**Reactor performance optimization using ML**

***Project report submitted to***

***Visvesvaraya National Institute of Technology, Nagpur in partial fulfilment of the requirements for the***

***award of the degree***

**Bachelor of Technology in**

**Chemical Engineering**

***By***

## Prashant Bhushan (BT21CME0G7)

****

Under the guidance of

**Dr. K. L. Wasewar**

**Department of Chemical Engineering Visvesvaraya National Institute of Technology Nagpur, 440010 (India)**

**2024-25**

© Visvesvaraya National Institute of Technology (VNIT) 2024

## Department of Chemical Engineering Visvesvaraya National Institute of Technology, Nagpur

Declaration

I Prashant Bhushan, hereby declare that this project work titled “Reactor performance optimization using ML/AI” is carried out by me in the Department of Chemical Engineering of Visvesvaraya National Institute of Technology, Nagpur. The work is original and has not been submitted earlier whole or in part for the award of any degree/diploma at this or any other Institution / University.

Date:10th May,2025

**Prashant Bhushan**

(BT21CME097)

**Department of Chemical Engineering Visvesvaraya National Institute of Technology, Nagpur**

# CERTIFICATE

This to certify that the project titled “Reactor performance optimization using ML”, submitted by Prashant Bhushan in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Chemical Engineering**, VNIT Nagpur. The work is comprehensive, complete and fit for final evaluation.

Dr K.L Wasewar

Professor, Department of Chemical Engineering,

VNIT, Nagpur

# Acknowledgments

I would like to express my deepest gratitude to **Dr. Kailas Wasewar**, whose invaluable guidance, expertise, and encouragement have been instrumental in the successful completion of this project. His profound knowledge and insightful suggestions have greatly enriched my understanding and shaped the direction of this work.

I am also immensely grateful to **Ms. Anjali**, Ph.D. scholar, for her constant support, mentorship, and constructive feedback throughout the course of this project. Her patience, dedication, and willingness to share her expertise have been a source of inspiration and motivation.

Additionally, I would like to thank my institution and department for providing the necessary resources and a conducive environment to carry out this research. Finally, I extend my heartfelt appreciation to my family and friends for their unwavering support and encouragement during this journey.

This project would not have been possible without the collective efforts and contributions of all those mentioned above. Thank you for making this endeavor a rewarding and fulfilling experience.

# Abstract

Reactor performance optimization is a critical aspect of chemical engineering, aimed at maximizing efficiency, yield, and sustainability in industrial processes. This study explores the application of Artificial Intelligence (AI) and Machine Learning (ML) techniques to optimize the performance of three reactor types: Batch Reactor, Continuous Stirred Tank Reactor (CSTR), and Plug Flow Reactor (PFR). The saponification reaction between ethyl acetate and sodium hydroxide was chosen as the model reaction due to its well-defined kinetics and industrial relevance.

The project begins with the generation of synthetic datasets for each reactor type, incorporating key operational parameters such as reactant concentrations, flow rates, reactor volume, temperature, and reaction time. Using kinetic equations and reactor-specific mole balances, conversion rates ((X\_A)) were calculated for each dataset. Data preprocessing steps, including cleaning, scaling, and outlier removal, ensured the datasets were suitable for machine learning applications.

A variety of machine learning models, including Random Forest, Gradient Boosting, XGBoost, Cat Boost, and Neural Networks, were trained to predict conversion rates based on the input parameters. Model performance was evaluated using metrics such as Mean Squared Error (MSE) and R-squared ((R^2)), with Cat Boost emerging as the best-performing model across all reactor types due to its high accuracy and ability to handle non-linear relationships.

Optimization of reactor performance was achieved using Bayesian Optimization, which identified the optimal operating conditions for maximizing conversion. For the Batch Reactor, the optimal conditions included specific values for initial reactant concentrations, temperature, and reaction time. Similarly, for the CSTR and PFR, optimal flow rates, concentrations, and reactor volumes were determined. The results demonstrated significant improvements in conversion rates, highlighting the effectiveness of AI/ML techniques in reactor optimization.

This study also included feature importance analysis to identify the most influential parameters affecting conversion, providing valuable insights for reactor design and operation. Residual analysis and cross-validation ensured the robustness and reliability of the models. The findings were validated against theoretical predictions and experimental data, where available.

In conclusion, this project showcases the potential of AI/ML in revolutionizing reactor performance optimization. By leveraging advanced machine learning models and optimization algorithms, this approach offers a data-driven, efficient, and scalable solution for enhancing chemical reactor performance, paving the way for smarter and more sustainable industrial processes.

## Index

[CERTIFICATE 3](#_bookmark0)

[Acknowledgments 4](#_bookmark1)

[Abstract 5](#_bookmark2)

1. [Introduction G](#_bookmark3)
   1. [Background 9](#_bookmark4)
   2. [Objective 9](#_bookmark5)
   3. [Scope of the Study 10](#_bookmark6)
   4. [Significance of the Study 10](#_bookmark7)
2. [Literature Review 10](#_bookmark8)
   1. [Introduction to Reactor Optimization 10](#_bookmark9)
   2. [Overview of Reactor Types 10](#_bookmark10)
   3. [Saponification Reaction as a Model System 11](#_bookmark11)
   4. [Role of AI/ML in Reactor Optimization 11](#_bookmark12)
   5. [Applications in Literature 11](#_bookmark13)
   6. [Optimization Techniques 12](#_bookmark14)
   7. [Challenges in Reactor Optimization 12](#_bookmark15)
   8. [Gaps in Existing Research 12](#_bookmark16)
3. [Methodology 12](#_bookmark17)
   1. [Data Generation 13](#_bookmark18)
      1. [Reaction Overview 13](#_bookmark19)
      2. [Batch reactor 13](#_bookmark20)
      3. [CSTR 14](#_bookmark21)

[**PFR**:(plug flow reactor) 15](#_bookmark22)

* 1. [Data Preprocessing 16](#_bookmark23)
     1. [Data Cleaning 16](#_bookmark24)
  2. [Exploratory Data Analysis (EDA) 17](#_bookmark25)
     1. [Visualization 17](#_bookmark26)
     2. [Insights 18](#_bookmark27)
  3. [Model Training and Evaluation 19](#_bookmark28)
     1. [Model Selection 19](#_bookmark29)
        1. [Random Forest Regressor 19](#_bookmark30)
        2. [Gradient Boosting Regressor 19](#_bookmark31)
        3. [XGBoost Regressor 20](#_bookmark32)
        4. [CatBoost Regressor 21](#_bookmark33)
        5. Neural Networks 21

[3.5 Data Splitting 22](#_TOC_250017)

* + 1. batch reaction (𝑀𝑆𝐸 𝑎𝑛𝑑 𝑅2 ) comparison for different Model: 23
    2. For CSTR: 23
    3. For PFR 24
  1. [Residual Analysis 24](#_TOC_250016)
     1. Purpose of Residual Analysis 24
     2. Residual analysis of data set generated for CSTR 25
     3. Residual analysis of Batch reactor data set: 25
  2. [Hyperparameter Tuning 26](#_TOC_250015)
  3. [Optimization 26](#_TOC_250014)
     1. Objective 26
     2. Optimization Techniques 26
  4. [Results 27](#_TOC_250013)
     1. For PFR 28
     2. FOR Batch reactor 28
  5. [Model Deployment 28](#_TOC_250012)

1. [Discussion 28](#_TOC_250011)
   1. [Batch Reactor 29](#_TOC_250010)
   2. [Continuous Stirred Tank Reactor (CSTR) 29](#_TOC_250009)
   3. [Plug Flow Reactor (PFR) 29](#_TOC_250008)
2. [Conclusion 30](#_TOC_250007)
   1. [Summary 30](#_TOC_250006)
   2. [Contributions 31](#_TOC_250005)
   3. [Limitations 31](#_TOC_250004)
   4. [Future Work 31](#_TOC_250003)
3. [References 31](#_TOC_250002)
4. [Appendix: A 33](#_TOC_250001)
   1. Appendix: B 3G
   2. [Appendix: C 43](#_TOC_250000)

**List of figures**

Fig. 1 Reactor 9

Fig. 2 Batch reactor data set view 14

Fig. 3 cstr Fig. 4 PFR 14

Fig. 5 cstr Data set view 15

Fig. 6 pair wise plot for Batch reactor 17

Fig. 7 correletion matrix for Batch 17

Fig. 8 pair wise plot of features of pf 17

Fig. 9 Correlation matrix of featurs of pfr 17

Fig. 10 Cstr data set features 17

Fig. 11cstr features pair wise plot 17

Fig. 12 Most important features of cstr 18

Fig. 13most important features of batch 18

Fig. 14 Random forest 19

Fig. 15Gradient boosting 20

Fig. 16XGboosst regressor 21

Fig. 17catboost regressor 21

Fig. 18Neural network 22

Fig. 19 R^2 C RMS error for Batch 23

Fig. 20 R^2 and RMS error for CSTR 23

Fig. 21error for pfr 24

Fig. 22Normal distribution curve for residual 24

Fig. 23residual vs predicted value 24

Fig. 24Normal distributin for Cstr 25

Fig. 25Normal distribution for residual of catboost 26

Fig. 26 Bayesian optimization 26

Fig. 27Output pallet 27

Fig. 28optimal condition for pfr 28

Fig. 29optimal condition for batch 28

# Introduction

#### Background

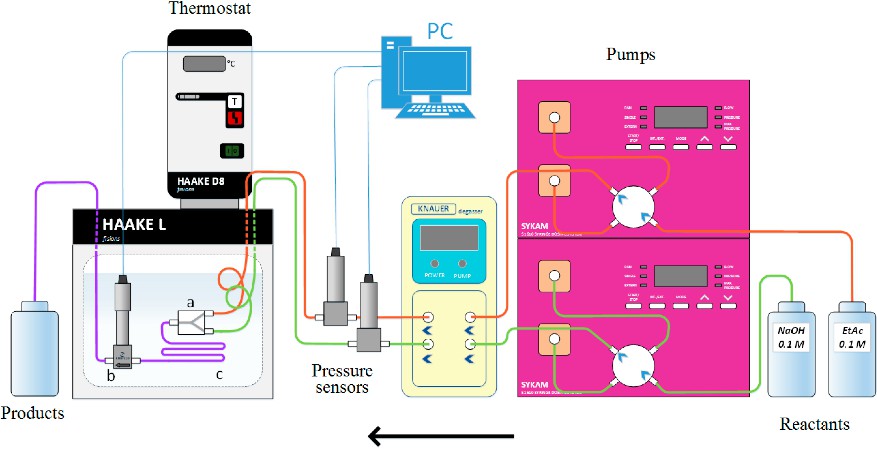
Chemical reactors are the backbone of the chemical and process industries, playing a pivotal role in the production of chemicals, pharmaceuticals, and other essential materials. The performance of these reactors directly impacts the efficiency, yield, and sustainability of industrial processes. Traditionally, reactor performance optimization has relied on experimental trial-and-error methods or mathematical modeling, which are often time-consuming, resource-intensive, and limited in their ability to handle complex, non- linear systems.

With the advent of **Artificial Intelligence (AI)** and **Machine Learning (ML)**, the landscape of reactor optimization has undergone a significant transformation. These technologies enable data-driven approaches to predict and optimize reactor performance by leveraging large datasets and advanced algorithms. AI/ML models can capture intricate relationships between input parameters (e.g., reactant concentrations, temperature, flow rates) and output variables (e.g., conversion, yield), offering a more efficient and scalable solution compared to traditional methods.

In this project, we focus on optimizing the performance of three types of reactors:

* + - * 1. **Batch Reactor**: A closed system where reactants are mixed and allowed to react over time.
        2. **Continuous Stirred Tank Reactor (CSTR)**: A reactor where reactants are continuously fed, and products are continuously removed, ensuring steady-state operation.
        3. **Plug Flow Reactor (PFR)**: A reactor where reactants flow through a tubular system, with conversion varying along the reactor length.

The saponification reaction between **ethyl acetate (CH₃COOC₂H₅)** and **sodium hydroxide (NaOH)** was chosen as the model reaction due to its well-defined kinetics and industrial relevance. The reaction produces **sodium acetate (CH₃COONa)** and **ethanol (C₂H₅OH)**, and its performance is influenced by factors such as reactant concentrations, temperature, and residence time.



*Fig. 1 Reactor*

Experimental study of saponification reaction using different reactor system

* + - 1. Objective

The primary objective of this project is to optimize reactor performance by:

* + - * 1. Predicting the conversion ((X\_A)) of the saponification reaction using AI/ML models.
        2. Identifying the optimal operating conditions (e.g., temperature, concentrations, flow rates) to maximize conversion.
        3. Comparing the performance of different machine learning models and optimization techniques.
      1. Scope of the Study

This study involves:

* **Data Generation**: Synthetic datasets were generated for each reactor type using kinetic equations and mole balances.
* **Data Preprocessing**: Cleaning, scaling, and feature engineering were performed to prepare the data for modelling.
* **Model Training**: Various machine learning models, including Random Forest, Gradient Boosting, CatBoost, and Neural Networks, were trained to predict conversion.
* **Optimization**: Bayesian Optimization was employed to determine the optimal reactor conditions.
* **Validation**: The models were validated using statistical metrics (e.g., Mean Squared Error, (R^2 )) and residual analysis.
  + - 1. Significance of the Study

This project demonstrates the potential of AI/ML in revolutionizing reactor performance optimization. By leveraging data-driven approaches, it provides:

* **Efficiency**: Faster and more accurate predictions compared to traditional methods.
* **Scalability**: Applicability to a wide range of reactions and reactor types.
* **Sustainability**: Improved resource utilization and reduced experimental costs.

1. Literature Review
   1. Introduction to Reactor Optimization

Reactor optimization is a cornerstone of chemical engineering, aiming to maximize efficiency, yield, and sustainability in industrial processes. Traditional optimization methods, such as experimental trial-and-error and mathematical modelling, are often resource-intensive and limited in their ability to handle complex, non-linear systems. The advent of **Artificial Intelligence (AI)** and **Machine Learning (ML)** has revolutionized this field, enabling data-driven approaches to predict and optimize reactor performance.

* 1. Overview of Reactor Types

This study focuses on three reactor types: **Batch Reactor**, **Continuous Stirred Tank Reactor (CSTR)**, and **Plug Flow Reactor (PFR)**. Each reactor type has unique characteristics and operational challenges:

###### Batch Reactor:

* + - * Operates in a closed system where reactants are mixed and allowed to react over time.
      * Commonly used for small-scale production and reactions requiring precise control.
      * Optimization involves parameters like reaction time, temperature, and initial concentrations.

###### CSTR:

* + - * Operates at steady state with continuous input and output of reactants and products.
      * Mixing ensures uniform concentration and temperature throughout the reactor.
      * Key optimization parameters include flow rates, reactor volume, and temperature.
    - **PFR**:
      * Reactants flow through a tubular reactor, with conversion varying along the reactor length.
      * Suitable for large-scale production and reactions with high conversion rates.
      * Optimization focuses on flow rates, residence time, and temperature.
  1. Saponification Reaction as a Model System

The saponification reaction between **ethyl acetate (CH₃COOC₂H₅)** and **sodium hydroxide (NaOH)** is widely studied due to its well-defined kinetics and industrial relevance. The reaction produces **sodium acetate (CH₃COONa)** and **ethanol (C₂H₅OH)**. The rate equation for this reaction is −𝑟𝐴 = 𝑘𝐶𝐴𝐶𝐵 where:

* + - (C\_A): Concentration of ethyl acetate.
    - (C\_B): Concentration of NaOH.

