**Project Title:**

Using machine learning algorithms to optimize chemical processes by analyzing data from sensors and reactors

**Name: MD TOUFIK**

**Roll: 224107206**

1. **Abstract**

In the field of chemical engineering, process optimization is crucial for maximizing efficiency, reducing costs, and improving product quality. Traditionally, chemical engineers relied on trial-and-error experimentation and manual calculations to optimize chemical processes. However, with the rise of big data and machine learning, it has become possible to analyze large amounts of data from sensors, reactors, and other sources to predict optimal reaction conditions and identify factors that affect process efficiency.

Overall, this project has the potential to improve the efficiency and effectiveness of heat transfer processes in chemical engineering, while also reducing costs and improving the quality of the product.

For the project, some of the necessary **features** that could be used as input data to train the model include:

1. Temperature
2. Pressure
3. Flow Rate
4. Concentration
5. Reactor Size
6. Catalyst Type
7. Reaction Time
8. Product Quality
9. Density
10. Viscosity
11. Solubility

It is important to note that the specific features that are necessary for the project may vary depending on the specific optimization process variables being studied and the available data. However, generally, the above features are often used in process optimization and could serve as a good starting point for building the machine learning model.

The first few rows of the dataset are shown below:

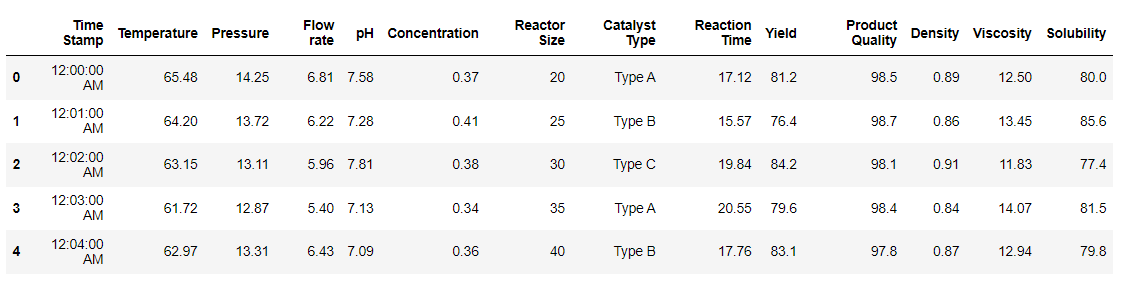


Figure 1: Dataset

***Objectiveof the project***: The *objective* of this project is to develop a model which can predict the *yield* using all the features involved. As all the data are labelled so it is a regression problem.

**Four** Regression models has been developed and trained based on the data provided. The models are:

1. Multilinear Regression Model
2. Support Vector Regression(SVR) Model
3. Decision Tree Model
4. Random Forest Model
5. **Introduction**

Process optimization refers to the systematic approach of improving and fine-tuning various aspects of a process to enhance its efficiency, productivity, and cost-effectiveness. It involves utilizing techniques, methodologies, and tools to analyze process data, identify bottlenecks, and optimize key parameters. In the context of machine learning, process optimization can be achieved by analyzing data collected from sensors, reactors, and other sources using machine learning algorithms. By leveraging machine learning techniques, such as predictive modeling, data analysis, and optimization algorithms, it becomes possible to predict optimal reaction conditions, identify factors that influence process efficiency, and reduce production costs. This data-driven approach enables process engineers to gain valuable insights, make data-informed decisions, and continuously improve the process by iteratively adjusting variables, parameters, and control strategies. Ultimately, process optimization using machine learning algorithms can lead to improved product quality, increased throughput, reduced energy consumption, and enhanced overall process performance.

The use of AI and ML in chemical engineering has become increasingly popular in recent years. AI and ML can help engineers optimize processes, reduce costs, and improve product quality. ML algorithms can analyse copious amounts of data and identify complex patterns that are difficult for humans to detect. This can lead to improved process control and more efficient use of resources.

The project is significant as it aims to develop a model that can accurately predict the htc based on various input parameters. This can lead to more efficient and effective chemical processes, ultimately reducing costs and improving product quality.

