasks:

Evaluate the performance of each model using metrics like mean squared error (MSE), root mean squared error (RMSE), and R-squared.

Identify areas for improvement and refine the models accordingly.

Investigate the impact of different factors and features on model performance through sensitivity analysis.

Optimize model parameters and fine-tune the models for better accuracy and generalization.

Phase 4: Documentation and Reporting

- Timeline: 1 week
- Tasks:

Document the entire project, including data preprocessing steps, model development, evaluation results, and conclusions.

Prepare visualizations, charts, and tables to summarize key findings and insights.

Write a detailed report outlining the methodology, results, limitations, and recommendations.

Present the findings to stakeholders, highlighting the strengths and limitations of the developed models and providing recommendations for future work.

.

Model Training: For training the regression models, including Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Multiple Linear Regression, the following strategies can be employed:

1. Algorithm Selection: For model training, we'll employ three regression algorithms: Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Multiple Linear Regression. SVM effectively captures nonlinear relationships, KNN considers local data neighborhoods, and Multiple Linear Regression is suitable for linear relationships.

2. Data Preprocessing: Standardization or normalization of features to ensure that all variables contribute equally to the model. Handling missing values, outliers, and categorical variables appropriately.

3.Parameter Tuning:

- For SVM and KNN, hyperparameter tuning can be performed using techniques like GridSearchCV to find the optimal values for parameters such as C (for SVM), gamma (for SVM with RBF kernel), and k (for KNN).
- In addition to hyperparameters, kernel selection is crucial for SVM, with options including linear, polynomial, and radial basis function (RBF) kernels.
- For Multiple Linear Regression, feature selection techniques such as backward elimination or regularization methods like Lasso or Ridge regression can be used to improve model performance and interpretability.

4. Cross-Validation:

- Utilize cross-validation techniques such as k-fold cross-validation to assess model performance and generalize well to unseen data.

- Cross-validation can help in estimating the model's performance on different subsets of the data and can provide insights into the model's stability and reliability.

Model Evaluation: For model evaluation, we'll employ the following metrics and methods:

1. Metrics:

- Mean Squared Error (MSE): Measures the average squared difference between the actual and predicted values. Lower values indicate better model performance.
- Root Mean Squared Error (RMSE): The square root of the MSE, providing a measure of the average magnitude of errors. It is interpretable in the same units as the target variable.
- R-squared (R2): Represents the proportion of the variance in the target variable that is predictable from the independent variables. R2 values range from 0 to 1, with higher values indicating better model fit.

2. Methods:

- Cross-Validation: Utilize k-fold cross-validation to assess model performance across different subsets of the data. This technique provides an estimate of the model's generalization performance and helps detect overfitting.
- Residual Analysis: Analyze the residuals (the differences between actual and predicted values) to assess the model's performance systematically. Visualizations like residual plots can identify patterns or heteroscedasticity in the errors.
- Comparative Analysis: Compare the performance of different models using the chosen evaluation metrics. This allows for selecting the best-performing model for deployment based on its ability to accurately predict fuel outlet temperatures.

By employing these metrics and methods, we can comprehensively evaluate the performance of the regression models and make informed decisions regarding their suitability for the task at hand.

Testing and Deployment

Testing Strategy: The model will be tested against unseen data using the following testing strategy:

1. Holdout Method:

- Split the dataset into training and testing sets, with a larger portion allocated for training (e.g., 80%) and the remaining portion for testing (e.g., 20%).

- Train the regression models (SVM, KNN, and Multiple Linear Regression) using the training set.

- Evaluate the trained models' performance on the unseen testing set using evaluation metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R2).

2. Cross-Validation:

- Utilize k-fold cross-validation, where the dataset is divided into k subsets (folds).

- Train the models k times, each time using k-1 folds for training and the remaining fold for validation.

- Compute the average performance metrics across all k iterations to obtain a more robust estimate of the models' generalization performance.

3. Out-of-Time Testing:

- Reserve a holdout period (e.g., the most recent time period) as a test set, representing unseen future data.

- Train the regression models on historical data preceding the holdout period.

- Evaluate the models' performance on the holdout test set to simulate real-world deployment scenarios and assess their ability to generalize to future data.

4. External Validation:

- Collect additional external datasets, if available, from similar or related domains.

- Train the regression models on the original dataset and evaluate their performance on the external validation dataset.

- This approach validates the models' performance across different datasets and ensures their applicability to diverse data sources.

By employing these testing strategies, we can rigorously evaluate the regression models' performance on unseen data and assess their ability to generalize to new observations, thereby ensuring their reliability and effectiveness in real-world scenarios.

Deployment Strategy: The deployment strategy for the regression models involves a systematic approach to ensure scalability, performance, and maintenance for real-world use. Here's a plan outlining the deployment strategy:

1. Infrastructure Setup:

- Establish a scalable and robust infrastructure for deploying regression models. This may include cloud-based platforms like AWS, Azure, or Google Cloud, or on-premises servers depending on the organization's requirements and resources.

2. Model Deployment:

- Deploy the trained regression models (SVM, KNN, and Multiple Linear Regression) as web services or APIs using frameworks like Flask, Django, or FastAPI.

- Containerize the models using Docker for portability and easy deployment across different environments.

- Ensure the models are deployed in a reliable and fault-tolerant manner to handle varying levels of incoming requests.

3. Scalability:

- Design the deployment architecture to be scalable, allowing for horizontal scaling to accommodate increased workload and demand.

- Utilize load balancers and auto-scaling mechanisms to dynamically adjust computing resources based on traffic patterns and usage.