−𝐸𝑎

* + - (k): Reaction rate constant, calculated using the Arrhenius equation:𝑘 = 𝐾0𝑒 𝑅𝑇
    - ( E\_a ): Activation energy.
    - (R): Universal gas constant.
  1. Role of AI/ML in Reactor Optimization

AI/ML techniques have emerged as powerful tools for reactor optimization, offering several advantages over traditional methods:

###### Data-Driven Insights:

* + - * ML models can capture complex, non-linear relationships between input parameters (e.g., concentrations, temperature, flow rates) and output variables (e.g., conversion, yield).

###### Efficiency:

* + - * Reduces the need for extensive experimental trials.
      * Enables real-time predictions and optimization.

###### Scalability:

* + - * Applicable to a wide range of reactions and reactor types.
  1. Applications in Literature:

###### Batch Reactor Optimization:

* + Studies have demonstrated the use of Random Forest and Gradient Boosting models to predict conversion rates in batch reactors.
  + Feature importance analysis highlights the significance of reaction time and temperature in determining conversion.

###### CSTR Optimization:

* + Bayesian Optimization has been used to identify optimal flow rates and reactor volumes for maximizing conversion in CSTRs.

###### PFR Optimization:

* + Neural Networks and XGBoost models have been applied to predict conversion in PFRs, with residence time and temperature identified as key parameters.
  1. Optimization Techniques

Several optimization techniques have been explored in the literature:

###### Bayesian Optimization:

* + Efficient for non-linear, non-differentiable problems.
  + Balances exploration and exploitation to find global optima.
  + Widely used for optimizing reactor conditions.

###### Genetic Algorithms:

* + Mimics natural selection to explore the parameter space.
  + Suitable for high-dimensional problems but computationally expensive.

###### Gradient-Based Methods:

* + Effective for smooth, differentiable objective functions.
  + Limited applicability for non-linear systems like chemical reactors.
  1. Challenges in Reactor Optimization

Despite the advancements in AI/ML, several challenges remain:

###### Data Quality:

* + - * Accurate and representative datasets are critical for training ML models.
      * Synthetic datasets, while useful, may not fully capture real-world variability.

###### Model Interpretability:

* + - * Complex ML models like Neural Networks are often considered "black boxes."
      * Feature importance analysis and residual plots can help improve interpretability.

###### Computational Cost:

* + - * Training and optimizing ML models can be computationally intensive, especially for large datasets.
  1. Gaps in Existing Research
     + Limited studies on the integration of real-time data with ML models for dynamic optimization.
     + Lack of comprehensive comparisons between different ML models and optimization techniques.
     + Need for experimental validation of ML-based predictions.

# Methodology

This section outlines the systematic approach adopted to optimize reactor performance using AI/ML techniques. The methodology is divided into several key steps, covering data generation, preprocessing, feature engineering, model training, evaluation, and optimization. The study focuses on three reactor types: **Batch Reactor**, **Continuous Stirred Tank Reactor (CSTR)**, and **Plug Flow Reactor (PFR)**.

#### Data Generation

* + 1. *Reaction Overview*

The saponification reaction between **ethyl acetate (CH₃COOC₂H₅)** and **sodium hydroxide (NaOH)** was chosen as the model reaction. The reaction produces **sodium acetate (CH₃COONa)** and **ethanol (C₂H₅OH)**

###### Reaction equation: -

CH₃COOC₂H₅ (𝑐𝑎) + NaOH (𝑐𝑏) CH₃COONa + C₂H₅OH

Rate equation: −𝑅𝐴 = 𝑘𝐶𝐴𝐶𝐵 (1)

Arrhenius equation 𝐾 = 𝑘0𝑒(−𝐸𝐴∕𝑅𝑇) (2)

* + 1. *Batch reactor:* **-**

###### Assumptions for Not Including Stirrer Speed in Batch Reactor Analysis

1. Homogeneous Reaction: The saponification reaction is a liquid-phase reaction controlled by chemical kinetics, not mass transfer.
2. Complete Mixing: Stirrer speed is assumed to be high enough to ensure uniform mixing, eliminating concentration gradients.
3. Reaction-Controlled Regime: The reaction rate depends on intrinsic kinetics, not external factors like mixing.
4. Negligible Impact: Beyond a certain threshold, increasing stirrer speed does not affect conversion or reaction rate.
5. Simplification: Excluding stirrer speed avoids unnecessary complexity without compromising model accuracy.

###### Features for batch reactor: -

* 1. Initial concentration (Ethyl acetate): Ca0(0.01 M – 0.1 M)
  2. Initial concentration (NaoH): Cb0(0.01 M – 0.1 M)
  3. Temperature (T ,0c)
  4. Reaction time (t, min)

Process of data set generation: using python random no function

###### Kinetic equations for batch data set generation: -

𝑀 = 𝐶𝑏0

𝐶𝐴0

−𝑟𝑎 = 𝑘𝑐𝑎𝑐𝑏

𝑐𝑎0 𝑑𝑥𝑎 = 𝑘 𝑐𝑎02(1 − 𝑥𝑎)(𝑀 − 𝑥𝑎)

𝑑𝑡

𝑑𝑥𝑎 = 𝑘 𝑐𝑎0(1 − 𝑥𝑎)(𝑀 − 𝑥𝑎)

𝑑𝑡

𝑥𝐴

𝑑𝑥 𝑡

∫ 𝐴 = ∫ 𝑘𝐶𝐴

𝑑𝑡

0 (1−𝑥𝐴)(𝑀−𝑥𝐴) 0 0

𝑙𝑛 | 𝑀−1

𝑀(1−𝑥𝐴)

| = (𝑀 − 1)𝐾𝐶𝐴𝑜𝑡

𝑥𝐴

= 𝑀[𝑒^(𝑀−1)𝐾𝐶𝐴0𝑡−1]

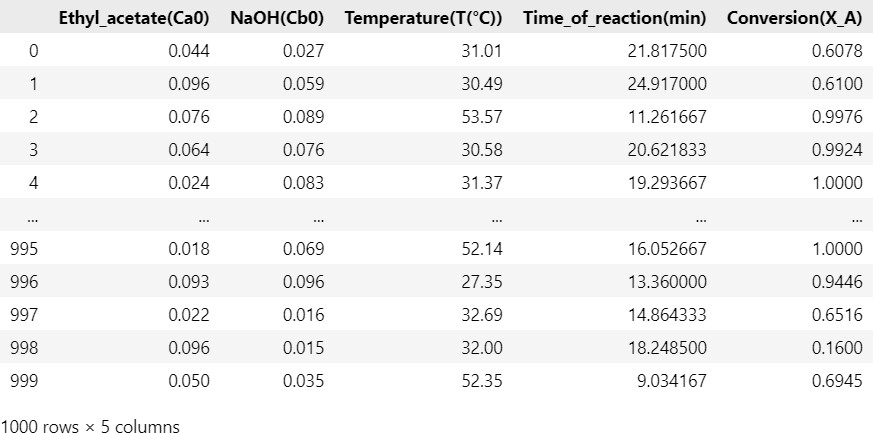
𝑀(𝑀−1)𝐾𝐶𝐴0 −1

𝑒

(3)

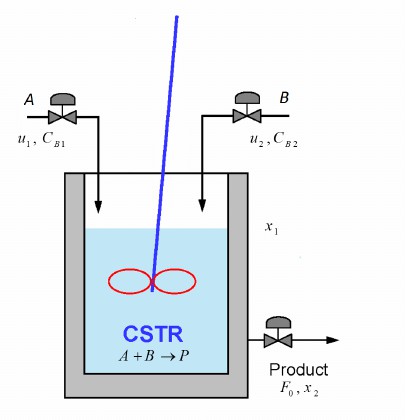
Const = 𝑒^(𝑀 − 1)𝐾𝐶𝐴0𝑡 (4)

###### Data set view: -

****

*Fig. 2 Batch reactor data set view*

* + 1. ***CSTR* PFR**

****

*Fig. 3 cstr Fig. 4 PFR*

###### Features:

* + - Flow rates of ethyl acetate and NaOH. (0.1 – 10 L/S)
    - Reactor volume. (v: 1 – 1000 L/s)
    - Temperature. (20 – 60 0c)

###### Why Stirrer Speed is Not Included in the Dataset

1. **Homogeneous Reaction**: The saponification reaction is a liquid-phase reaction where the reaction rate is controlled by chemical kinetics, not mass transfer.
2. **Complete Mixing Assumption**: It is assumed that the stirrer speed is high enough to ensure uniform mixing, eliminating concentration gradients.
3. **Reaction-Controlled Regime**: The reaction operates in a regime where the rate is determined by intrinsic kinetics, making stirrer speed irrelevant.
4. **Negligible Impact**: Beyond a certain threshold, increasing stirrer speed does not affect conversion or reaction rate.
5. **Simplification**: Excluding stirrer speed avoids unnecessary complexity and noise in the dataset, focusing on key parameters like concentrations, temperature, and residence time.

Kinetic equations used for data set generation

Space time(**τ) =**  𝑪𝑨𝒐𝒙𝑨

−𝒓𝑨

−𝒓𝑨 = 𝒌𝑪𝑨𝑪𝑩

𝑀 = 𝐶𝑏0

𝐶𝐴0

**-** **(5)**

**τ =**  𝑪𝑨𝟎𝒙𝑨 **(6)**

𝒌𝑪𝟐 (𝟏−𝒙𝑨)(𝑴−𝒙𝑨)

𝑨𝒐

1

2 2

𝑥 − ( + 𝑀 + 1) 𝑋 + 𝑀 = 0

𝐴 𝑘𝑐𝐴𝑜𝑇

𝐴

1 ̅̅̅̅̅̅̅̅̅̅̅̅̅̅̅1̅̅̅̅̅̅2̅̅̅̅̅̅̅̅̅

𝑥𝐴 = [ (1 + 𝑀 +

𝑘𝐶

𝐴𝑜

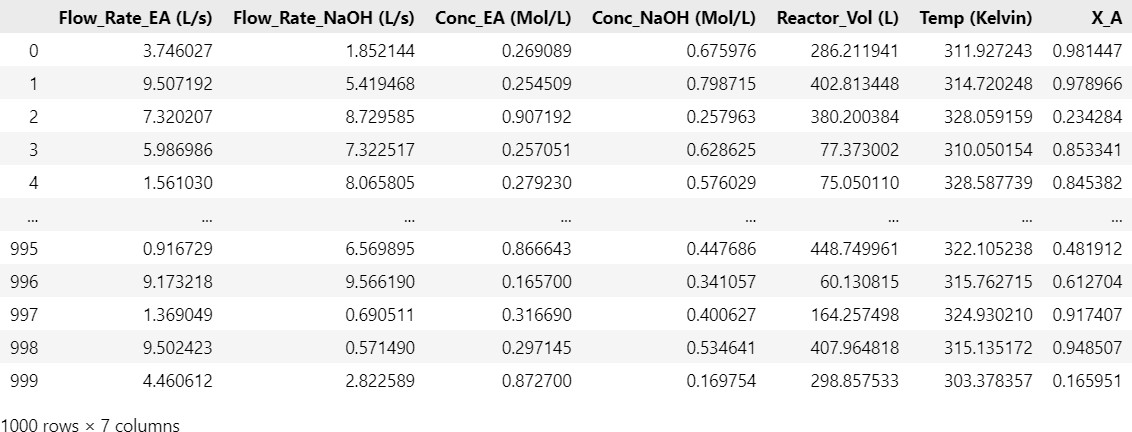
) − √(1 + 𝑀 + )

𝜏 𝑘𝐶𝐴𝑜𝜏

− 4𝑀

] / 2 (7)

###### Data set overview (CSTR reactor): -

****

*Fig. 5 cstr Data set view*

***PFR****:(plug ffow reactor)*

Features for PFR:

* Flow rates of ethyl acetate and NaOH. (0.1 – 10 L/S)
* Reactor volume. (0.1 – 1000 L)
* Temperature. (20 – 60 0c)
* Concentration: Ca0, Cb0(0.01 – 0.1 M) Kinetic equations for data set generation for PFR:

𝜏 = 𝑣 (𝑣01+𝑣02)

= 𝑣

𝑣0

(8)

𝑥𝐴 𝑑𝑥

𝑣

𝑑𝑣

∫ 𝐴 = ∫ (9)

0 −𝑟𝐴

0 𝐹𝐴𝑜

−𝑟𝐴 = 𝑘𝐶𝐴𝐶𝐵

𝐹𝐴0 = 𝑣0𝐶𝐴𝑜 (10)

𝑥𝐴

∫

0

𝑑𝑥𝐴 = 𝑘𝐶 𝜏

(1−𝑥𝐴)(𝑀−𝑥𝐴)

𝐴0

𝑥𝐴

= 𝑀[𝑒^(𝑀−1)𝐾𝐶𝐴0𝜏−1]

𝑀(𝑀−1)𝐾𝐶𝐴0 −1

𝑒

(11)

###### PFR Data set overview:

*Figure c pfr data set*

#### Data Preprocessing

* + 1. *Data Cleaning*
       - **Handling Missing Values**: Rows with missing values were removed to ensure data completeness.
       - **Outlier Removal**: Outliers were identified and removed using the Z-score method, retaining only rows where all feature values were within 3 standard deviations.

Z-score = (x−μ) / σ (11)

###### Scaling and Normalization

* + - * Features were scaled using **StandardScaler** to ensure all variables contributed equally to the model. This step was crucial for distance-based and gradient-based machine learning algorithms.

**Scaling** (Standardization):

**Goal**: Make features have **zero mean** and **unit variance**

X -scaled = (x−μ) /σ (12)

1. **Normalization** (Min-Max Scaling):

**Goal**: Rescale features to a **fixed range**, usually [0, 1].

(13)

𝑥𝑁

= (𝑥−𝑥𝑚) (𝑥𝑀−𝑥𝑚)

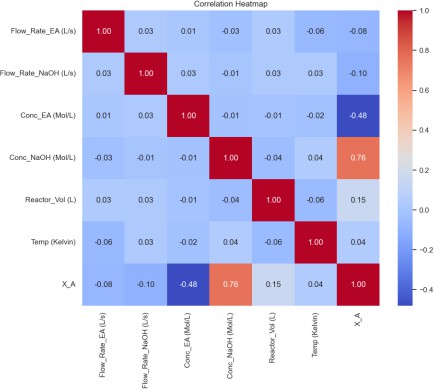
###### Statistical Summary

* + A statistical summary of the dataset was generated to understand the distribution of features and identify potential data quality issues.

#### Exploratory Data Analysis (EDA)

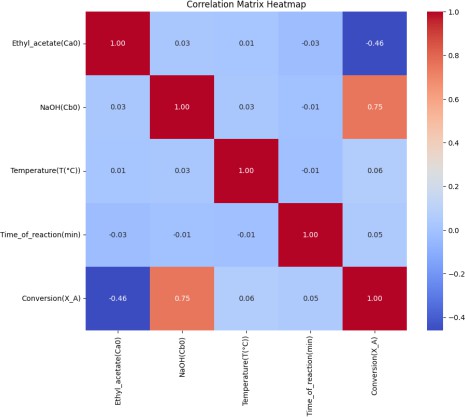
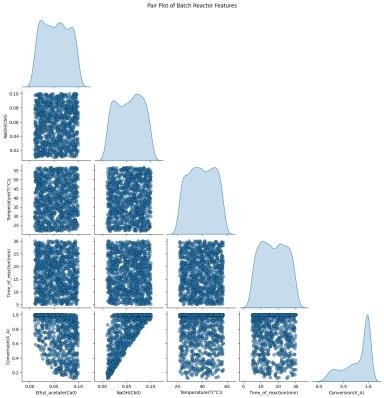
* + 1. *Visualization*
       - **Scatter Plots**: Analysed relationships between temperature, flow rates, and conversion.
       - **Heatmaps**: Identified correlations between features.
       - **Pair Plots**: Explored interactions between variables.