The use of AI and ML in chemical engineering is still relatively new, but it has enormous potential to transform the field. The development of accurate and efficient models for predicting the yield is just one example of how AI and ML can be used to improve processes and reduce costs. As more data becomes available, and algorithms become more advanced, the possibilities for using AI and ML in chemical engineering are endless.

1. **Methodology**

Data preprocessing is a crucial step in the machine learning (ML) pipeline. It involves transforming raw data into a format that can be easily understood and analysed by ML algorithms. The sklearn library in Python provides several modules for data pre-processing.

The pair plots and the correlation matrix were studied first and then the data were pre-processed.

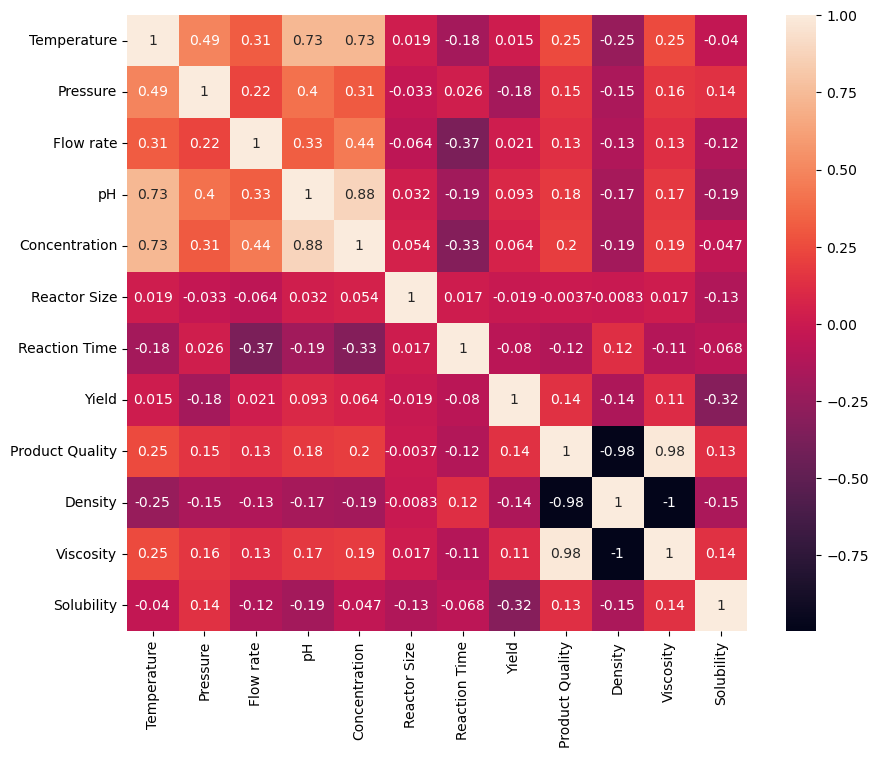


Figure : Correlation Matrix

Here are some of the commonly used preprocessing techniques in sklearn:

1. *Data Cleaning* : The first step in data preprocessing is to remove any unwanted data such as missing values, duplicate rows, or irrelevant columns. The `sklearn.impute` module provides various strategies to handle missing data, such as replacing missing values with the mean or median of the other values in that column.
2. *Feature Scaling* : Unique features in the dataset may have different scales. Feature scaling is the process of normalizing the data to a standard scale, usually between 0 and 1 or -1 and 1. This can be done using the `sklearn.preprocessing` module, which includes the `MinMaxScaler`, `StandardScaler`, and `RobustScaler` classes.
3. *Encoding Categorical Variables* : ML algorithms cannot work with categorical variables directly. Therefore, categorical variables need to be converted into numerical variables using techniques such as one-hot encoding or label encoding. The `sklearn.preprocessing` module includes the `OneHotEncoder` and `LabelEncoder` classes for this purpose.
4. *Data Transformation* : ML algorithms may require data to be transformed into different forms, such as reducing the dimensionality of the data or creating new features from the existing ones. The `sklearn.decomposition` module provides methods for dimensionality reduction, such as PCA, while the `sklearn.preprocessing` module provides methods for feature extraction, such as `PolynomialFeatures`.
5. *Splitting Data* : After pre-processing the data, it needs to be split into training and testing sets. The `sklearn.model\_selection` module provides various methods for splitting the data, such as `train\_test\_split` and `StratifiedShuffleSplit`.