4. Performance Optimization:

- Optimize model inference speed and latency by leveraging techniques such as model quantization, model pruning, and hardware acceleration (e.g., GPUs).

- Implement caching mechanisms to store frequently accessed data and precomputed results, reducing the computational overhead of model inference.

5. Monitoring and Logging:

- Implement robust monitoring and logging systems to track the performance and health of the deployed models.

- Monitor key metrics such as response time, throughput, error rates, and resource utilization to identify performance bottlenecks and ensure efficient operation.

6. Security:

- Implement stringent security measures to protect the deployed models from unauthorized access, data breaches, and cyber threats.

- Utilize encryption, authentication, and access control mechanisms to safeguard sensitive data and ensure compliance with privacy regulations.

7. Maintenance and Updates:

- Establish a regular maintenance schedule to monitor and update the deployed models as needed.

- Incorporate feedback loops to collect user feedback and model performance metrics for continuous improvement.

- Plan for regular retraining of the models using updated data to maintain their accuracy and relevance over time.

8. Documentation and Training:

- Provide comprehensive documentation and user guides for developers, data scientists, and end-users on how to use and interact with the deployed models.

- Offer training and support to stakeholders to ensure smooth adoption and utilization of the deployed solution.

By following this deployment strategy, we can ensure that the regression models are deployed effectively for real-world use, offering scalability, performance, and maintainability while meeting the organization's objectives and requirements.

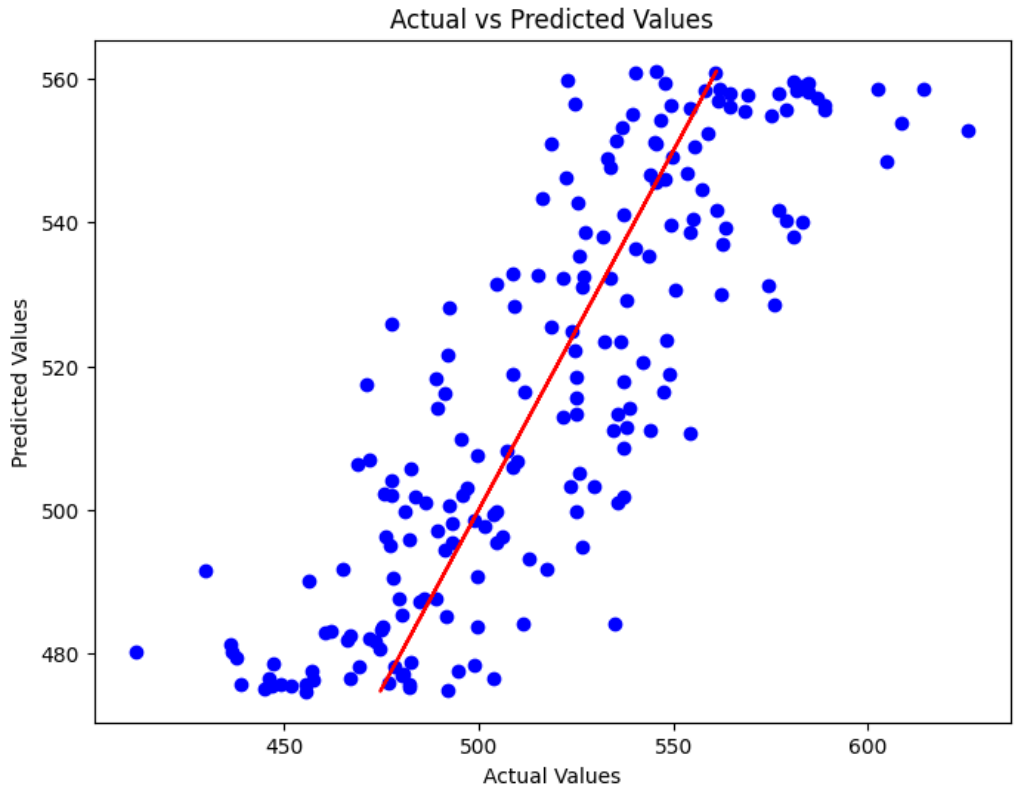
Ethical Considerations: Deploying machine learning models, such as regression models, entails ethical considerations to ensure fair and responsible use. Key aspects include addressing bias and fairness, ensuring transparency and interpretability, safeguarding privacy and data protection, fostering accountability and responsibility, identifying ethical use cases, obtaining informed consent, mitigating bias, and engaging stakeholders. By addressing these