**Correlation Matrix and pair plot**

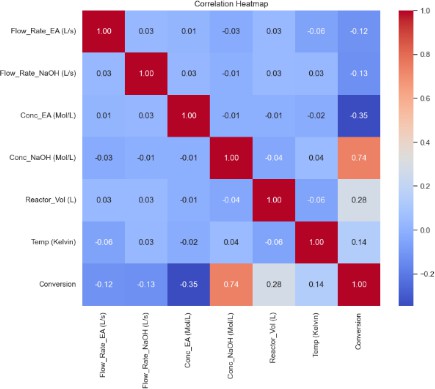
 

*Fig. c pair wise plot for Batch reactor*

*Fig. 7 correletion matrix for Batch*

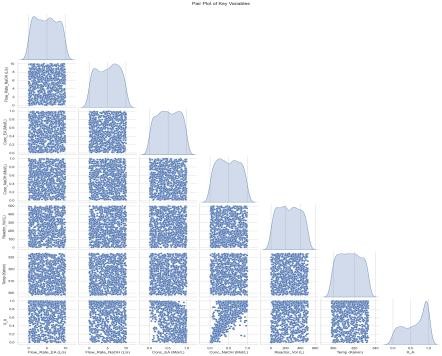
 

*Fig. S Correlation matrix of featurs of pfr*

**

*Fig. 10 Cstr data set features*

*Fig. 8 pair wise plot of features of pf*

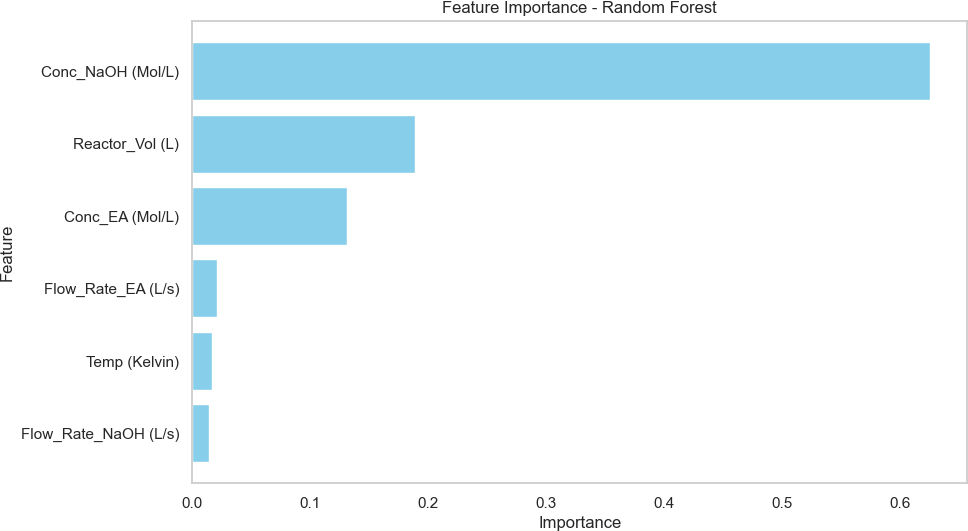
**

*Fig. 11cstr features pair wise plot*

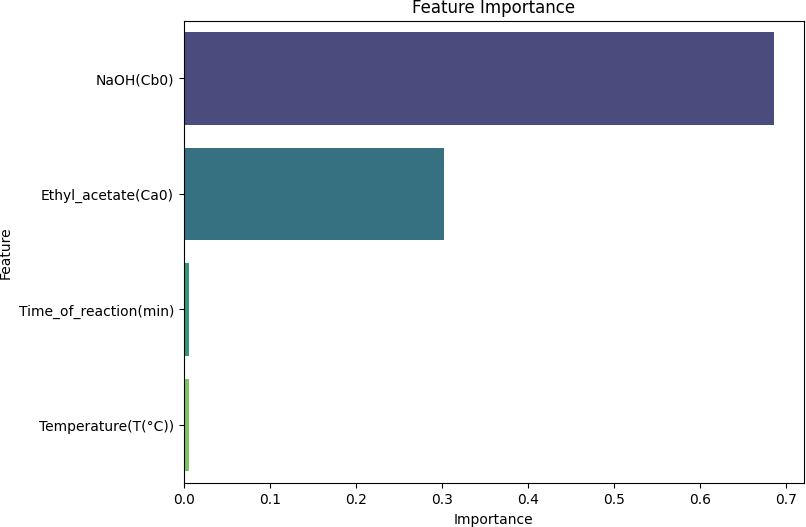
* + 1. Insights

Key variables influencing conversion were identified, Concentrations of reactant

* + - * Reactor volume.
      * Temperature.
      * Flow rates.



*Fig. 12 Most important features of cstr*

**

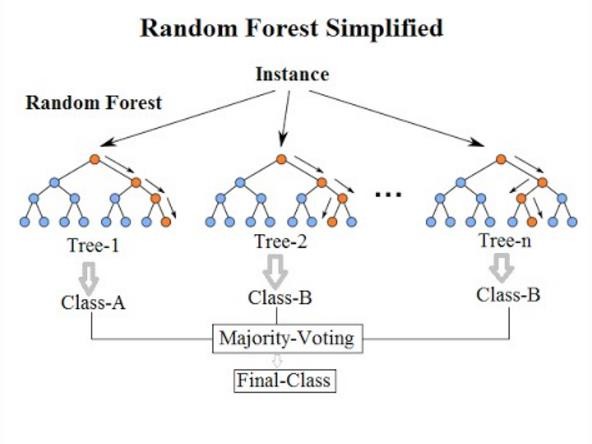
*Fig. 13most important features of batch*

Batch reactor

#### Model Training and Evaluation

* + 1. *Model Selection*

The following machine learning models were trained to predict conversion ((X\_A)):

* + - 1. *****Random Forest Regressor*.

*Fig. 14 Random forest*

* + - * + Random Forest is an **ensemble learning method** that combines multiple decision trees to improve prediction accuracy and reduce overfitting.
        + It works by averaging the predictions of individual decision trees, which are trained on different subsets of the data (using bootstrapping) and features.

###### How Does It Work?

1. **Bootstrap Sampling**:
   * Random subsets of the training data are created with replacement.

###### Feature Randomness:

* + At each split in a tree, a random subset of features is considered to reduce correlation between trees.

###### Aggregation:

* + For regression, the final prediction is the **average** of predictions from all trees.

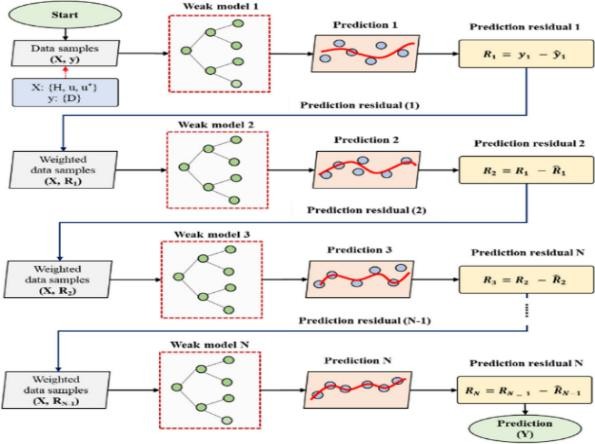
###### Advantages

1. **Handles Non-Linearity**: Can model complex, non-linear relationships.
2. **Robust to Overfitting**: Reduces overfitting by averaging multiple trees.
3. **Feature Importance**: Provides insights into which features are most important for predictions.
4. **Handles Missing Data**: Can handle missing values effectively.
5. **Scalability**: Works well with large datasets.

###### Disadvantages

1. **Computationally Intensive**: Training can be slow for large datasets with many trees.
2. **Interpretability**: Less interpretable compared to single decision trees.
3. **Overfitting Risk**: May overfit if the number of trees is too small or if hyperparameters are not tuned.

##### Gradient Boosting Regressor.



*Fig. 15Gradient boosting*

Gradient Boosting Regressor is an ensemble learning technique used for regression tasks. It builds models sequentially, where each new model corrects the errors of the previous one by minimizing a loss function, such as Mean Squared Error, using gradient descent. The process starts with an initial prediction, typically the mean of the target variable, and calculates residuals (the difference between actual and predicted values). New models, usually decision trees, are then trained to predict these residuals. The final prediction is a weighted sum of all the models' outputs, making Gradient Boosting highly effective for capturing complex, non-linear relationships in data.

This method is highly accurate and customizable, allowing for the tuning of hyperparameters like the number of boosting stages (**n\_estimators**), learning rate (**learning rate**), and tree depth (**max\_depth**). It also provides insights into feature importance, helping identify the most influential variables in the dataset. However, Gradient Boosting can be computationally expensive and prone to overfitting if not carefully tuned. It is less interpretable compared to simpler models, but its performance often outweighs this drawback.

Gradient Boosting Regressor is widely used in applications such as finance (e.g., risk assessment, stock price prediction), healthcare (e.g., disease progression modelling), e-commerce (e.g., sales forecasting, customer segmentation), and engineering (e.g., process optimization, energy consumption prediction). It is particularly effective for structured/tabular data where high accuracy is required. Despite its computational cost, Gradient Boosting is a powerful tool for predictive modelling in scenarios with complex patterns and relationships

* + - 1. *XGBoost Regressor*

XGBoost (Extreme Gradient Boosting) is an advanced machine learning algorithm based on Gradient Boosting. It is designed for high performance, scalability, and accuracy, making it one of the most popular algorithms in predictive modelling. XGBoost builds decision trees sequentially, where each tree corrects the errors of the previous ones. It introduces optimizations like regularization, parallel processing, and efficient memory usage, which make it faster and more robust than traditional Gradient Boosting methods. XGBoost is particularly effective for handling large datasets and complex relationships.

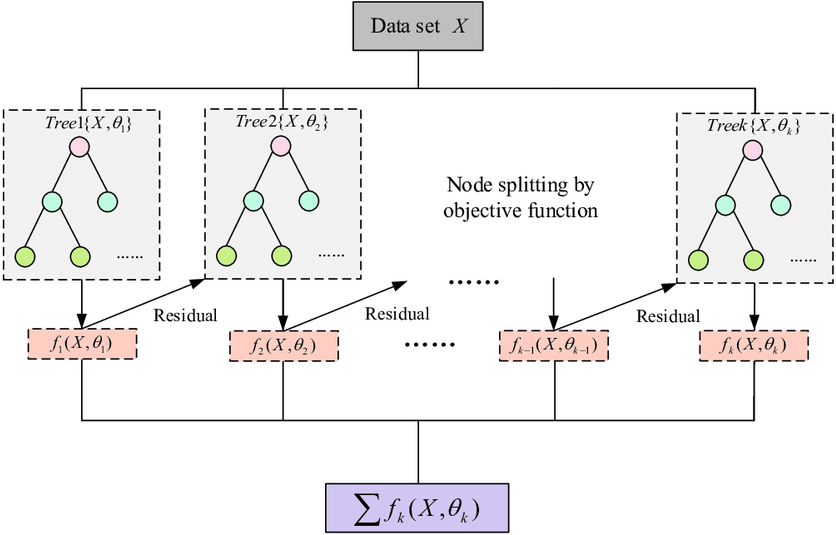
One of its key strengths is its ability to prevent overfitting through L1 (Lasso) and L2 (Ridge) regularization. It also supports early stopping, which halts training when performance stops improving, saving computational resources. Additionally, XGBoost can handle missing values automatically and provides feature importance scores, helping users understand which variables contribute most to predictions.

###### Advantages

* + - * + High accuracy and efficiency.
        + Handles missing data and outliers effectively.
        + Scalable for large datasets.
        + Provides feature importance for interpretability.

###### Disadvantages

* + - * + Computationally intensive for very large datasets.
        + Requires careful hyperparameter tuning.
        + Less interpretable compared to simpler models.



*Fig. 1cXGboosst regressor*

* + - 1. *CatBoost Regressor*

CatBoost (Categorical Boosting) is a high-performance machine learning algorithm based on Gradient Boosting. It is specifically designed to handle categorical features efficiently without requiring extensive preprocessing, such as one-hot encoding. CatBoost builds decision trees sequentially, where each tree corrects the errors of the previous ones, and it uses a novel technique called **ordered boosting** to reduce overfitting. This makes it particularly robust and accurate for both small and large datasets.

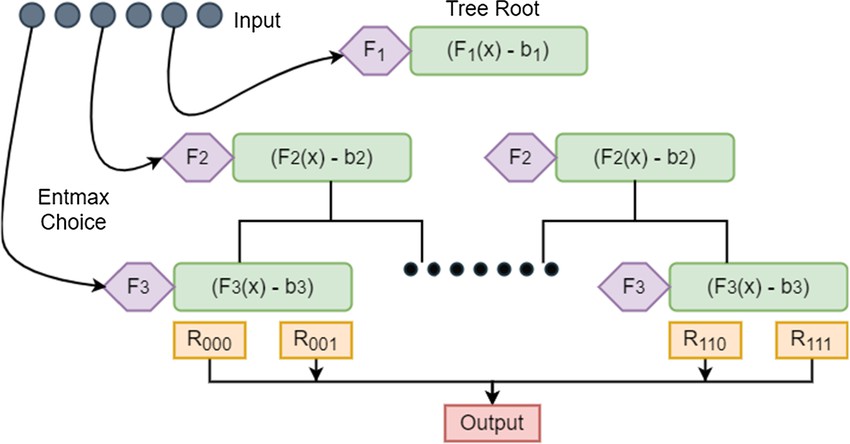
One of the standout features of CatBoost is its ability to handle categorical data natively, which simplifies the preprocessing pipeline and reduces the risk of information loss. It also supports GPU acceleration, making it faster for large datasets. CatBoost is known for its ease of use, requiring minimal hyperparameter tuning to achieve competitive results. Additionally, it provides feature importance scores, helping users understand the impact of each variable on the predictions.

###### Advantages

* + - * + Handles categorical features natively.
        + Reduces overfitting with ordered boosting.
        + Requires minimal preprocessing and hyperparameter tuning.
        + Supports GPU acceleration for faster training.

###### Disadvantages

* + - * + Computationally intensive for very large datasets.
        + Less interpretable compared to simpler models



*Fig. 17catboost regressor*

* + - 1. *Neural Networks*

###### Working of a Neural Network

A neural network is a machine learning model inspired by the structure of the human brain. It consists of layers of interconnected nodes (neurons) that process data and learn patterns.

1. **Input Layer**: Receives raw data features (e.g., numbers, images).
2. **Hidden Layers**: Perform computations using weights, biases, and activation functions to extract patterns and relationships.
   * **Weights**: Determine the importance of inputs.
   * **Activation Functions**: Introduce non-linearity (e.g., ReLU, Sigmoid).
3. **Output Layer**: Produces the final prediction (e.g., classification or regression).

###### How It Works

1. **Forward Propagation**:
   * Data flows through the network from the input layer to the output layer.
   * Each neuron computes a weighted sum of inputs, applies an activation function, and passes the result to the next layer.

###### Loss Calculation:

* + The difference between the predicted output and the actual target is calculated using a loss function (e.g., Mean Squared Error, Cross-Entropy).