Overall, data pre-processing is a crucial step in ML and can significantly affect the performance of the ML algorithm. Sklearn provides a wide range of tools and modules to pre-process data efficiently in Python.

One of the pre-processing module used for the models is the `*train\_test\_split*` module from the `sklearn.model\_selection` module in Python is used to split a dataset into training and testing sets for machine learning purposes. It randomly divides the dataset into two parts, one for training the model and the other for assessing its performance. The `train\_test\_split` module can split data into any percentage ratio, but the commonly used ratio is 80:20 or 70:30 for the training and testing datasets, respectively.

The `train\_test\_split` module provides several parameters to customize the split, including:

1. test\_size : The proportion of the dataset to include in the test split. This parameter accepts either a float representing the proportion of the dataset to include in the test split or an integer representing the number of samples to include in the test split.
2. train\_size : The proportion of the dataset to include in the training split. This parameter accepts either a float representing the proportion of the dataset to include in the training split or an integer representing the number of samples to include in the training split.
3. random\_state : The random seed to ensure the same random split occurs each time the code is executed.
4. shuffle : Whether to shuffle the data before splitting. By default, shuffle is set to True.
5. stratify : If the dataset has a class imbalance, the `stratify` parameter can be used to ensure that each class is represented proportionally in both the training and testing datasets.

Once the data is split using the `train\_test\_split` module in the ratio 80:20, the training dataset is used to train the machine learning model, and the testing dataset is used to evaluate its performance. This module is a fundamental tool for model selection and performance estimation, as it helps to avoid overfitting by checking the model's generalization ability on unseen data.



Figure 2: Splitting the data set into training and test set

The other pre-processing module used for the models is the*StandardScaler*. The StandardScaler module from the sklearn Python library is a data pre-processing technique that scales the features to have a mean of zero and a standard deviation of one. It is often used in ML to normalize the input data to improve model performance. The StandardScaler module works by first computing the mean and standard deviation of each feature in the training data. It then scales the features such that each feature has a mean of zero and a standard deviation of one. This is achieved by subtracting the mean from each feature and dividing by the standard deviation.

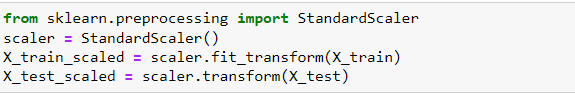


Figure 3: Feature Scaling using StandardScaler

**Multilinear Regression Model (MLR):**

Multiple Linear Regression is a statistical technique that is used to model the relationship between a dependent variable and multiple independent variables. In other words, it helps to find the best linear relationship between the dependent variable and multiple independent variables.

In Python, the sklearn.linear\_model module provides a class called LinearRegression to implement Multiple Linear Regression. This class uses the Ordinary Least Squares (OLS) method to estimate the coefficients of the independent variables. The OLS method calculates the values of the coefficients such that the sum of the squares of the differences between the observed values and the predicted values is minimized.

To implement Multiple Linear Regression using the LinearRegression class, the first step is to import the necessary libraries and load the data. Then, the independent variables and the dependent variable are separated. Next, the data is split into training and testing sets using the train\_test\_split module.

*Note: Here, the Feature Scaling (in this case, StandardScaler) was not used as the coefficients of this regression model will compensate for the values and in not applying the scaling method will not affect the performance metrics of the model.*

Finally, the model is trained using the fit() method of the LinearRegression class.

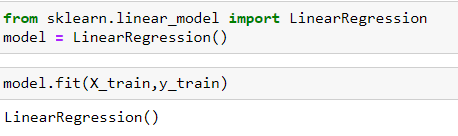


Figure 3: Training the Multilinear Regression model

The *hyperparameters* defined for this model are:

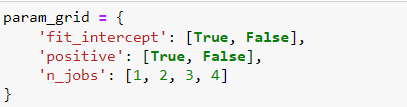


Figure 4: Hyper-parameters for the Multilinear Regression model

The *best* model is then predicted by GridSearchCV method.