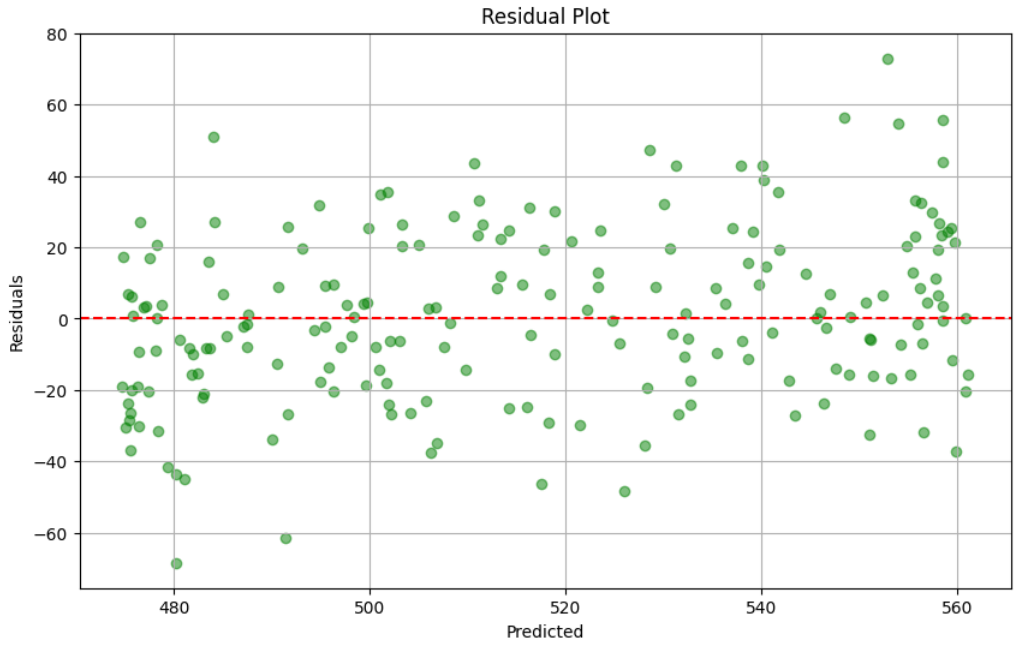
considerations, organizations can deploy regression models ethically, promoting trust, transparency, and societal benefit while minimizing potential harm.

Results and Discussion

SVM Model without hyperparameter tuning:
The code used here trains an SVM model without performing hyperparameter tuning. It then evaluates the model's performance on the test set and visualizes the results, including plotting the actual vs. predicted values and the residuals.

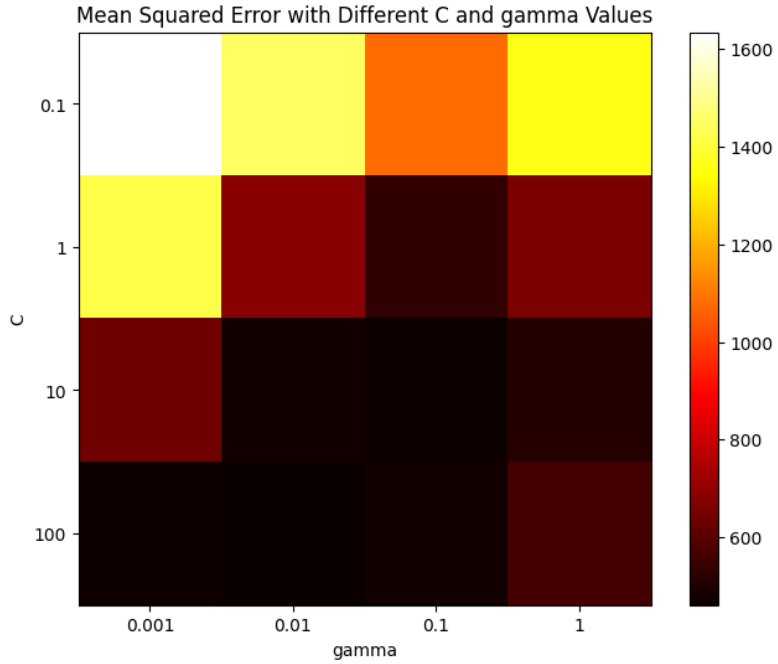
```
Performance Metrics:
Mean Squared Error: 568.1525060830477
Root Mean Squared Error: 23.835949867438632
R-squared: 0.6748379671041964
```





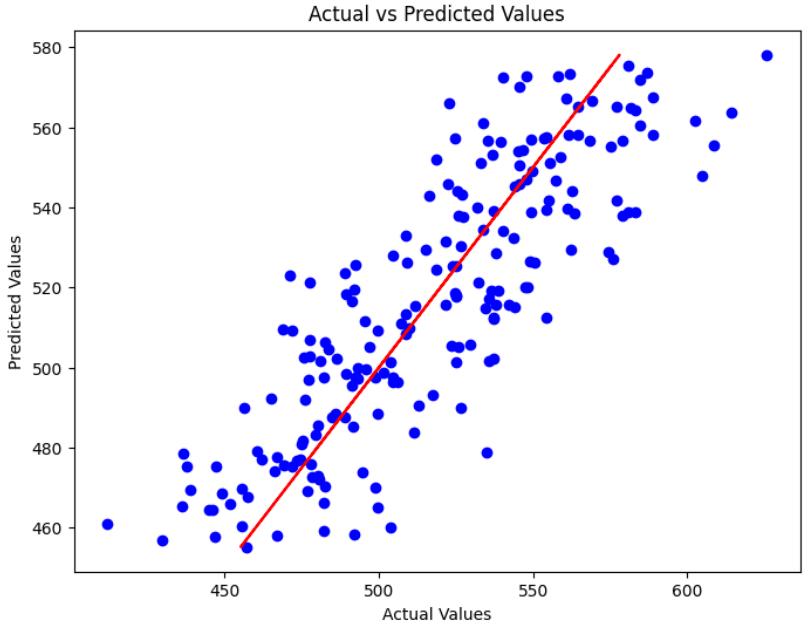
SVM Model with hyperparameter tuning:
Here I am performing grid search cross-validation to tune hyperparameters for a Support Vector Regression (SVR) model, and then evaluating the model's performance using various metrics and visualizations.