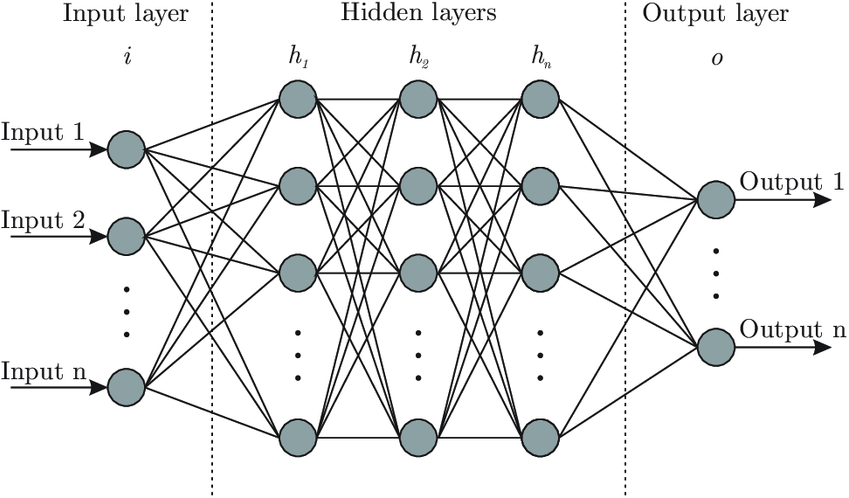
###### Backward Propagation:

* + The network adjusts weights and biases using the gradient of the loss function (calculated via backpropagation) to minimize errors.
  + Optimization algorithms like **Stochastic Gradient Descent (SGD)** or **Adam** are used.

###### Training:

* + The process of forward propagation, loss calculation, and backward propagation is repeated over multiple iterations (epochs) until the model converges.

Neural networks are highly effective for tasks like image recognition, natural language processing, and regression, especially when dealing with large and complex datasets.



*Fig. 18Neural network*

#### Data Splitting

* The dataset was split into training (80%) and testing (20%) sets.
* K-fold cross-validation (k=5) was performed to ensure robust evaluation.
  1. **Model Evaluation**

Models were evaluated using:

###### Mean Squared Error (MSE).

**MSE =** 𝜮(𝒀𝒊−𝒚̂𝒊)𝟐 **(14)**

𝒏′

###### R-squared ((R^2)).

∑𝑛 (𝑦 −𝑦̂ )2

Coefficient of Determination (𝑅2) = 𝑗=1 𝑖 𝑖 (15)

𝑛

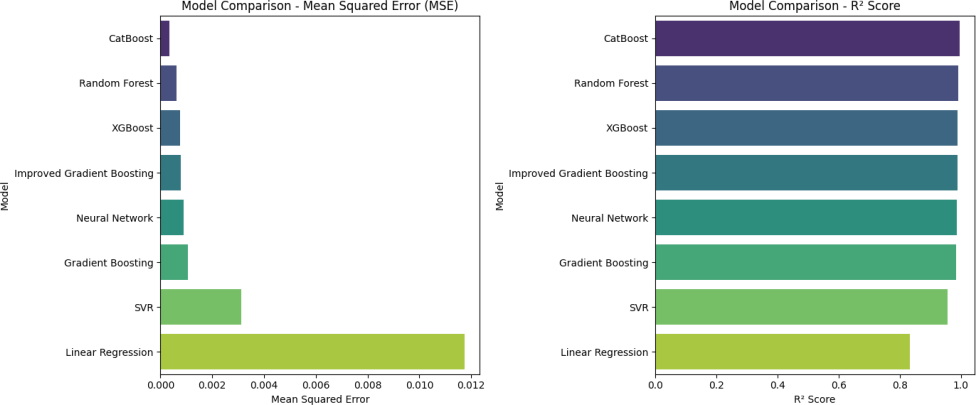
2

∑ (𝑦𝜍2−𝑦̅𝑖)−

𝐽=1

* Feature importance analysis was conducted to identify the most influential parameters.

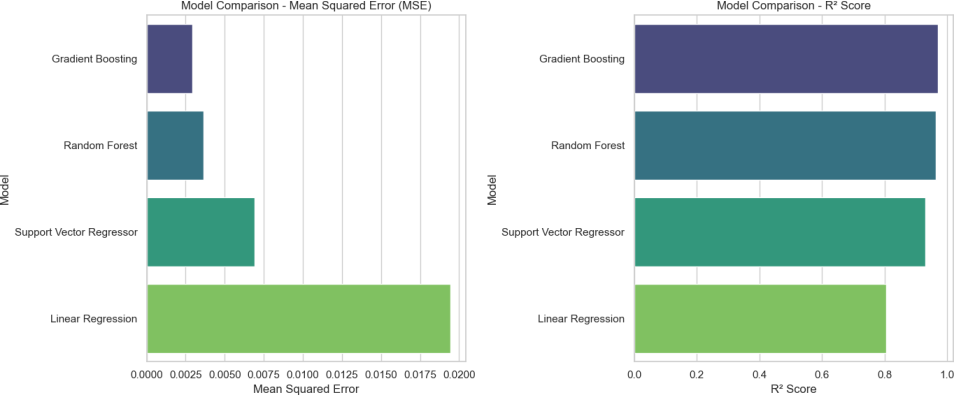
***3.C.1 batch reaction*** *(*𝑀𝑆𝐸 𝑎𝑛𝑑 𝑅2 *) comparison for different Model: -*

**

*Fig. 1S R^2 & RMS error for Batch*

**On basis of** 𝑀𝑆𝐸 𝑎𝑛𝑑 𝑅2 best model for Batch reactor **catboost**

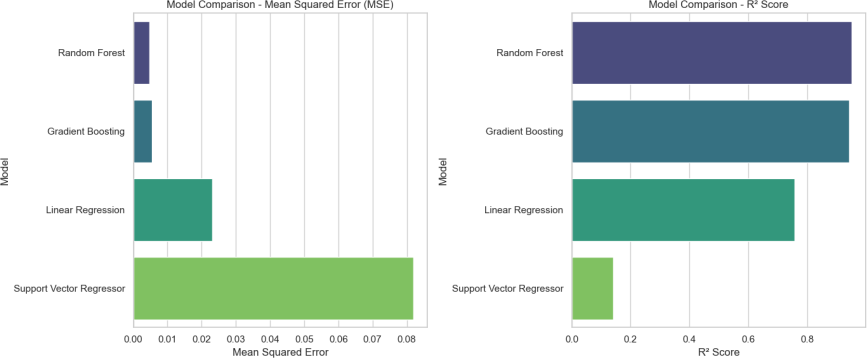
*3.c.2 For CSTR:*

**

*Fig. 20 R^2 and RMS error for CSTR*

**Gradient Boosting** is the best model working best **on CSTR** data set.

*3.c.3 For PFR*

**

*Fig. 21error for pfr*

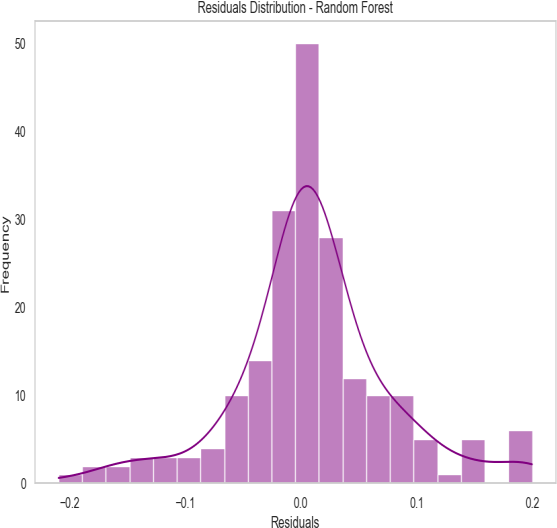
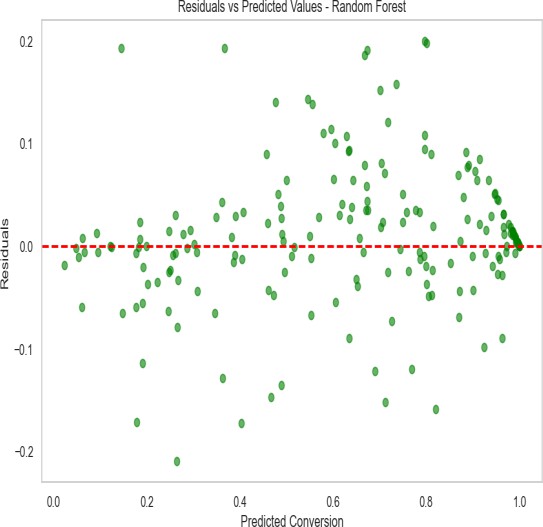
For PFR **Random Forest** is best Model …

#### Residual Analysis

* Residual plots and histograms were used to assess model performance and identify potential biases. Residual (𝑒) = 𝑌𝑖 − 𝑦̂𝑖 (16)
  + 1. *Purpose of Residual Analysis*

Residual analysis evaluates the difference between actual and predicted values (residuals) to check the model's performance. It helps identify:

* + - 1. **Bias**: Whether the model systematically over- or under-predicts.
      2. **Error Distribution**: Whether residuals are normally distributed.
      3. **Model Fit**: Whether the model captures the data patterns effectively.



*Fig. 22Normal distribution curve for residual*

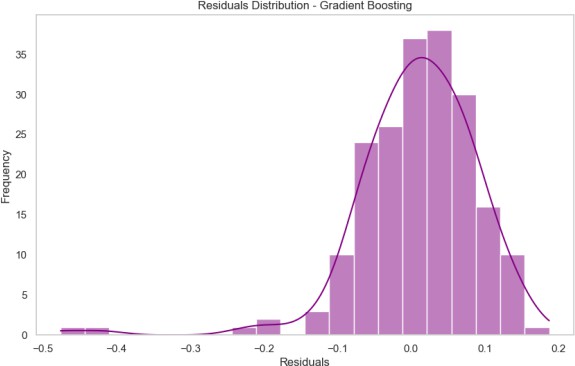
*Fig. 23residual vs predicted value*

###### Interpretation of the Two Residual Analysis Curves for PFR: -

1. **Residuals vs Predicted Values (Scatter Plot)**:
   * **Observation**: Residuals are scattered around the red dashed line (y=0), but some patterns are visible.
   * **Conclusion**: The model performs reasonably well, but slight patterns suggest it may not fully capture all relationships.

###### Residuals Distribution (Histogram):

* + **Observation**: Residuals are approximately normally distributed, cantered around 0, with a few outliers.
  + **Conclusion**: The model's errors are unbiased, but the presence of outliers indicates areas for improvement.
    1. *Residual analysis of data set generated for CSTR*

**

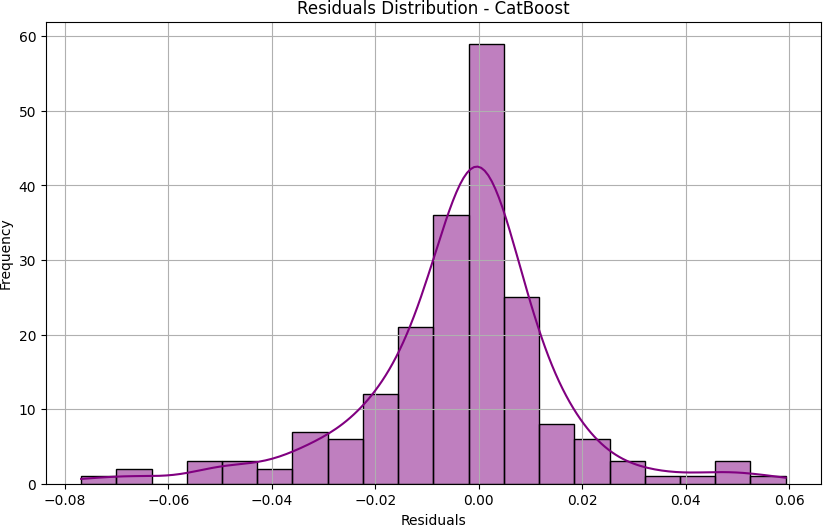
*Fig. 24Normal distributin for Cstr*

###### What the Graph Denotes:

* + - 1. Centering Around Zero:
         * The residuals are mostly centered around 0, indicating that the model's predictions are unbiased on average.
         * This is a positive sign, as it shows the model does not systematically overpredict or underpredict.
      2. Normal Distribution:
         * The residuals approximately follow a normal distribution (bell-shaped curve), which is desirable for a well-performing regression model.
         * This suggests that the model's errors are random and not influenced by specific patterns in the data.
      3. Spread of Residuals:
         * Most residuals are close to 0, indicating that the model's predictions are accurate for the majority of data points.
         * However, there are a few outliers (residuals far from 0), which may indicate cases where the model struggles to predict accurately.

Conclusion:

* The model performs well overall, with unbiased and normally distributed residuals.
* The presence of a few outliers suggests that the model could be further improved by:
  + Investigating the outliers for potential data issues.
  + Refining the model with additional features or hyperparameter tuning.
    1. *Residual analysis of Batch reactor data set: -*



*Fig. 25Normal distribution for residual of catboost*

#### Hyperparameter Tuning

* **Grid Search** was used to optimize hyperparameters for Random Forest, Gradient Boosting, and XGBoost models.
* The best-performing model was selected based on ( R^2) and MSE.

#### 3.9. Optimization

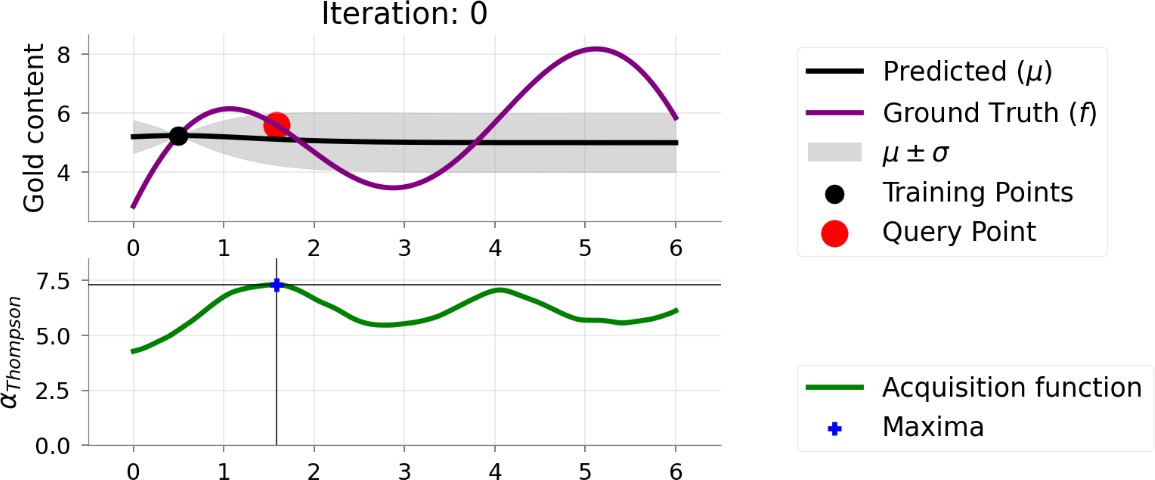
* + 1. *Objective*

To identify the optimal operating conditions (e.g., temperature, concentrations, flow rates) that maximize conversion ((X\_A)).

* + 1. *Optimization Techniques*

###### Bayesian Optimization:

* + - * + Defined the objective function as the predicted conversion ((X\_A)).
        + Explored the parameter space to find the global optimum.



*Fig. 2c Bayesian optimization*

* + - * **Genetic Algorithm** (optional): Used for comparison in some cases.

Comparison of Different Optimization Algorithms: -

|  |  |  |
| --- | --- | --- |
| **Method** | **Advantages** | **Disadvantages** |
| **L-BFGS-B** | Fast for smooth, differentiable functions. | Struggles with non-differentiable, non-linear models like Random Forest. |
| **Bayesian Optimization** | Efficient for non-linear, non- differentiable models. Finds global optima. | Computationally expensive for very high-dimensional problems. |
| **Genetic Algorithm** | Good for non-linear, non- differentiable problems. | Requires many function evaluations, slower convergence. |
| **Grid Search** | Exhaustive search, guarantees finding the best solution in the grid. | Computationally expensive, especially for high-dimensional problems. |
| **Random Search** | Simple and easy to implement. | Inefficient, may miss the global optimum. |

* + - * implementation to optimize your reactor performance.