The predicted results from the model are determined using the following code:

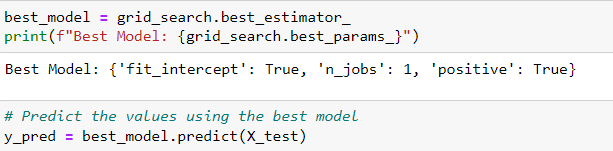


Figure 5: Code snippet for predicting the results

The performance of the model is evaluated using the following code :

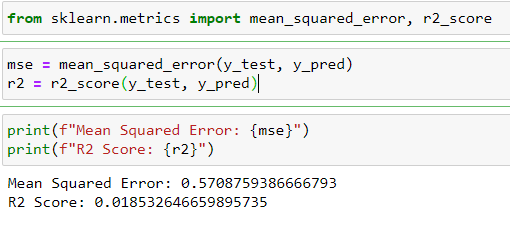


Figure 6: Model performance and evaluation

Finally, the true vs predicted values, residual plot and the histogram of the residuals were plotted and are shown below.

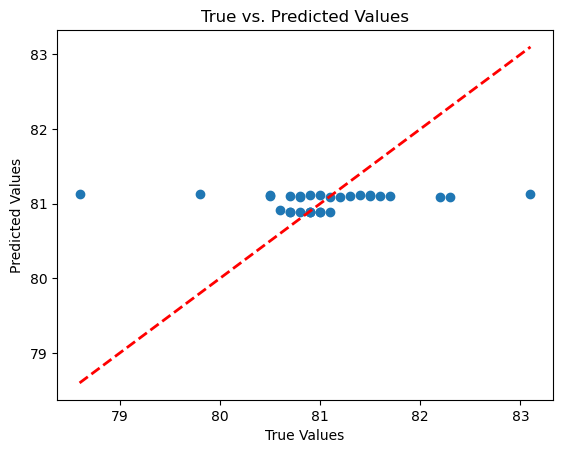
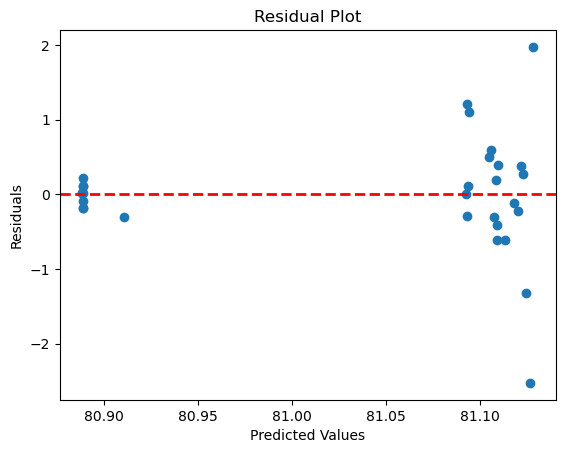


Figure 7: True vs Predicted values (MLR model)



The *performance metrics* of this model are as follows:



Figure 10: Performance Metrics of MLR model

**Support Vector Regression Model (SVR)**:

The next model developed is the *Support Vector Regression* model.Support Vector Regression (SVR) is a popular machine learning algorithm used for regression problems. In contrast to traditional linear regression models, which attempt to fit a line to the data, SVR models use support vectors to identify a nonlinear decision boundary that best separates the data into different regions.

The goal of SVR is to find a function f(x) that approximates the relationship between the input variables (x) and the output variable (y). The function is defined as a linear combination of kernel functions, which transform the input variables into a higher dimensional space where the data can be more easily separated. The SVR model then tries to find the optimal value of the weights (w) and bias (b) that minimize the error between the predicted values and the actual values of the output variable.

In the Support Vector Regression (SVR) model, hyperparameters are the parameters that are set before the training of the model and cannot be learned from the data. These parameters can significantly affect the performance of the model, and therefore, it is essential to tune them to achieve better results.