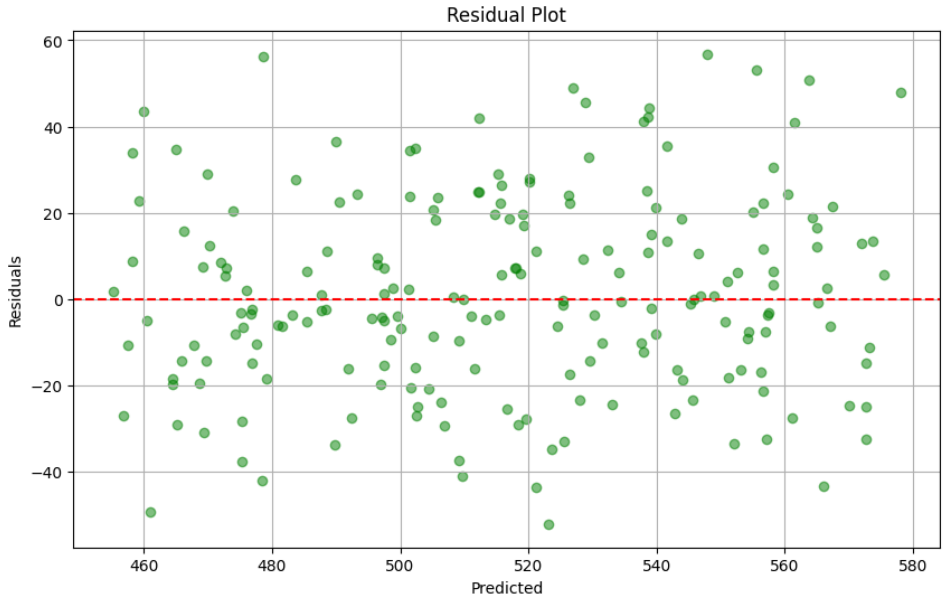
The code does the following:
1. Defines a grid of hyperparameters (`param_grid`) for the SVR model.
2. Initializes the SVM model.
3. Performs grid search cross-validation to find the best hyperparameters using mean squared error (MSE) as the scoring metric.
4. Plots the effect of different combinations of hyperparameters on the model's performance using a heatmap.
5. Initializes and trains the SVR model with the best parameters found.
6. Makes predictions on the test set and evaluates the model using MSE, root mean squared error (RMSE), and R-squared.
7. Prints the performance metrics.
8. Plots the actual vs. predicted values and the residuals.




```
Best Parameters: {'C': 100, 'gamma': 0.01}
Best Score (negative MSE): -460.8350569759543
```

```
Performance Metrics:
Mean Squared Error: 513.715290733188
Root Mean Squared Error: 22.665288234063734
R-squared: 0.7059931858506217
```





In summary, the SVM model with hyperparameter tuning performed better in terms of MSE, RMSE, and R-squared compared to the model without hyperparameter tuning. Therefore, the SVM model with hyperparameter tuning is the better choice for model development. My data consists of 1000 datapoints and 5 columns so other factors such as computational complexity and interpretability will not have a major effect on the above choice.

KNN Regression

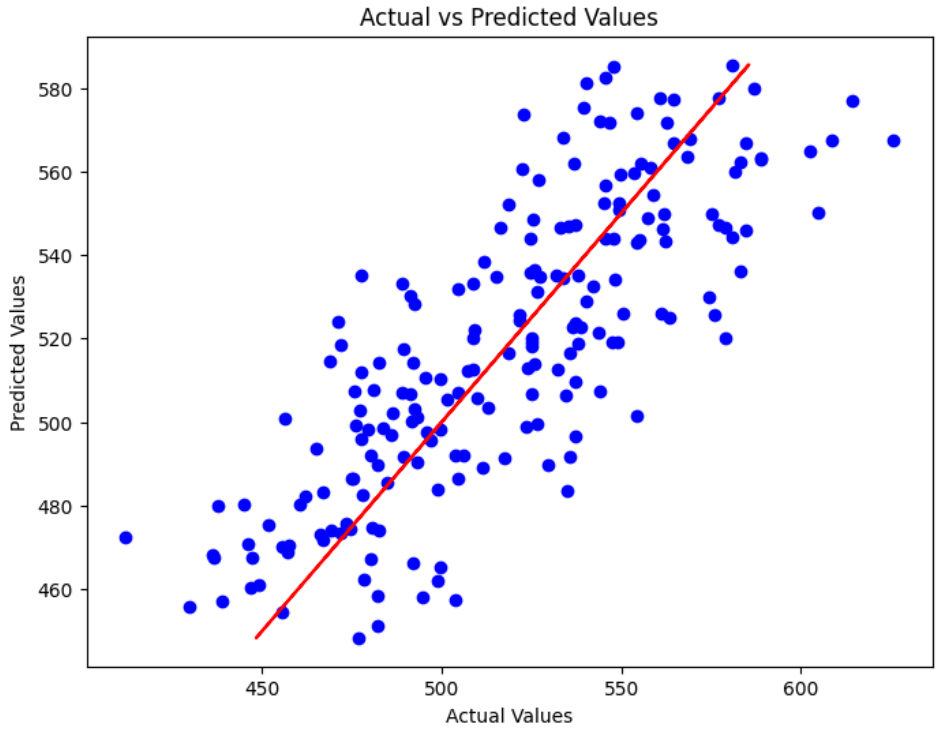
In KNN Regression, instead of predicting a class label as in classification tasks, the model predicts a continuous value based on the average or weighted average of the values of its k nearest neighbors.