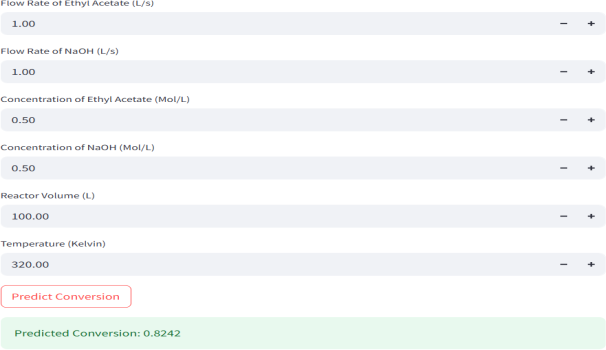
###### Conclusion

* + - * **Bayesian Optimization** is the best choice for your CSTR model because it handles non-linear, non- differentiable models effectively and efficiently explores the parameter space to find the global optimum.
      * Use the provided Bayesian Optimization implementation to optimize your reactor performance

#### Results

* Optimal conditions were determined for each reactor type, including:
  + Initial concentrations.
  + Flow rates.
  + Temperature.
  + Reaction time

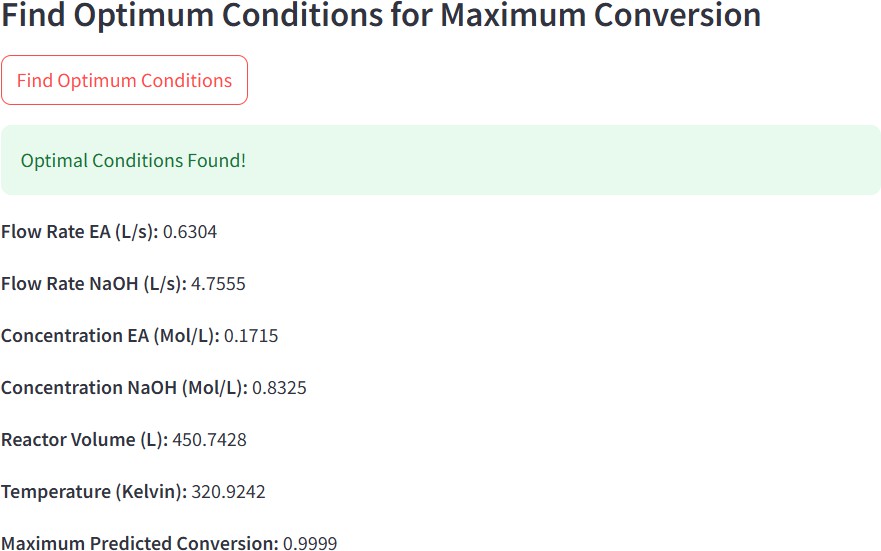
###### Result pallet of reactor conversion predictor:

****

*Fig. 27Output pallet*

###### Optimum Condition for Maximum conversion

* + 1. *For PFR*

**

*Fig. 28optimal condition for pfr*

* + 1. *FOR Batch reactor*

**

*Fig. 2Soptimal condition for batch*

#### Model Deployment

* The best-performing models (e.g., CatBoost) were saved using **joblib** for deployment.
* A **Stream lit GUI** was developed to allow users to input reactor parameters and predict conversion in real-time.

# Discussion

The results obtained from the analysis and modelling of the Batch Reactor, Continuous Stirred Tank Reactor (CSTR), and Plug Flow Reactor (PFR) provide valuable insights into the optimization of reactor performance. This section discusses the implications of the findings for reactor design, operation, and future improvements.

#### Batch Reactor

###### Key Findings:

* The conversion ((X\_A)) in the Batch Reactor is highly dependent on the initial concentrations of reactants ((𝐶𝐴𝑜 (𝐶𝐵0 )), reaction time, and temperature.
* The reaction rate constant ((k)) calculated using the Arrhenius equation showed a strong

dependence on temperature, with higher temperatures leading to faster reaction rates and higher conversions.

* The model evaluation revealed that **CatBoost Regressor** performed the best, achieving the highest (R^2) score and lowest Mean Squared Error (MSE).

###### Implications:

* **Reactor Design**: Batch reactors should be designed to allow precise control of temperature and reaction time to achieve desired conversions.
* **Operation**: Operating at higher temperatures within safe limits can significantly improve conversion, but care must be taken to avoid side reactions or thermal degradation.
* **Optimization**: The Bayesian Optimization results provide optimal operating conditions, which can be used to maximize conversion while minimizing resource usage.

#### Continuous Stirred Tank Reactor (CSTR)

###### Key Findings:

* Conversion in the CSTR is influenced by flow rates, reactant concentrations, reactor volume, and temperature.
* The residence time, calculated as the ratio of reactor volume to the total flow rate, was identified as a critical factor affecting conversion.
* The **Random Forest Regressor** and **Gradient Boosting Regressor** performed well, with Gradient Boosting showing slightly better results after hyperparameter tuning.

###### Implications:

* **Reactor Design**: Larger reactor volumes and lower flow rates can increase residence time, leading to higher conversions. However, this must be balanced against economic and operational constraints.
* **Operation**: Maintaining optimal reactant concentrations and temperatures can enhance conversion efficiency. The insights from feature importance analysis suggest prioritizing the control of reactant concentrations and residence time.
* **Optimization**: The Bayesian Optimization results provide a roadmap for achieving maximum conversion by fine-tuning operating parameters.

#### Plug Flow Reactor (PFR)

###### Key Findings:

* Conversion in the PFR is strongly influenced by flow rates, reactant concentrations, reactor volume, and temperature.
* The mole balance equations and residence time calculations revealed that higher residence times and reactant concentrations lead to better conversions.
* The **CatBoost Regressor** outperformed other models, demonstrating its ability to capture complex, non-linear relationships in the data.

###### Implications:

* **Reactor Design**: PFRs should be designed with sufficient length and volume to ensure adequate residence time for the reaction to reach completion.
* **Operation**: Operating at optimal flow rates and reactant concentrations can maximize conversion while minimizing energy and material costs.
* **Optimization**: The optimization results provide actionable insights into the best operating conditions for achieving maximum conversion.
  1. **Model Comparison and Selection**
* Across all reactor types, **CatBoost Regressor** consistently performed well, highlighting its robustness and ability to handle non-linear relationships and feature interactions.
* The comparison of models (Random Forest, Gradient Boosting, XGBoost, CatBoost, and Neural Networks) showed that ensemble methods generally outperformed simpler models like Linear Regression and SVR.
* Residual analysis and cross-validation confirmed the reliability and accuracy of the selected models.
  1. **Implications for Reactor Design and Operation**

###### Reactor Design:

* The insights from feature importance analysis and optimization can guide the design of reactors with improved efficiency and performance.
* For example, larger reactor volumes and precise control of flow rates and reactant concentrations can enhance conversion in CSTRs and PFRs.

###### Operation:

* The optimization results provide a clear set of operating conditions for maximizing conversion while minimizing resource usage.
* The ability to predict conversion under different conditions allows for real-time adjustments to operating parameters, improving flexibility and efficiency.

###### Sustainability:

* The use of machine learning models to optimize reactor performance reduces the need for extensive experimental trials, saving time, energy, and materials.
* By maximizing conversion, the approach minimizes waste and improves the overall sustainability of chemical processes.
  1. **Limitations and Future Work**

###### Limitations:

* The models rely on synthetic datasets, which may not fully capture the variability and complexity of real-world systems.
* The exclusion of stirrer speed in the CSTR analysis assumes ideal mixing, which may not hold true for all systems.

###### Future Work:

* Validate the models and optimization results with experimental data to ensure real-world applicability.
* Extend the analysis to include additional factors, such as stirrer speed for non-ideal systems or side reactions.
* Explore advanced optimization techniques, such as Genetic Algorithms or Reinforcement Learning, for more complex systems.

## Conclusion

#### Summary

This study focused on optimizing the performance of Batch Reactors, Continuous Stirred Tank Reactors (CSTR), and Plug Flow Reactors (PFR) using Artificial Intelligence (AI) and Machine Learning (ML) techniques. The saponification reaction between ethyl acetate and sodium hydroxide was used as the model reaction due to its well-defined kinetics and industrial relevance. Synthetic datasets were generated based on kinetic equations, and various machine learning models, including Random Forest, Gradient Boosting, CatBoost, and Neural Networks, were trained to predict conversion ((X\_A)). Bayesian Optimization was employed to identify the optimal operating conditions for maximizing conversion. The results demonstrated that CatBoost consistently outperformed other models in terms of accuracy, with the highest (R^2) and lowest Mean Squared Error (MSE). The optimization results provided actionable insights into the best operating conditions for each reactor type, enabling significant improvements in conversion efficiency.

#### Contributions

This study advances reactor optimization by integrating AI/ML techniques into the traditional chemical engineering workflow. Key contributions include:

* + 1. **Data-Driven Insights**: The use of machine learning models provided a deeper understanding of the relationships between operating parameters and conversion, enabling more accurate predictions compared to traditional methods.
    2. **Optimization Framework**: The application of Bayesian Optimization offered a systematic approach to identify optimal reactor conditions, reducing the need for extensive experimental trials.
    3. **Model Comparisons**: A comprehensive evaluation of multiple machine learning models highlighted the strengths of ensemble methods like CatBoost in capturing complex, non-linear relationships in reactor performance data.
    4. **Scalability**: The methodology demonstrated scalability, making it applicable to other reactions and reactor types beyond the saponification reaction studied here.

#### Limitations

Despite its success, the study faced several challenges:

* + 1. **Synthetic Data**: The models were trained on synthetic datasets generated from kinetic equations, which may not fully capture the variability and complexity of real-world systems.
    2. **Computational Cost**: Training advanced machine learning models and performing Bayesian Optimization required significant computational resources.
    3. **Simplified Assumptions**: Certain assumptions, such as ideal mixing in the CSTR and the exclusion of stirrer speed, may limit the applicability of the results to non-ideal systems.
    4. **Experimental Validation**: The lack of experimental validation for the predicted optimal conditions limits the immediate applicability of the findings in industrial settings.

#### Future Work

To build on the findings of this study, the following directions are suggested for future research:

* + 1. **Experimental Validation**: Validate the machine learning models and optimization results with experimental data to ensure real-world applicability.
    2. **Dynamic Optimization**: Extend the methodology to include dynamic optimization for real-time adjustments in reactor operation.
    3. **Broader Applications**: Apply the framework to other reactions and reactor types, including heterogeneous and multi-phase systems.
    4. **Integration with IoT**: Incorporate real-time data from sensors and Internet of Things (IoT) devices to improve model accuracy and enable adaptive optimization.
    5. **Advanced Models**: Explore the use of advanced optimization techniques, such as Genetic Algorithms or Reinforcement Learning, for more complex systems.

# References

1. Bhalekar, A. A., & Kanawade, S. M. (2015). Kinetic study of saponification reaction in a batch reactor. *Research Journal of Chemical Sciences*, **5**(11), 17–21. <https://www.isca.in/rjcs/Archives/v5/i11/3.ISCA-RJCS-2015-137.pdf>
2. Karthik, P., & Ashok, K. (2015). Optimization of reactor performance using ANN and genetic algorithms. *International Journal of Engineering Research and Applications (IJERA)*, **5**(2, Part 2), 74–78. https://[www.ijera.com/papers/Vol5\_issue2/Part%20-%202/K502027478.pdf](http://www.ijera.com/papers/Vol5_issue2/Part%20-%202/K502027478.pdf)
3. Saini, A., & Kumar, R. (2019). Experimental kinetic study of saponification reaction in a plug flow reactor. *International Research Journal of Engineering and Technology (IRJET)*, **6**(3), 1202–1206. https://[www.irjet.net/archives/V6/i3/IRJET-V6I31210.pdf](http://www.irjet.net/archives/V6/i3/IRJET-V6I31210.pdf)
4. Rajalakshmi, R., & Sivakumar, T. (2018). Kinetic and thermodynamic study of drug degradation reactions using UV spectrophotometer. *Asian Journal of Pharmaceutical Analysis*, **8**(1), 20–24. https://ajpaonline.com/HTMLPaper.aspx?Journal=Asian%20Journal%20of%20Pharmaceutical%2 0Analysis;PID=2018-8-1-4
5. Narayan, P., & Patel, M. (2023). Optimization of chemical reactor design using AI algorithms.

*ResearchGate*. https://[www.researchgate.net/publication/387556470\_Optimization\_of\_Chemical\_Reactor\_Design](http://www.researchgate.net/publication/387556470_Optimization_of_Chemical_Reactor_Design)

\_Using\_AI\_Algorithms

1. Jahan, M. A., Hossain, S., Khan, M. S., & Rahman, M. A. (2024). A machine learning-based surrogate modelling approach for reactor optimization. *Chemical Engineering Science*, **287**, 119815. https://doi.org/10.1016/j.ces.2024.119815
2. Levenspiel, O. (1999). \*Chemical reaction engineering\* (3rd ed.). Wiley.
3. Fogler, H. S. (2016). \*Elements of chemical reaction engineering\* (5th ed.). Pearson.
4. Kumar, P., & Kotecha, K. (2021). Reactor optimization using AI techniques: A review. *Applied Sciences*, **11**(15), 6789. https://doi.org/10.3390/app11156789
5. Bayesian Optimization. (n.d.). *Bayesian optimization library documentation*. https://github.com/fmfn/BayesianOptimization
6. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, **12**, 2825–2830. https://scikit-learn.org/
7. Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: Unbiased boosting with categorical features. *Advances in Neural Information Processing Systems*, **31**. https://catboost.ai/
8. Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. https://xgboost.ai/
9. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. *OSDI*, **16**, 265–283. https://[www.tensorflow.org/](http://www.tensorflow.org/)
10. Waskom, M. L. (2021). Seaborn: Statistical data visualization. *Journal of Open Source Software*, **6**(60), 3021. https://seaborn.pydata.org/
11. Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, **9**(3), 90–95. https://matplotlib.org/
12. Wikipedia contributors. (n.d.). *Arrhenius equation*. Wikipedia. Retrieved May 14, 2025, from https://en.wikipedia.org/wiki/Arrhenius\_equation
13. LibreTexts. (2020, August 15). *Saponification*. Chemistry LibreTexts. https://chem.libretexts.org/Bookshelves/Physical\_and\_Theoretical\_Chemistry\_Textbook\_ Maps/Supplemental\_Modules\_(Physical\_and\_Theoretical\_Chemistry)/Kinetics/Reaction\_ Rates/Saponification
14. Sharma, S., & Patel, R. (2023). *AI in chemical engineering: A new paradigm for Industry*

*4.0*. ResearchGate.

https://[www.researchgate.net/publication/387556470\_Optimization\_of\_Chemical\_Reactor](http://www.researchgate.net/publication/387556470_Optimization_of_Chemical_Reactor)

\_Design\_Using\_AI\_Algorithms

# Appendix: A

**# Data set generation:**

1. import numpy as np
2. np.random.seed(42) # For reproducibility
3. import pandas as pd
4. import matplotlib.pyplot as plt
5. import seaborn as sns
6. import statsmodels.api as sm
7. import statsmodels.formula.api as smf
8. import statsmodels.stats.api as sms
9. import statsmodels.stats.proportion as ssp
10. import statsmodels.stats.multitest as smm
11. import statsmodels.stats.diagnostic as smd
12. Ea = 29775 #J/mol
13. k0 = 27038 #L·mol⁻¹·s⁻¹
14. n=1000 # number of samples
15. # Sample data generation
16. sample = {
17. "initial\_concentration": {
18. "ethyl\_acetate": np.round(np.random.uniform(0.01, 0.1, n), 3), # Ethyl acetate concentration in mol/L
19. "NaOH": np.round(np.random.uniform(0.01, 0.1, n), 3) },
20. "temperature": np.round(np.random.uniform(295, 330, n), 2), # Temperature in Kelvin (22°C to 57°C)
21. "time":np.round(np.random.uniform(300, 1800, n), 2) # time in randge from 5 to 30 minutes to vary conversion from 0.1 to 0.99
22. }
23. M=np.round((sample["initial\_concentration"]["NaOH"]/sample["initial\_concentration"]["ethyl\_acetate"]),2)
24. def calculate\_k():
25. temps = np.array(sample["temperature"])
26. k\_vals = k0 \* np.exp(-Ea/ (R \* temps))
27. return k\_vals
28. ​
29. k\_values = calculate\_k()
30. print("k values:")
31. k\_values
32. ​
33. ​
34. def conversion\_of\_A():
35. ethyl\_acetate\_conc = sample["initial\_concentration"]['ethyl\_acetate']
36. time = sample["time"] 37.
37. # Case where M != 1
38. exp\_term = np.exp(k\_values \* time \* ethyl\_acetate\_conc \* (M - 1))
39. numerator = M \* (exp\_term - 1)
40. denominator = M \* exp\_term - 1 42.
41. # Avoid division by zero or invalid values in the denominator
42. result = np.divide(numerator, denominator, out=np.zeros\_like(numerator), where=denominator != 0) 45.
43. # Case where M == 1
44. mask = (M == 1)
45. result[mask] = (k\_values[mask] \* ethyl\_acetate\_conc[mask] \* tim 49.