The main *hyperparameters* in the SVR model are:

* 1. Kernel: The kernel function specifies the type of non-linear transformation that is applied to the input data. The most used kernels in the SVR model are the linear, polynomial, and radial basis function (RBF) kernels.
  2. C: The C parameter controls the trade-off between the simplicity of the model and its ability to fit the training data. A smaller value of C leads to a simpler model, while a larger value of C leads to a more complex model that may overfit the data.
  3. Gamma: The gamma parameter controls the width of the Gaussian kernel in the RBF kernel function. A smaller value of gamma leads to a wider kernel and a smoother decision boundary, while a larger value of gamma leads to a narrower kernel and a more complex decision boundary.

To optimize these hyperparameters, techniques such as grid search, random search, and Bayesian optimization can be used. These techniques involve trying different combinations of hyperparameter values and evaluating the performance of the model on a validation set to find the best hyperparameter values that lead to the best performance. For this model, the hyperparameters were optimised using the GridSearchCV technique.

In sklearn, the SVR model is implemented in the SVR class.

*Note: Here, the Feature Scaling technique (StandardScaler) was used to scale the feature.*

The model was then trained with the training dataset obtained from the train\_test\_split module.

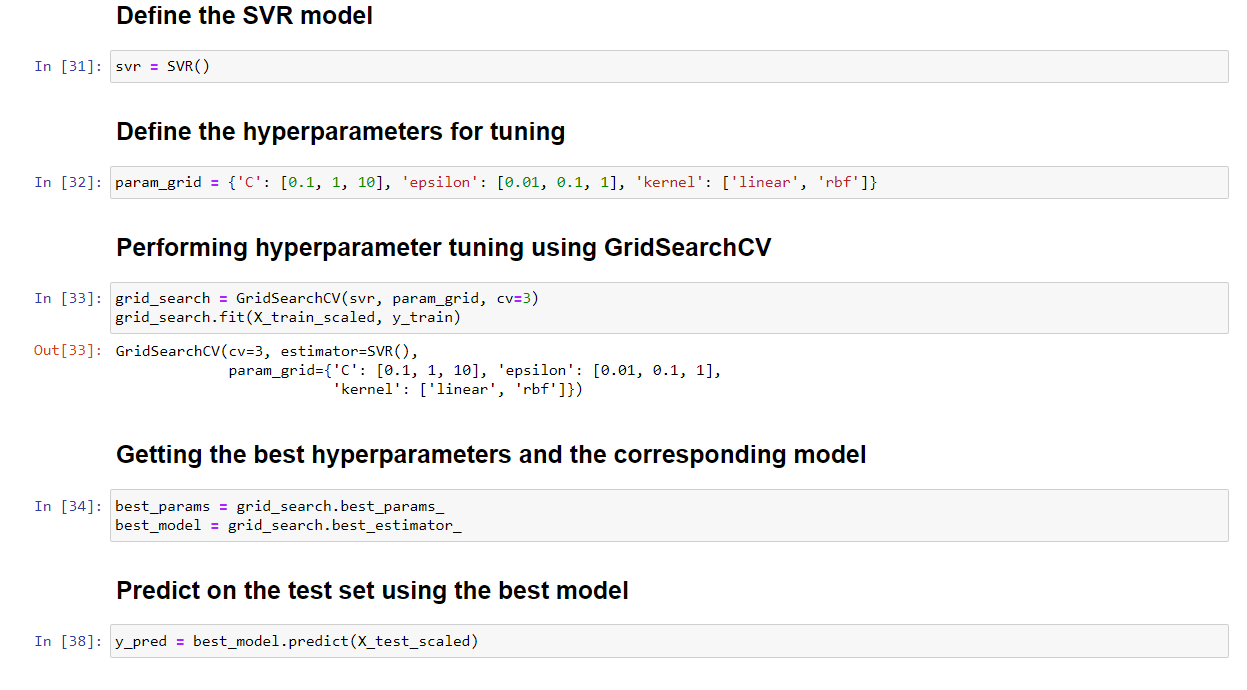


Figure 12: Training the SVR model using the hyperparameters

The true vs predicted values, residual plot and the histogram of the residuals are shown below.