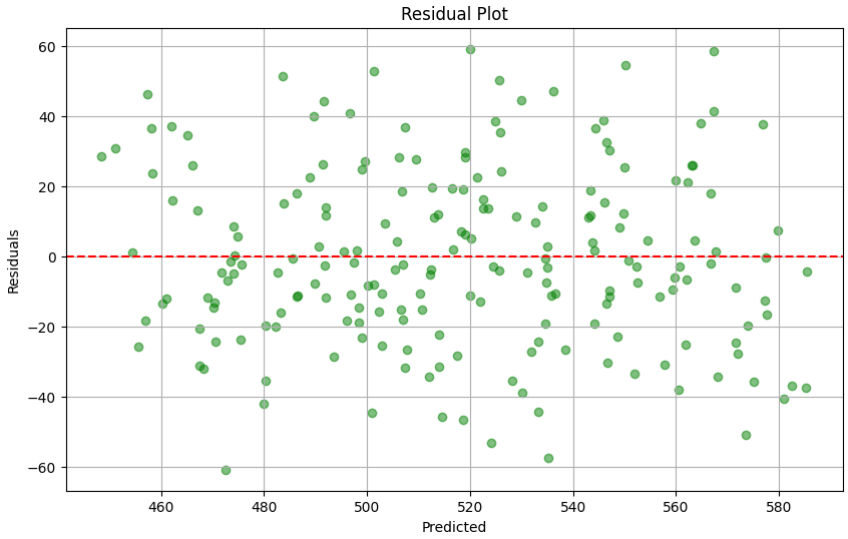
KNN Regression without hyperparameter tuning:

In the code for this:

- We initialize the KNN model with specific hyperparameters (**n_neighbors** and **weights**).
- We train the model on the training data.
- We make predictions on the test data.
- We evaluate the model's performance using mean squared error (MSE), root mean squared error (RMSE), and R-squared.
- We plot the actual vs. predicted values and the residuals to visualize the model's performance.

```
Performance Metrics:
Mean Squared Error: 638.365824434622
Root Mean Squared Error: 25.265902406892614
R-squared: 0.6346538526505661
```





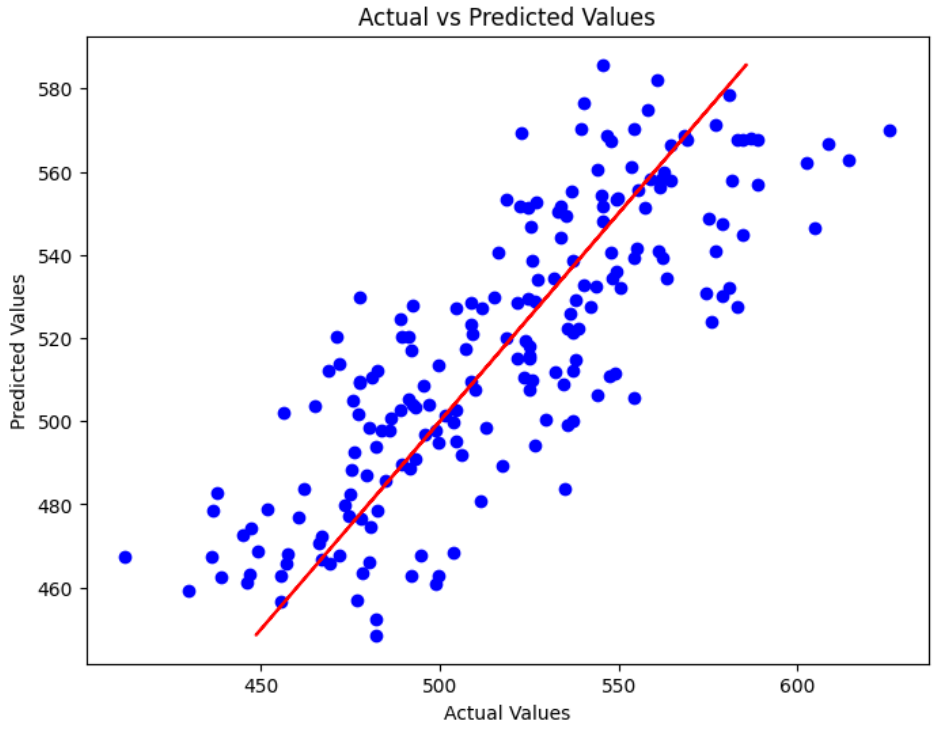
KNN Regression with hyperparameter tuning:
The code performs grid search cross-validation to tune hyperparameters for a KNN regression model. It then evaluates the model's performance on a test set and visualizes the results, including plotting the actual vs. predicted values and the residuals.

Here is a summary of the steps:
1. Define Hyperparameter Grid: Specify a grid of hyperparameters for the KNN model, including the number of neighbors (`n_neighbors`) and the weight function used in prediction (`weights`).
2. Initialize KNN Model: Create an instance of the KNN regression model.
3. Perform Grid Search Cross-Validation: Use `GridSearchCV` to search for the best combination of hyperparameters using cross-validation. The scoring metric used is negative mean squared error (`neg_mean_squared_error`).

4. Plot Hyperparameter Performance: Visualize the effect of different combinations of hyperparameters on the model's performance using a heatmap.
5. Train Best Model: Initialize and train the KNN model with the best parameters found during grid search.
6. Make Predictions and Evaluate: Make predictions on the test set using the best model and evaluate its performance using mean squared error (MSE), root mean squared error (RMSE), and R-squared.
7. Plot Results: Plot the actual vs. predicted values and the residuals to visualize the model's performance.


```
Best Parameters: {'n_neighbors': 9, 'weights': 'uniform'}
Best Score (negative MSE): -511.23710963953016


Performance Metrics:
Mean Squared Error: 599.0776719042998
Root Mean Squared Error: 24.476063243591682
R-squared: 0.6571390400055482
```