50.

1. e[mask]) / (
2. 1 + k\_values[mask] \* ethyl\_acetate\_conc[mask] \* time[mask]

53. )

54.

55. return np.round(result, 4)

56.

57. X\_A = conversion\_of\_A()

58. X\_A # Print the first 10 conversion values 59.

# 1. Create a DataFrame with the required features

1. data = {
2. "Ethyl\_acetate(Ca0)": sample["initial\_concentration"]["ethyl\_acetate"],
3. "NaOH(Cb0)": sample["initial\_concentration"]["NaOH"],
4. "Temperature(T(°C))": sample["temperature"] - 273.15, # Convert Kelvin to Celsius
5. "Time\_of\_reaction(min)": sample["time"] / 60, # Convert seconds to minutes
6. "Conversion(X\_A)": X\_A

8. }

9.

10. df = pd.DataFrame(data) 11.

1. # Save the DataFrame to a CSV file
2. df.to\_csv("reaction\_data.csv", index=False)
3. ​
4. print("Dataset created and saved as 'reaction\_data.csv'")
5. df

**# Data cleaning and visulation**

1. import seaborn as sns
2. from sklearn.preprocessing import StandardScaler 3.

4. import matplotlib.pyplot as plt 5.

1. # Check for missing values
2. print("Missing values in the dataset:")
3. df.isnull().sum()
4. ​
5. # Statistical summary of the dataset
6. print("\nStatistical summary:")
7. df.describe()
8. ​
9. # Correlation matrix to understand relationships between features
10. correlation\_matrix = df.corr()
11. print("\nCorrelation matrix:")
12. correlation\_matrix
13. # Visualize the correlation matrix
14. plt.figure(figsize=(10, 8))
15. sns.heatmap(correlation\_matrix, annot=True, cmap="coolwarm", fmt=".2f")
16. plt.title("Correlation Matrix Heatmap")
17. plt.show()
18. import seaborn as sns
19. import matplotlib.pyplot as plt 25.
20. # Pair plot to visualize relationships between features
21. sns.pairplot(df, diag\_kind='kde', corner=True,
22. plot\_kws={'alpha': 0.6, 's': 50, 'edgecolor': 'k'},
23. diag\_kws={'shade': True})
24. ​
25. plt.suptitle("Pair Plot of Batch Reactor Features", y=1.02)
26. plt.show()
27. ​

**# scaling ,training ,testing , selecting features,ploting:-**

1. scaler = StandardScaler()
2. scaled\_features = scaler.fit\_transform(df.drop(columns=["Conversion(X\_A)"]))
3. scaled\_df = pd.DataFrame(scaled\_features, columns=df.columns[:-1])
4. scaled\_df["Conversion(X\_A)"] = df["Conversion(X\_A)"] 5.
5. print("\nScaled dataset preview:")
6. scaled\_df.head()
7. from sklearn.ensemble import RandomForestRegressor
8. from sklearn.model\_selection import train\_test\_split
9. from sklearn.metrics import mean\_squared\_error, r2\_score 11.
10. # Split the data into features and target
11. X = scaled\_df.drop(columns=["Conversion(X\_A)"])
12. y = scaled\_df["Conversion(X\_A)"] 15.
13. # Split into training and testing sets
14. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) 18.
15. # Train a Random Forest Regressor
16. rf\_model = RandomForestRegressor(random\_state=42, n\_estimators=100)
17. rf\_model.fit(X\_train, y\_train)
18. ​
19. # Predict on the test set
20. y\_pred = rf\_model.predict(X\_test) 25.
21. # Evaluate the model
22. mse = mean\_squared\_error(y\_test, y\_pred)
23. r2 = r2\_score(y\_test, y\_pred)
24. print(f"Mean Squared Error: {mse}")
25. print(f"R-squared: {r2}")
26. ​
27. feature\_importances = rf\_model.feature\_importances\_
28. importance\_df = pd.DataFrame({
29. "Feature": X.columns,
30. "Importance": feature\_importances
31. }).sort\_values(by="Importance", ascending=False)
32. ​
33. print("\nFeature Importances:")
34. importance\_df
35. plt.figure(figsize=(8, 6))
36. sns.barplot(x="Importance", y="Feature", data=importance\_df, palette="viridis")
37. plt.title("Feature Importance")
38. plt.xlabel("Importance")
39. plt.ylabel("Feature")
40. plt.show()
41. # Plotting actual vs predicted values
42. plt.figure(figsize=(10, 6))
43. plt.scatter(y\_test, y\_pred, color='blue', alpha=0.6)
44. plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', lw=2)
45. plt.title("Actual vs Predicted Conversion(X\_A)")
46. plt.xlabel("Actual Conversion(X\_A)")
47. plt.ylabel("Predicted Conversion(X\_A)")
48. plt.xlim(y.min(), y.max())
49. plt.ylim(y.min(), y.max())
50. plt.grid()
51. plt.show()
52. # Residuals plot 58.

**# Hypertuning,Different model training,testing,error calculation**

1. from sklearn.model\_selection import GridSearchCV 2.

1. # Define the parameter grid for Random Forest
2. param\_grid = {
3. 'n\_estimators': [100, 200, 300],
4. 'max\_depth': [None, 10, 20, 30],
5. 'min\_samples\_split': [2, 5, 10],
6. 'min\_samples\_leaf': [1, 2, 4],
7. 'bootstrap': [True, False]

10. }

11.

1. # Perform Grid Search
2. grid\_search = GridSearchCV(estimator=rf\_model, param\_grid=param\_grid, cv=5, scoring='r2', n\_jobs=-1, verbose=2)
3. grid\_search.fit(X\_train, y\_train)
4. ​
5. # Get the best parameters and best model
6. best\_rf\_model = grid\_search.best\_estimator\_
7. print("Best Parameters for Random Forest:", grid\_search.best\_params\_) 19.
8. # Evaluate the improved model
9. y\_pred\_improved = best\_rf\_model.predict(X\_test)
10. mse\_improved = mean\_squared\_error(y\_test, y\_pred\_improved)
11. r2\_improved = r2\_score(y\_test, y\_pred\_improved)
12. print(f"Improved Random Forest - Mean Squared Error: {mse\_improved}")
13. print(f"Improved Random Forest - R-squared: {r2\_improved}") 26.

27. from sklearn.ensemble import GradientBoostingRegressor 28.

1. # Train a Gradient Boosting Regressor
2. gb\_model = GradientBoostingRegressor(random\_state=42, n\_estimators=100, learning\_rate=0.1, max\_depth=3)
3. gb\_model.fit(X\_train, y\_train)
4. ​
5. # Predict on the test set
6. y\_pred\_gb = gb\_model.predict(X\_test)

35.

1. # Evaluate the Gradient Boosting model
2. mse\_gb = mean\_squared\_error(y\_test, y\_pred\_gb)
3. r2\_gb = r2\_score(y\_test, y\_pred\_gb) 39.
4. print(f"Gradient Boosting - Mean Squared Error: {mse\_gb}")
5. print(f"Gradient Boosting - R-squared: {r2\_gb}") 42.
6. # Compare models
7. print("\nModel Comparison:")
8. print(f"Random Forest - MSE: {mse}, R2: {r2}")
9. print(f"Gradient Boosting - MSE: {mse\_gb}, R2: {r2\_gb}") 47.
10. if r2\_gb > r2:
11. print("Gradient Boosting performs better for this reactor performance optimization.")
12. else:
13. print("Random Forest performs better for this reactor performance optimization.")
14. from sklearn.model\_selection import GridSearchCV 53.
15. # Define the parameter grid for Gradient Boosting Regressor
16. param\_grid\_gb = {
17. 'n\_estimators': [100, 200, 300],
18. 'learning\_rate': [0.01, 0.1, 0.2],
19. 'max\_depth': [3, 5, 7],
20. 'min\_samples\_split': [2, 5, 10],
21. 'min\_samples\_leaf': [1, 2, 4]

61. }

62.

1. # Perform Grid Search
2. grid\_search\_gb = GridSearchCV(estimator=gb\_model, param\_grid=param\_grid\_gb, cv=5, scoring='r2', n\_jobs=-1, verbose=2)
3. grid\_search\_gb.fit(X\_train, y\_train)
4. ​
5. # Get the best parameters and best model
6. best\_gb\_model = grid\_search\_gb.best\_estimator\_
7. print("Best Parameters for Gradient Boosting:", grid\_search\_gb.best\_params\_) 70.
8. # Evaluate the improved model
9. y\_pred\_gb\_improved = best\_gb\_model.predict(X\_test)
10. mse\_gb\_improved = mean\_squared\_error(y\_test, y\_pred\_gb\_improved)
11. r2\_gb\_improved = r2\_score(y\_test, y\_pred\_gb\_improved)
12. print(f"Improved Gradient Boosting - Mean Squared Error: {mse\_gb\_improved}")
13. print(f"Improved Gradient Boosting - R-squared: {r2\_gb\_improved}")
14. from sklearn.svm import SVR
15. from sklearn.linear\_model import LinearRegression
16. from sklearn.metrics import mean\_squared\_error, r2\_score 80.
17. # Train a Support Vector Regressor (SVR)
18. svr\_model = SVR(kernel='rbf', C=1.0, epsilon=0.1)
19. svr\_model.fit(X\_train, y\_train)
20. ​
21. # Predict on the test set using SVR
22. y\_pred\_svr = svr\_model.predict(X\_test) 87.
23. # Evaluate the SVR model
24. mse\_svr = mean\_squared\_error(y\_test, y\_pred\_svr)
25. r2\_svr = r2\_score(y\_test, y\_pred\_svr)
26. print(f"SVR - Mean Squared Error: {mse\_svr}")
27. print(f"SVR - R-squared: {r2\_svr}") 93.
28. # Train a Linear Regression model
29. lr\_model = LinearRegression()
30. lr\_model.fit(X\_train, y\_train)
31. ​
32. # Predict on the test set using Linear Regression
33. y\_pred\_lr = lr\_model.predict(X\_test) 100.
34. # Evaluate the Linear Regression model
35. mse\_lr = mean\_squared\_error(y\_test, y\_pred\_lr)
36. r2\_lr = r2\_score(y\_test, y\_pred\_lr)
37. print(f"Linear Regression - Mean Squared Error: {mse\_lr}")
38. print(f"Linear Regression - R-squared: {r2\_lr}") 106.
39. # Compare all models
40. print("\nModel Comparison:")
41. print(f"Random Forest - MSE: {mse}, R2: {r2}")
42. print(f"Gradient Boosting - MSE: {mse\_gb}, R2: {r2\_gb}")
43. print(f"SVR - MSE: {mse\_svr}, R2: {r2\_svr}")
44. print(f"Linear Regression - MSE: {mse\_lr}, R2: {r2\_lr}")

113.

**# ANN,Error comparison,finding optimize value using BayesianOptimization,saving model**

* 1. import tensorflow as tf
  2. from tensorflow.keras.models import Sequential
  3. from tensorflow.keras.layers import Dense, Dropout
  4. from sklearn.metrics import mean\_squared\_error, r2\_score 5.

1. # Define the neural network model
2. def create\_model():
3. model = Sequential([
4. Dense(64, activation='relu', input\_shape=(X\_train.shape[1],)),
5. Dropout(0.2),
6. Dense(32, activation='relu'),
7. Dropout(0.2),
8. Dense(1, activation='linear') # Output layer for regression

14. ])

1. model.compile(optimizer='adam', loss='mse', metrics=['mae'])
2. return model
3. ​
4. # Create the model
5. nn\_model = create\_model() 20.
6. # Train the model
7. history = nn\_model.fit(X\_train, y\_train, validation\_split=0.2, epochs=100, batch\_size=32, verbose=1) 23.
8. # Predict on the test set
9. y\_pred\_nn = nn\_model.predict(X\_test).flatten() 26.
10. # Evaluate the model
11. mse\_nn = mean\_squared\_error(y\_test, y\_pred\_nn)
12. r2\_nn = r2\_score(y\_test, y\_pred\_nn) 30.
13. print(f"Neural Network - Mean Squared Error: {mse\_nn}")
14. print(f"Neural Network - R-squared: {r2\_nn}")
15. import seaborn as sns
16. import pandas as pd 35.
17. # Import necessary libraries
18. import matplotlib.pyplot as plt 38.
19. # Create a DataFrame to store model performance metrics
20. model\_comparison = pd.DataFrame({
21. "Model": [
22. "Random Forest", "Gradient Boosting", "Improved Gradient Boosting",
23. "CatBoost", "XGBoost", "SVR", "Linear Regression", "Neural Network"

44. ],

1. "MSE": [
2. mse, mse\_gb, mse\_gb\_improved, mse\_catboost, mse\_xgb, mse\_svr, mse\_lr, mse\_nn

47. ],

1. "R²": [
2. r2, r2\_gb, r2\_gb\_improved, r2\_catboost, r2\_xgb, r2\_svr, r2\_lr, r2\_nn

50. ]

51. })

52.

1. # Sort by R² for better visualization
2. model\_comparison = model\_comparison.sort\_values(by="R²", ascending=False) 55.
3. # Plot MSE and R² for all models
4. plt.figure(figsize=(14, 6))
5. ​
6. # MSE Bar Plot
7. plt.subplot(1, 2, 1)
8. sns.barplot(x="MSE", y="Model", data=model\_comparison, palette="viridis")
9. plt.title("Model Comparison - Mean Squared Error (MSE)")
10. plt.xlabel("Mean Squared Error")
11. plt.ylabel("Model")
12. ​
13. # R² Bar Plot
14. plt.subplot(1, 2, 2)
15. sns.barplot(x="R²", y="Model", data=model\_comparison, palette="viridis")
16. plt.title("Model Comparison - R² Score")
17. plt.xlabel("R² Score")
18. plt.ylabel("Model")
19. ​
20. plt.tight\_layout()
21. plt.show()
22. ​
23. # Scatter plot for Actual vs Predicted values for the best model (CatBoost)
24. plt.figure(figsize=(10, 6))
25. plt.scatter(y\_test, y\_pred\_catboost, color='blue', alpha=0.6, label="Predicted")
26. plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'r--', lw=2, label="Ideal")
27. plt.title("Actual vs Predicted Conversion(X\_A) - CatBoost")
28. plt.xlabel("Actual Conversion(X\_A)")
29. plt.ylabel("Predicted Conversion(X\_A)")
30. plt.legend()
31. plt.grid()
32. plt.show()
33. ​
34. # Residuals plot for the best model (CatBoost)
35. residuals\_catboost = y\_test - y\_pred\_catboost
36. plt.figure(figsize=(10, 6))
37. sns.histplot(residuals\_catboost, kde=True, color="purple", bins=20)
38. plt.title("Residuals Distribution - CatBoost")
39. plt.xlabel("Residuals")
40. plt.ylabel("Frequency")
41. plt.grid()
42. plt.show()
43. ​
44. # Print the best model based on R²
45. best\_model = model\_comparison.iloc[0]
46. print(f"The best model is {best\_model['Model']} with R² = {best\_model['R²']:.4f} and MSE = {best\_model['MSE']:.4f}.")
47. # Plot feature importance for the best model (CatBoost in this case)
48. plt.figure(figsize=(8, 6))
49. sns.barplot(x="Importance", y="Feature", data=importance\_df, palette="viridis")
50. plt.title("Feature Importance")
51. plt.xlabel("Importance")
52. plt.ylabel("Feature")
53. plt.show()
54. from bayes\_opt import BayesianOptimization
55. import numpy as np
56. import pandas as pd 110.
57. # Define the objective function for Bayesian Optimization
58. def objective\_function(Ca0, Cb0, T, time):
59. # Prepare the input data for prediction
60. input\_data = np.array([[Ca0, Cb0, T, time]])
61. # Or use DataFrame if your model was trained on DataFrame inputs
62. # input\_data = pd.DataFrame([[Ca0, Cb0, T, time]], columns=['Ethyl\_acetate(Ca0)', 'NaOH(Cb0)', 'Temperature(T(°C))', 'Time\_of\_reaction(min)'])
63. ​
64. # Predict the conversion rate using the CatBoost model
65. predicted\_conversion = catboost\_model.predict(input\_data) 120.
66. # Return as scalar float
67. return float(predicted\_conversion[0])
68. ​
69. # Define the bounds for the decision variables
70. pbounds = {
71. 'Ca0': (scaled\_df["Ethyl\_acetate(Ca0)"].min(), scaled\_df["Ethyl\_acetate(Ca0)"].max()),
72. 'Cb0': (scaled\_df["NaOH(Cb0)"].min(), scaled\_df["NaOH(Cb0)"].max()),
73. 'T': (scaled\_df["Temperature(T(°C))"].min(), scaled\_df["Temperature(T(°C))"].max()),
74. 'time': (scaled\_df["Time\_of\_reaction(min)"].min(), scaled\_df["Time\_of\_reaction(min)"].max())

130. }

131.

1. # Initialize the Bayesian Optimizer
2. optimizer = BayesianOptimization(
3. f=objective\_function,
4. pbounds=pbounds,
5. random\_state=42

137. )

138.

1. # Perform the optimization
2. optimizer.maximize(
3. init\_points=10, # Number of random initial points
4. n\_iter=50 # Number of iterations for optimization 143. )

144.

1. # Extract the optimal conditions
2. optimal\_conditions = optimizer.max['params']
3. optimal\_conversion = optimizer.max['target'] 148.

149. print("Optimal Conditions:")

1. print(f"Ethyl\_acetate(Ca0): {optimal\_conditions['Ca0']}")
2. print(f"NaOH(Cb0): {optimal\_conditions['Cb0']}")
3. print(f"Temperature(T(°C)): {optimal\_conditions['T']}")
4. print(f"Time\_of\_reaction(min): {optimal\_conditions['time']}")
5. print(f"Maximum Conversion(X\_A): {optimal\_conversion}") 155.

156. import joblib

157.

1. # Save the trained model
2. joblib.dump(catboost\_model, 'catboost\_model.pkl')
3. joblib.dump(scaler, 'scaler.pkl')
4. ​

# Appendix: B

**CSTR**

**# Data set generation: -**

1. import numpy as np
2. import pandas as pd 3.
3. R=8.314
4. Ea=29775
5. k0=27038
6. ​
7. # Set the number of samples to generate
8. num\_samples = 1000
9. ​
10. # Generate random data within the specified ranges
11. np.random.seed(42) # For reproducibility
12. data = {
13. 'Flow\_Rate\_EA (L/s)': np.random.uniform(0.001, 10, num\_samples),
14. 'Flow\_Rate\_NaOH (L/s)': np.random.uniform(0.001, 10, num\_samples),
15. 'Conc\_EA (Mol/L)': np.random.uniform(0.01, 1, num\_samples),
16. 'Conc\_NaOH (Mol/L)': np.random.uniform(0.01, 1, num\_samples),
17. 'Reactor\_Vol (L)': np.random.uniform(0.5, 500, num\_samples),
18. 'Temp (Kelvin)': np.random.uniform(25, 60, num\_samples) + 273.15

20. }

21. # Create a DataFrame 22.

23. CSTR\_Data\_Set = pd.DataFrame(data) 24.

1. # Display the first few rows of the generated data
2. CSTR\_Data\_Set
3. # Data Cleaning Step
4. # Handle missing values
5. CSTR\_Data\_Set.dropna(inplace=True)
6. ​
7. # Remove outliers using Z-score (optional, based on your dataset)
8. from scipy.stats import zscore
9. z\_scores = CSTR\_Data\_Set.apply(zscore)
10. CSTR\_Data\_Set = CSTR\_Data\_Set[(z\_scores < 3).all(axis=1)] # Keep rows with Z-scores < 3 35.
11. # Reset index after cleaning
12. CSTR\_Data\_Set.reset\_index(drop=True, inplace=True)
13. ​
14. # Display cleaned data
15. print("Cleaned Data Summary:")
16. CSTR\_Data\_Set.describe()
17. ​
18. residence\_time = CSTR\_Data\_Set['Reactor\_Vol (L)'] / (CSTR\_Data\_Set['Flow\_Rate\_EA (L/s)'] +CSTR\_Data\_Set['Flow\_Rate\_NaOH (L/s)'])
19. residence\_time.head()
20. ​
21. M=CSTR\_Data\_Set['Conc\_NaOH (Mol/L)']/CSTR\_Data\_Set['Conc\_EA (Mol/L)']
22. M.head()
23. const\_term=(1 + M + 1/(M\*CSTR\_Data\_Set['Conc\_EA (Mol/L)']\*residence\_time))
24. X\_A=(const\_term -np.sqrt(const\_term\*\*2 - 4\*M))/2
25. CSTR\_Data\_Set['X\_A'] = X\_A
26. CSTR\_Data\_Set['X\_A'] = CSTR\_Data\_Set['X\_A'].clip(lower=0, upper=1)
27. CSTR\_Data\_Set['X\_A'].head()

53.

54. random\_data\_df = CSTR\_Data\_Set.sample(frac=1, random\_state=42).reset\_index(drop=True) 55.

1. CSTR\_Data\_Set.to\_csv('CSTR data set.csv', index=False)
2. CSTR\_Data\_Set
3. ​

**# visulation,EDA,train,test ,error evaluation:-**

1. import seaborn as sns
2. import matplotlib.pyplot as plt 3.
3. # Set the style for seaborn
4. sns.set(style="whitegrid")
5. ​
6. # Heatmap: Identify correlations
7. plt.figure(figsize=(10, 8))
8. correlation\_matrix = CSTR\_Data\_Set.corr()
9. sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt='.2f')
10. plt.title('Correlation Heatmap')
11. plt.show()
12. ​
13. # Pair Plots: Explore variable interactions
14. sns.pairplot(CSTR\_Data\_Set[['Flow\_Rate\_EA (L/s)', 'Flow\_Rate\_NaOH (L/s)', 'Conc\_EA (Mol/L)',
15. 'Conc\_NaOH (Mol/L)', 'Reactor\_Vol (L)', 'Temp (Kelvin)', 'X\_A']],
16. diag\_kind='kde', corner=True)
17. plt.suptitle('Pair Plot of Key Variables', y=1.02)
18. plt.show()
19. from sklearn.model\_selection import train\_test\_split, cross\_val\_score
20. from sklearn.ensemble import RandomForestRegressor
21. from sklearn.metrics import mean\_squared\_error, r2\_score, make\_scorer
22. from sklearn.preprocessing import StandardScaler
23. import pandas as pd 25.
24. # Step 1: Prepare the data
25. X = CSTR\_Data\_Set[['Flow\_Rate\_EA (L/s)', 'Flow\_Rate\_NaOH (L/s)', 'Conc\_EA (Mol/L)',
26. 'Conc\_NaOH (Mol/L)', 'Reactor\_Vol (L)', 'Temp (Kelvin)']]
27. y = CSTR\_Data\_Set['X\_A']
28. ​
29. # Split the data into training and testing sets
30. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) 33.
31. # Scale the features
32. scaler = StandardScaler()
33. X\_train\_scaled = scaler.fit\_transform(X\_train)
34. X\_test\_scaled = scaler.transform(X\_test) 38.
35. # Step 2: Perform K-Fold Cross-Validation
36. model = RandomForestRegressor(n\_estimators=100, random\_state=42) 41.
37. # Define a custom scoring function (negative MSE for compatibility with cross\_val\_score)
38. scorer = make\_scorer(mean\_squared\_error, greater\_is\_better=False) 44.
39. # Perform k-fold cross-validation
40. k = 5 # Number of folds
41. cv\_scores = cross\_val\_score(model, X\_train\_scaled, y\_train, cv=k, scoring=scorer) 48.
42. # Convert negative MSE to positive
43. cv\_scores = -cv\_scores 51.
44. # Print the cross-validation results
45. print(f"Cross-Validation Scores (MSE): {cv\_scores}")
46. print(f"Mean MSE: {cv\_scores.mean():.4f}")
47. print(f"Standard Deviation of MSE: {cv\_scores.std():.4f}") 56.
48. # Step 3: Train a machine learning model
49. model.fit(X\_train\_scaled, y\_train)
50. ​
51. # Step 4: Evaluate the model
52. y\_pred = model.predict(X\_test\_scaled)
53. mse = mean\_squared\_error(y\_test, y\_pred)
54. r2 = r2\_score(y\_test, y\_pred) 64.
55. print(f"Mean Squared Error: {mse}")
56. print(f"R² Score: {r2}") 67.
57. # Step 5: Optimize reactor performance
58. # Example: Predict conversion for a new set of operating conditions
59. new\_conditions = pd.DataFrame({
60. 'Flow\_Rate\_EA (L/s)': [5],
61. 'Flow\_Rate\_NaOH (L/s)': [5],
62. 'Conc\_EA (Mol/L)': [0.5],
63. 'Conc\_NaOH (Mol/L)': [0.5],
64. 'Reactor\_Vol (L)': [100],
65. 'Temp (Kelvin)': [320]

77. })

78.

1. # Scale the new conditions
2. new\_conditions\_scaled = scaler.transform(new\_conditions) 81.
3. # Predict conversion
4. predicted\_conversion = model.predict(new\_conditions\_scaled)
5. print(f"Predicted Conversion: {predicted\_conversion[0]}")
6. from sklearn.linear\_model import LinearRegression
7. from sklearn.ensemble import GradientBoostingRegressor
8. from sklearn.svm import SVR
9. from sklearn.metrics import mean\_squared\_error, r2\_score 89.
10. # Define models to train
11. models = {
12. "Random Forest": RandomForestRegressor(n\_estimators=100, random\_state=42),
13. "Linear Regression": LinearRegression(),
14. "Gradient Boosting": GradientBoostingRegressor(random\_state=42),
15. "Support Vector Regressor": SVR()

96. }

97.

1. # Train and evaluate each model
2. results = {}
3. for name, model in models.items():

101.

102.

103.

104.

105.

106.

107.

108.

109.

110.

111.

112.

113.

# Train the model model.fit(X\_train\_scaled, y\_train)

# Predict on the test set

y\_pred = model.predict(X\_test\_scaled)

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred) r2 = r2\_score(y\_test, y\_pred)

# Store the results

results[name] = {"MSE": mse, "R²": r2}

1. # Display the results
2. for name, metrics in results.items():
3. print(f"{name}:")
4. print(f" Mean Squared Error: {metrics['MSE']}")
5. print(f" R² Score: {metrics['R²']}")
6. print()
7. ​
8. # Select the best model based on R² score
9. best\_model\_name = max(results, key=lambda x: results[x]["R²"])
10. print(f"The best model is {best\_model\_name} with R² Score: {results[best\_model\_name]['R²']}") 124.

125.

**# comarision of error,residual analysis,search for optimum fitures,model saving**

1. import seaborn as sns
2. import pandas as pd 3.

4. import matplotlib.pyplot as plt 5.

1. # Create a DataFrame to store model performance metrics
2. model\_comparison\_cstr = pd.DataFrame({
3. "Model": [
4. "Random Forest", "Gradient Boosting", "Support Vector Regressor", "Linear Regression"

10. ],

1. "MSE": [
2. results["Random Forest"]["MSE"],
3. results["Gradient Boosting"]["MSE"],
4. results["Support Vector Regressor"]["MSE"],
5. results["Linear Regression"]["MSE"]

16. ],

1. "R²": [
2. results["Random Forest"]["R²"],
3. results["Gradient Boosting"]["R²"],
4. results["Support Vector Regressor"]["R²"],
5. results["Linear Regression"]["R²"]

22. ]

23. })

24.

1. # Sort by R² for better visualization
2. model\_comparison\_cstr = model\_comparison\_cstr.sort\_values(by="R²", ascending=False) 27.
3. # Plot MSE and R² for all models
4. plt.figure(figsize=(14, 6))
5. ​
6. # MSE Bar Plot
7. plt.subplot(1, 2, 1)
8. sns.barplot(x="MSE", y="Model", data=model\_comparison\_cstr, palette="viridis")
9. plt.title("Model Comparison - Mean Squared Error (MSE)")
10. plt.xlabel("Mean Squared Error")
11. plt.ylabel("Model")
12. ​
13. # R² Bar Plot
14. plt.subplot(1, 2, 2)
15. sns.barplot(x="R²", y="Model", data=model\_comparison\_cstr, palette="viridis")
16. plt.title("Model Comparison - R² Score")
17. plt.xlabel("R² Score")
18. plt.ylabel("Model")
19. ​
20. plt.tight\_layout()
21. plt.show()
22. ​
23. # Scatter plot for Actual vs Predicted values for the best model (Gradient Boosting)
24. y\_pred\_best = model.predict(X\_test\_scaled) # Assuming Gradient Boosting is the best model
25. plt.figure(figsize=(10, 6))
26. plt.scatter(y\_test, y\_pred\_best, color='blue', alpha=0.6, label="Predicted")
27. plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'r--', lw=2, label="Ideal")
28. plt.title("Actual vs Predicted Conversion (X\_A) - Gradient Boosting")
29. plt.xlabel("Actual Conversion (X\_A)")
30. plt.ylabel("Predicted Conversion (X\_A)")
31. plt.legend()
32. plt.grid()
33. plt.show()
34. ​
35. # Residuals plot for the best model (Gradient Boosting)
36. residuals\_best = y\_test - y\_pred\_best
37. plt.figure(figsize=(10, 6))
38. sns.histplot(residuals\_best, kde=True, color="purple", bins=20)
39. plt.title("Residuals Distribution - Gradient Boosting")
40. plt.xlabel("Residuals")
41. plt.ylabel("Frequency")
42. plt.grid()
43. plt.show()
44. ​
45. # Print the best model based on R²
46. best\_model\_cstr = model\_comparison\_cstr.iloc[0]
47. print(f"The best model is {best\_model\_cstr['Model']} with R² = {best\_model\_cstr['R²']:.4f} and MSE =

{best\_model\_cstr['MSE']:.4f}.")