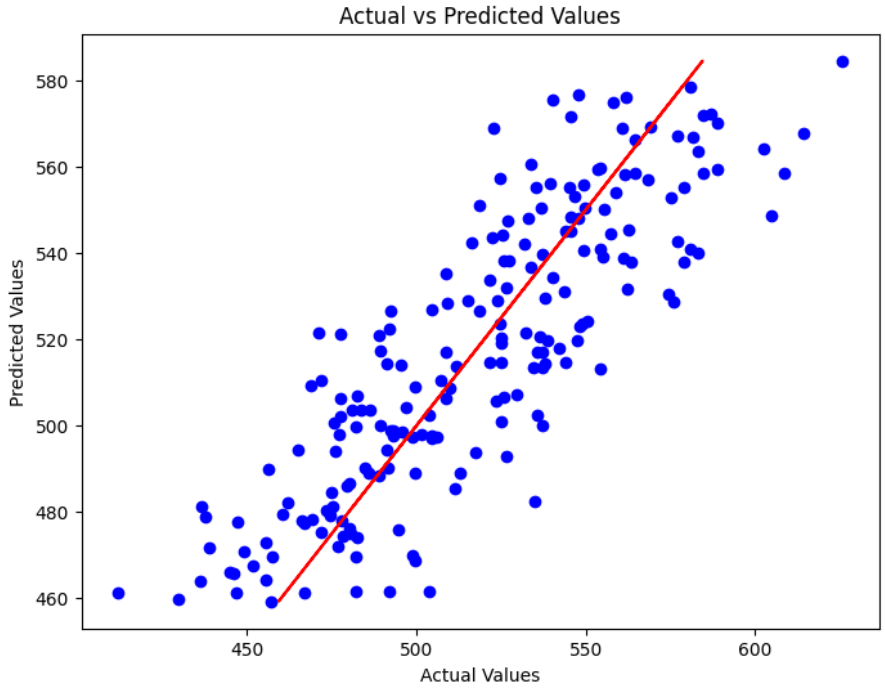
Based on performance metrics, the KNN model with hyperparameter tuning performs better than the model without hyperparameter tuning. It has lower MSE and RMSE, indicating better predictive performance, and a higher R-squared value, indicating better fit to the data. Therefore, the KNN model with hyperparameter tuning is preferred for this task.

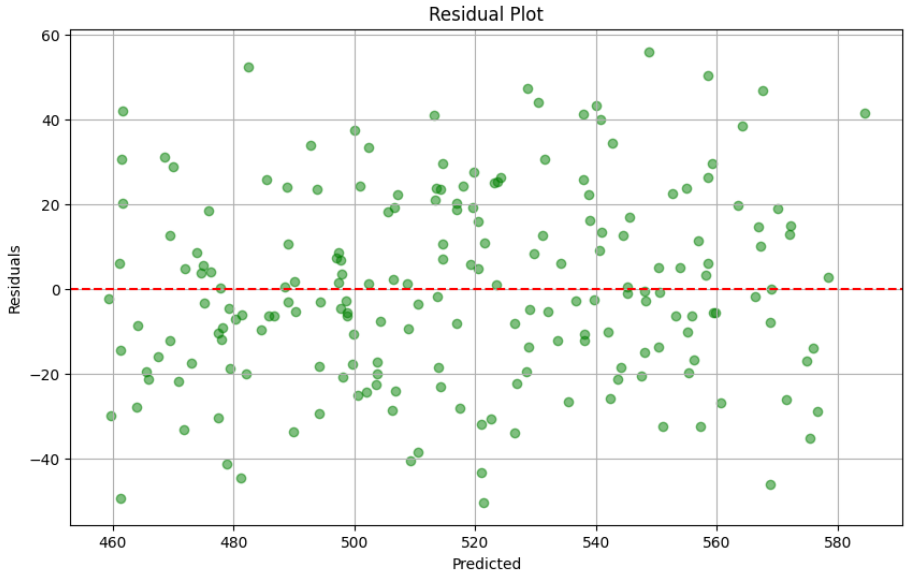
Multilinear regression without Hyperparameter Tuning

It directly fits the Multi Linear Regression model to the training data without hyperparameter tuning. It then evaluates the model's performance on the test set and visualizes the results, including plotting the actual vs. predicted values and the residuals.

```
Multi Regression Equation:
y = (83833.77 * X_1) + (55126.98 * X_2) + (-0.01 * X_3) + (34.28 * X_4)
+ (-0.47 * X_5) + 451.31

Performance Metrics:
Mean Squared Error: 507.25377053499346
Root Mean Squared Error: 22.522294965988557
R-squared: 0.7096912088651239
```