1. from sklearn.model\_selection import GridSearchCV 74.
2. # Define the parameter grid
3. param\_grid = {
4. 'n\_estimators': [50, 100, 200],
5. 'max\_depth': [None, 10, 20, 30],
6. 'min\_samples\_split': [2, 5, 10],
7. 'min\_samples\_leaf': [1, 2, 4]

81. }

82.

1. # Initialize the model
2. rf = RandomForestRegressor(random\_state=42) 85.
3. # Perform GridSearchCV
4. grid\_search = GridSearchCV(estimator=rf, param\_grid=param\_grid, cv=5, scoring='r2', verbose=2, n\_jobs=-1)
5. grid\_search.fit(X\_train\_scaled, y\_train)
6. ​
7. # Best parameters and score
8. print("Best Parameters:", grid\_search.best\_params\_)
9. print("Best R² Score:", grid\_search.best\_score\_) 93.
10. # Use the best model
11. best\_CSTR\_model = grid\_search.best\_estimator\_
12. import joblib

97.

1. # Save the model
2. joblib.dump(best\_CSTR\_model, 'best\_CSTR\_model.pkl')
3. ​
4. ​

### Appendix: C

**PFR**

**# Data set generation**

1. import numpy as np
2. import pandas as pd 3.
3. R=8.314
4. Ea=29775
5. k0=27038
6. ​
7. # Set the number of samples to generate
8. num\_samples = 1000
9. np.random.seed(42) # For reproducibility
10. # Generate random data within the specified ranges 12.
11. data = {
12. 'Flow\_Rate\_EA (L/s)': np.random.uniform(0.001, 10, num\_samples),
13. 'Flow\_Rate\_NaOH (L/s)': np.random.uniform(0.001, 10, num\_samples),
14. 'Conc\_EA (Mol/L)': np.random.uniform(0.01, 1, num\_samples),
15. 'Conc\_NaOH (Mol/L)': np.random.uniform(0.01, 1, num\_samples),
16. 'Reactor\_Vol (L)': np.random.uniform(0.5, 500, num\_samples),
17. 'Temp (Kelvin)': np.random.uniform(25, 60, num\_samples) + 273.15

20. }

1. # Create a DataFrame
2. random\_data\_df = pd.DataFrame(data) 23.
3. # Display the first few rows of the generated data
4. random\_data\_df
5. def calculate\_reaction\_constant(T):
6. return k0 \* np.exp(-Ea / (R \* T))
7. k = calculate\_reaction\_constant(random\_data\_df['Temp (Kelvin)'])
8. k.head()
9. const\_value = np.exp(k \* (M - 1) \* random\_data\_df['Conc\_EA (Mol/L)'] \* residence\_time)
10. X\_A=M\*(const\_value-1)/(M\*const\_value-1)
11. random\_data\_df['Conversion'] = X\_A
12. random\_data\_df['Conversion'] = random\_data\_df['Conversion'].clip(upper=1.0)
13. random\_data\_df['Conversion'] = random\_data\_df['Conversion'].clip(lower=0.0)
14. # Display the first few rows of the generated data with conversion 36.
15. # Save the DataFrame to a CSV file
16. random\_data\_df.to\_csv('PFR data set.csv', index=False)
17. random\_data\_df
18. ​

**# Data cleaning,EDA,Train,test,Error finding,cross-validation,Feature importance analysis**

1. const\_value = np.exp(k \* (M - 1) \* random\_data\_df['Conc\_EA (Mol/L)'] \* residence\_time)
2. X\_A=M\*(const\_value-1)/(M\*const\_value-1)
3. random\_data\_df['Conversion'] = X\_A
4. random\_data\_df['Conversion'] = random\_data\_df['Conversion'].clip(upper=1.0)
5. random\_data\_df['Conversion'] = random\_data\_df['Conversion'].clip(lower=0.0)
6. # Display the first few rows of the generated data with conversion 7.
7. # Save the DataFrame to a CSV file
8. random\_data\_df.to\_csv('PFR data set.csv', index=False)
9. random\_data\_df
10. import seaborn as sns
11. import matplotlib.pyplot as plt 13.
12. # Set the style for seaborn
13. sns.set(style="whitegrid")
14. ​
15. # Scatter Plot: Temperature vs Conversion
16. plt.figure(figsize=(12, 6))
17. sns.scatterplot(data=random\_data\_df, x='Temp (Kelvin)', y='Conversion', hue='Flow\_Rate\_EA (L/s)', palette='viridis')
18. plt.title('Temperature vs Conversion with Flow Rate of Ethyl Acetate')
19. plt.xlabel('Temperature (Kelvin)')
20. plt.ylabel('Conversion')
21. plt.legend(title='Flow Rate EA (L/s)', bbox\_to\_anchor=(1.05, 1), loc='upper left')
22. plt.show()
23. ​
24. # Heatmap: Correlation Matrix
25. plt.figure(figsize=(10, 8))
26. correlation\_matrix = random\_data\_df.corr()
27. sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt='.2f')
28. plt.title('Correlation Heatmap')
29. plt.show()
30. ​
31. # Pair Plot: Explore variable interactions
32. sns.pairplot(random\_data\_df[['Flow\_Rate\_EA (L/s)', 'Flow\_Rate\_NaOH (L/s)', 'Conc\_EA (Mol/L)',
33. 'Conc\_NaOH (Mol/L)', 'Reactor\_Vol (L)', 'Temp (Kelvin)', 'Conversion']],
34. diag\_kind='kde', corner=True)
35. plt.suptitle('Pair Plot of Key Variables', y=1.02)
36. plt.show()
37. from sklearn.model\_selection import train\_test\_split 40.
38. # Define features (X) and target (y)
39. X = random\_data\_df[['Flow\_Rate\_EA (L/s)', 'Flow\_Rate\_NaOH (L/s)', 'Conc\_EA (Mol/L)',
40. 'Conc\_NaOH (Mol/L)', 'Reactor\_Vol (L)', 'Temp (Kelvin)']]
41. y = random\_data\_df['Conversion'] 45.
42. # Split the data into training and testing sets
43. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)
44. from sklearn.ensemble import RandomForestRegressor
45. from sklearn.metrics import mean\_squared\_error, r2\_score 50.
46. # Train a Random Forest model
47. model = RandomForestRegressor(n\_estimators=100, random\_state=42)
48. model.fit(X\_train, y\_train)
49. ​
50. # Evaluate the model
51. y\_pred = model.predict(X\_test)
52. mse = mean\_squared\_error(y\_test, y\_pred)
53. r2 = r2\_score(y\_test, y\_pred) 59.
54. print(f"Mean Squared Error: {mse}")
55. print(f"R² Score: {r2}")
56. from sklearn.model\_selection import cross\_val\_score
57. from sklearn.metrics import make\_scorer 64.
58. # Define a custom scoring function (negative MSE for compatibility with cross\_val\_score)
59. scorer = make\_scorer(mean\_squared\_error, greater\_is\_better=False) 67.
60. # Perform k-fold cross-validation
61. k = 5 # Number of folds
62. cv\_scores = cross\_val\_score(model, X\_train, y\_train, cv=k, scoring=scorer) 71.
63. # Convert negative MSE to positive
64. cv\_scores = -cv\_scores 74.
65. # Print cross-validation results
66. print(f"Cross-Validation Scores (MSE): {cv\_scores}")
67. print(f"Mean MSE: {cv\_scores.mean():.4f}")
68. print(f"Standard Deviation of MSE: {cv\_scores.std():.4f}") 79.

**# Residual analysis,model comparison,error comparison,Hyper parameter tuning..**

1. import seaborn as sns 2.

3. # Calculate residuals

4. residuals = y\_test - y\_pred 5.

1. # Residuals Distribution
2. plt.figure(figsize=(10, 6))
3. sns.histplot(residuals, kde=True, color="purple", bins=20)
4. plt.title("Residuals Distribution - Random Forest")
5. plt.xlabel("Residuals")
6. plt.ylabel("Frequency")
7. plt.grid()
8. plt.show()
9. ​
10. # Residuals vs Predicted Values
11. plt.figure(figsize=(10, 6))
12. plt.scatter(y\_pred, residuals, alpha=0.6, color='green')
13. plt.axhline(0, color='red', linestyle='--', lw=2)
14. plt.title("Residuals vs Predicted Values - Random Forest")
15. plt.xlabel("Predicted Conversion")
16. plt.ylabel("Residuals")
17. plt.grid()
18. plt.show()
19. from sklearn.linear\_model import LinearRegression
20. from sklearn.ensemble import GradientBoostingRegressor
21. from sklearn.svm import SVR 27.
22. # Define models to train
23. models = {
24. "Random Forest": RandomForestRegressor(n\_estimators=100, random\_state=42),
25. "Linear Regression": LinearRegression(),
26. "Gradient Boosting": GradientBoostingRegressor(random\_state=42),
27. "Support Vector Regressor": SVR()

34. }

35.

1. # Train and evaluate each model
2. results = {}
3. for name, model in models.items():
4. # Train the model
5. model.fit(X\_train, y\_train)
6. ​
7. # Predict on the test set
8. y\_pred = model.predict(X\_test) 44.
9. # Evaluate the model
10. mse = mean\_squared\_error(y\_test, y\_pred)
11. r2 = r2\_score(y\_test, y\_pred) 48.
12. # Store the results
13. results[name] = {"MSE": mse, "R²": r2} 51.
14. # Display the results
15. for name, metrics in results.items():
16. print(f"{name}:")
17. print(f" Mean Squared Error: {metrics['MSE']}")
18. print(f" R² Score: {metrics['R²']}")
19. print()
20. ​
21. # Select the best model based on R² score
22. best\_model\_name = max(results, key=lambda x: results[x]["R²"])
23. print(f"The best model is {best\_model\_name} with R² Score: {results[best\_model\_name]['R²']}")
24. import seaborn as sns
25. import matplotlib.pyplot as plt
26. import pandas as pd 65.
27. # Create a DataFrame to store model performance metrics
28. model\_comparison = pd.DataFrame({
29. "Model": list(results.keys()),
30. "MSE": [metrics["MSE"] for metrics in results.values()],
31. "R²": [metrics["R²"] for metrics in results.values()]

71. })

72.

1. # Sort by R² for better visualization
2. model\_comparison = model\_comparison.sort\_values(by="R²", ascending=False) 75.
3. # Plot MSE and R² for all models
4. plt.figure(figsize=(14, 6))
5. ​
6. # MSE Bar Plot
7. plt.subplot(1, 2, 1)
8. sns.barplot(x="MSE", y="Model", data=model\_comparison, palette="viridis")
9. plt.title("Model Comparison - Mean Squared Error (MSE)")
10. plt.xlabel("Mean Squared Error")
11. plt.ylabel("Model")
12. ​
13. # R² Bar Plot
14. plt.subplot(1, 2, 2)
15. sns.barplot(x="R²", y="Model", data=model\_comparison, palette="viridis")
16. plt.title("Model Comparison - R² Score")
17. plt.xlabel("R² Score")
18. plt.ylabel("Model")
19. ​
20. plt.tight\_layout()
21. plt.show()
22. from sklearn.model\_selection import GridSearchCV 96.
23. # Define the parameter grid for Random Forest (example)
24. param\_grid = {
25. 'n\_estimators': [50, 100, 200],
26. 'max\_depth': [None, 10, 20, 30],
27. 'min\_samples\_split': [2, 5, 10],
28. 'min\_samples\_leaf': [1, 2, 4]

103. }

104.

1. # Perform Grid Search
2. grid\_search = GridSearchCV(RandomForestRegressor(random\_state=42), param\_grid, cv=5, scoring='r2', n\_jobs=-1)
3. grid\_search.fit(X\_train, y\_train)
4. ​
5. # Get the best parameters and model
6. best\_params = grid\_search.best\_params\_
7. best\_model = grid\_search.best\_estimator\_ 112.
8. print(f"Best Parameters: {best\_params}")
9. print(f"Best R² Score: {grid\_search.best\_score\_}") 115.

**# optimization,Saving…**

1. from bayes\_opt import BayesianOptimization
2. import numpy as np 3.
3. # Define the objective function for Bayesian Optimization
4. def objective\_function(flow\_rate\_ea, flow\_rate\_naoh, conc\_ea, conc\_naoh, reactor\_vol, temp):

6.

7.

8.

9.

10.

11.

12.

13.

14.

15.

16.

17.

18.

19.

20.

21.

# Prepare the input data for prediction input\_data = pd.DataFrame({

'Flow\_Rate\_EA (L/s)': [flow\_rate\_ea], 'Flow\_Rate\_NaOH (L/s)': [flow\_rate\_naoh], 'Conc\_EA (Mol/L)': [conc\_ea], 'Conc\_NaOH (Mol/L)': [conc\_naoh], 'Reactor\_Vol (L)': [reactor\_vol],

'Temp (Kelvin)': [temp]

})

# Predict the conversion using the trained model predicted\_conversion = best\_model.predict(input\_data)

# Return the predicted conversion as the objective to maximize return float(predicted\_conversion[0])

1. # Define the bounds for the decision variables
2. pbounds = {
3. 'flow\_rate\_ea': (random\_data\_df['Flow\_Rate\_EA (L/s)'].min(), random\_data\_df['Flow\_Rate\_EA (L/s)'].max()),
4. 'flow\_rate\_naoh': (random\_data\_df['Flow\_Rate\_NaOH (L/s)'].min(), random\_data\_df['Flow\_Rate\_NaOH (L/s)'].max()),
5. 'conc\_ea': (random\_data\_df['Conc\_EA (Mol/L)'].min(), random\_data\_df['Conc\_EA (Mol/L)'].max()),
6. 'conc\_naoh': (random\_data\_df['Conc\_NaOH (Mol/L)'].min(), random\_data\_df['Conc\_NaOH (Mol/L)'].max()),
7. 'reactor\_vol': (random\_data\_df['Reactor\_Vol (L)'].min(), random\_data\_df['Reactor\_Vol (L)'].max()),
8. 'temp': (random\_data\_df['Temp (Kelvin)'].min(), random\_data\_df['Temp (Kelvin)'].max())

30. }

31.

1. # Initialize the Bayesian Optimizer
2. optimizer = BayesianOptimization(
3. f=objective\_function,
4. pbounds=pbounds,
5. random\_state=42

37. )

38.

1. # Perform the optimization
2. optimizer.maximize(
3. init\_points=10, # Number of random initial points
4. n\_iter=50 # Number of iterations for optimization

43. )

44.

1. # Extract the optimal conditions
2. optimal\_conditions = optimizer.max['params']
3. optimal\_conversion = optimizer.max['target'] 48.
4. # Display the results
5. print("Optimal Conditions:")
6. print(f"Flow Rate EA (L/s): {optimal\_conditions['flow\_rate\_ea']:.4f}")
7. print(f"Flow Rate NaOH (L/s): {optimal\_conditions['flow\_rate\_naoh']:.4f}")
8. print(f"Concentration EA (Mol/L): {optimal\_conditions['conc\_ea']:.4f}")
9. print(f"Concentration NaOH (Mol/L): {optimal\_conditions['conc\_naoh']:.4f}")
10. print(f"Reactor Volume (L): {optimal\_conditions['reactor\_vol']:.4f}")
11. print(f"Temperature (Kelvin): {optimal\_conditions['temp']:.4f}")
12. print(f"Maximum Predicted Conversion: {optimal\_conversion:.4f}")
13. import pickle
14. ​
15. # Save the trained model to a file
16. with open('pfr\_model.pkl', 'wb') as f:
17. pickle.dump(best\_model, f)
18. ​
19. print("Model saved as 'pfr\_model.pkl'") 65.