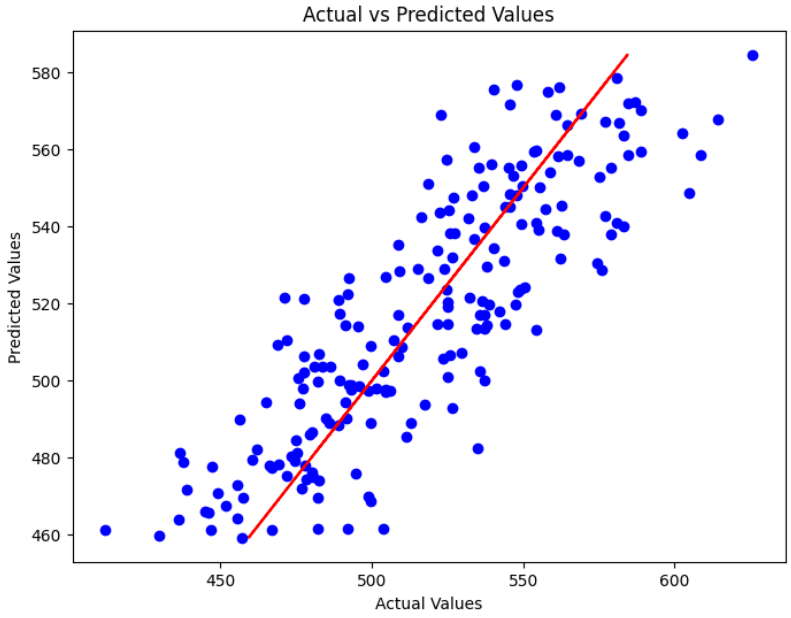


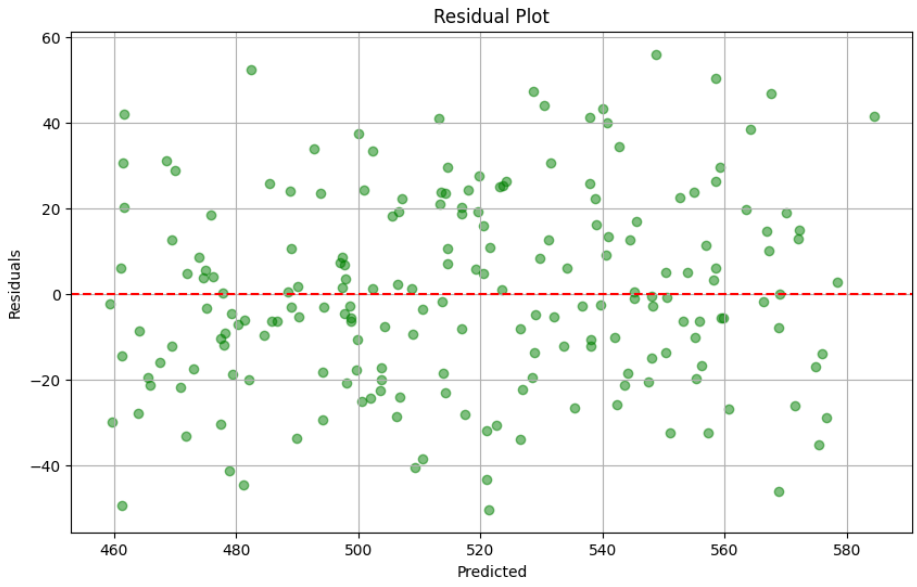
Multilinear regression with Hyperparameter Tuning

It performs hyperparameter tuning for a Linear Regression model using GridSearchCV. It then evaluates the best model's performance on a test set and visualizes the results, including plotting the actual vs. predicted values and the residuals.

```
Multi Regression Equation:
y = (83833.77 * X_1) + (55126.98 * X_2) + (-0.01 * X_3) + (34.28 * X_4)
+ (-0.47 * X_5) + 451.31

Performance Metrics:
Mean Squared Error: 507.25377053499346
Root Mean Squared Error: 22.522294965988557
R-squared: 0.7096912088651239
```





In this case, both multilinear regression models, with and without hyperparameter tuning, demonstrate identical performance metrics. Therefore, there is no clear distinction in predictive accuracy between the two. Since the model without hyperparameter tuning achieves the same performance without the added computational cost, it should be preferred for its simplicity and efficiency.

Best Model Evaluation: Based on the performance matrices of the above # models, we can conclude that:

1. Multi Regression appears to have the lowest Mean Squared Error and Root Mean Squared Error, indicating better predictive performance compared to the other two models.

2. Multi Regression also has the highest R-squared value, indicating that it explains the variance in the data better than the other models.

Comparative Analysis: In the comparative analysis, the model's performance will be evaluated against the results obtained from a detailed Computational Fluid Dynamics (CFD) model, which provides comprehensive 3D thermochemical simulations. Specifically, metrics such as mean squared error (MSE), root mean squared error (RMSE), and R-squared (R2) will be calculated to compare the accuracy of the ML model's predictions with the CFD-calculated results.

The comparison will focus on scenarios where detailed distributions of temperature, species concentrations, or pressure are of concern. This analysis will help determine the ML model's ability to provide rapid predictions that can serve as a screening tool in situations where detailed simulations are computationally expensive or time-consuming.

The comparative analysis will provide insights into the ML model's strengths and limitations compared to detailed CFD simulations, informing its potential use as a predictive tool in engineering and scientific domains.

Challenges and Limitations: During the project, several challenges were encountered, along with limitations of the proposed solution:

1. Data Quality and Availability:

- Limited availability of high-quality data posed a challenge, leading to potential biases and limitations in model performance.

2. Model Complexity and Interpretability:

- Complex regression models such as SVM and KNN posed challenges in terms of interpretability, making it difficult to explain model decisions to stakeholders.

- Balancing model complexity with interpretability was a trade-off, as more complex models often yielded better performance but were harder to interpret.

3. Hyperparameter Tuning and Optimization:

- Tuning hyperparameters for SVM and KNN models required significant computational resources and time, especially when exploring a large hyperparameter space.

- Optimal hyperparameter selection was crucial for achieving the best model performance but often required extensive experimentation and tuning.

4. Generalization and Overfitting:

- Ensuring that the trained regression models generalized well to unseen data was challenging, particularly in cases where the model's exhibited signs of overfitting.

5. Evaluation Metrics and Performance Benchmarking:

- Selecting appropriate evaluation metrics and establishing performance benchmarks for regression models was challenging, particularly in cases where domain-specific metrics were required.

- Comparing model performance against baseline models or industry standards posed challenges in benchmarking and performance evaluation.

Despite these challenges, the proposed solution aimed to address the project's objectives effectively. However, it's essential to acknowledge the limitations inherent in any machine learning solution and continually strive to improve model robustness, interpretability, and performance in future iterations.

Conclusion and Future Work

In conclusion, this project endeavors to optimize the generation of clean syngas for Solid Oxide Cells (SOCs) by harnessing advanced artificial intelligence (AI) and machine learning (ML) methodologies. Through the development and deployment of regression models, including SVM, KNN, and Multiple Linear Regression, the project addresses a pivotal challenge in transitioning to a hydrogen-based economy: This project catalyzes collaboration across engineering, chemistry, and AI domains, driving innovation, economic prosperity, and global sustainability efforts towards a hydrogen-powered future.

The impact of this initiative extends to domains such as industrial processes, energy production, and environmental monitoring, where accurate predictions of fuel outlet temperatures are critical for process efficiency and optimization. By leveraging innovative AI/ML techniques such as regression modeling and comparing it with the results available from Computational Fluid Dynamics (CFD) modeling, interdisciplinary collaboration is fostered, propelling progress in renewable energy research.

Moreover, the project's deployment strategies prioritize seamless integration with existing systems and user-friendly interface design, facilitating practical implementation in syngas production facilities and energy systems. This scalability enables widespread adoption, amplifying the project's impact on emissions reduction and the advancement of clean energy technologies.

For future work, several directions can be explored:

1. Enhanced Model Interpretability: Investigate methods for improving the interpretability of complex regression models like SVM and KNN to provide more actionable insights to stakeholders.

2. Ensemble Methods: Explore the use of ensemble learning techniques, such as model stacking or boosting, to further improve prediction accuracy and robustness.

3. Dynamic Model Updating: Implement mechanisms for dynamically updating the regression models with new data to ensure their continued relevance and performance over time.

4. Integration with Real-Time Systems: Integrate the deployed models with real-time monitoring and control systems to enable predictive maintenance and optimization in industrial settings.

References

a. Academic Papers:

Optimising pre-reforming for quality r-SOC syngas preparation using artificial intelligence (AI) based machine learning (ML) - ScienceDirect

b. AI tools: Chat-GPT.

c. Software Documentation: Documentation for AI/ML frameworks like matplotlib, NumPy and Scikit-learn.

Appendices

Any supplementary material, including code snippets, detailed data analysis, or additional plots and graphs.

Table1:

	Fuel [kg/s]	Air [kg/s]	Air [°C]	O_2 [%]	Catalyst	ML	CFD	%Error
Case 1	3.9392E-05	7.203E-04	675	10	33	575	577	0.34
Case 2	1.57569E-04	2.6508E-04	685	5	32	461	460	-0.54
Case 3	3.9392E-05	5.803E-04	695	10	50	556	553	0.21
Case 4	1.5757E-04	4.4015E-04	705	5	29	522	504	-3.5

Code 1:

```
import numpy as np
import pandas as pd
# Set random seed for reproducibility
np.random.seed(94)
# Define the number of data points
num_data_points = 1000
# Define the percentage of noise (4%)
noise_percentage = 0.04
# Define ranges for various parameters
fuel_flow_min, fuel_flow_max = 0.00002, 0.0009
air_flow_min, air_flow_max = 0.0002, 0.0009
air_temp_min, air_temp_max = 660, 740
porosity_min, porosity_max = 0.2, 0.5
o2_min, o2_max = 460, 580
fuel_temp_min, fuel_temp_max = 460, 580
# Generate synthetic data with noise
data = {
  'Fuel [kg/s]': np.linspace(fuel_flow_min, fuel_flow_max, num_data_points) * (1 + noise_percentage * np.random.randn(num_data_points)),
  'Air [kg/s]': np.linspace(air_flow_min, air_flow_max, num_data_points) * (1 + noise_percentage * np.random.randn(num_data_points)),
  'Air [C]': np.linspace(air_temp_min, air_temp_max, num_data_points) * (1 + noise_percentage * np.random.randn(num_data_points)),
  'Catalyst_Porosity': np.linspace(porosity_min, porosity_max, num_data_points) * (1 + noise_percentage * np.random.randn(num_data_points)),
```

'O2 [%]': np.where((np.linspace(o2_min, o2_max, num_data_points) >= 460) & (np.linspace(o2_min, o2_max, num_data_points) <= 520), 5 * (1 + noise_percentage * np.random.randn(num_data_points)),

np.where((np.linspace(o2_min, o2_max, num_data_points) >= 520) & (np.linspace(o2_min, o2_max, num_data_points) <= 580), 10 * (1 + noise_percentage * np.random.randn(num_data_points)), 5 * (1 + noise_percentage * np.random.randn(num_data_points)))),

'Fuel_outlet_temp': np.linspace(fuel_temp_min, fuel_temp_max, num_data_points) * (1 + noise_percentage * np.random.randn(num_data_points))

}

Create a Data Frame with column order rearranged

df = pd.DataFrame(data, columns= ['Fuel [kg/s]', 'Air [kg/s]', 'Air [C]', 'Catalyst_Porosity', 'O2 [%]', 'Fuel_outlet_temp'])

Display the first few rows of the Data Frame

print(df.head())

Save the DataFrame to a CSV file

df.to_csv('final_data.csv', index=False)

Auxiliaries

Data Source:

https://raw.githubusercontent.com/VidhishaAgarwal/cl653/main/final_data.csv

Python file:

https://colab.research.google.com/drive/1q5TSyjBusG7fYlBi6uLhK5NJl6BXzu2i?usp=sharing