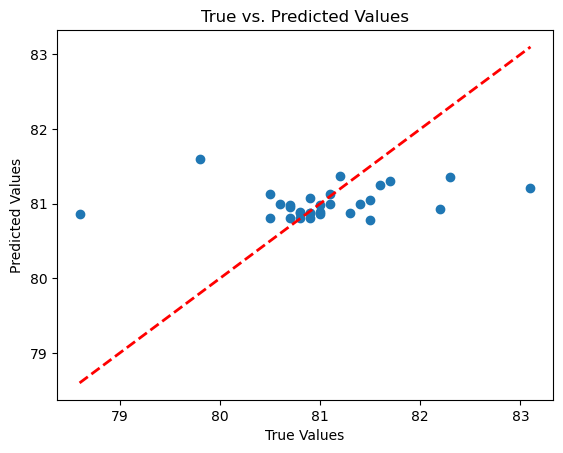
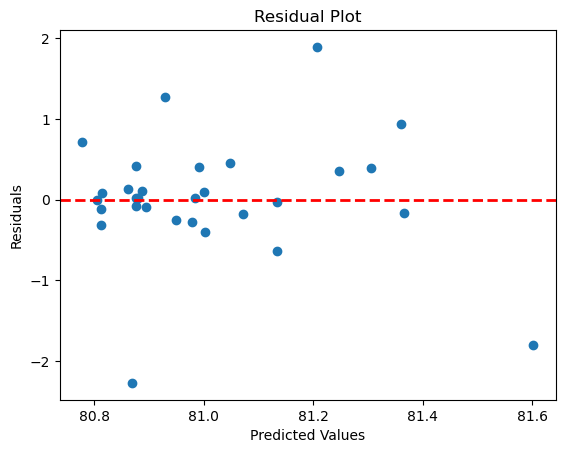


Figure 13: True vs Predicted Values (SVR model)



The performance metrics of the SVR model are:

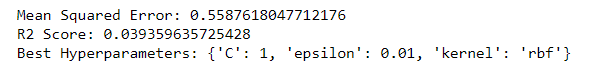


Figure 16: Performance metrics of the SVR model

**Decision Tree Model (DT)**:

The 3rd model developed is the decision tree model. Decision Tree Regression is a popular machine learning algorithm used for regression problems. The algorithm creates a decision tree based on the training data, which is then used to make predictions for new data.

In Decision Tree Regression, the algorithm divides the feature space into rectangular regions or partitions, with each partition representing a region in which the target variable has a constant value. The algorithm selects the best feature and split point to divide the data into two subsets that minimize the residual sum of squares.

To make a prediction for a new data point, the algorithm traverses the decision tree by evaluating the feature values of the data point at each node and following the appropriate branch based on the decision criteria. Once the algorithm reaches a leaf node, it outputs the mean value of the target variable in that leaf node as the prediction for the new data point.

In sklearn, the Decision Tree Regression model is implemented in the DecisionTreeRegressor class.

In decision tree regression models, some of the important *hyperparameters* include:

1. max\_depth: This parameter determines the maximum depth of the decision tree. If this value is set too high, the model may overfit the training data, while if it is set too low, the model may underfit and have poor predictive power.
2. min\_samples\_split: This parameter sets the minimum number of samples required to split an internal node. Setting this value too high may lead to underfitting, while setting it too low may lead to overfitting.
3. min\_samples\_leaf: This parameter sets the minimum number of samples required to be at a leaf node. Setting this value too high may lead to underfitting, while setting it too low may lead to overfitting.
4. max\_leaf\_nodes: This parameter sets the maximum number of leaf nodes that the decision tree can have. Setting this value too low may lead to underfitting, while setting it too high may lead to overfitting.
5. criterion: This parameter determines the function that is used to measure the quality of a split. The two most common criteria are “mse” (mean squared error) and “mae” (mean absolute error).
6. splitter: This parameter determines the strategy used to split at each node. The two most common strategies are “best” (choose the best split) and “random” (choose the best random split).

To tune these hyperparameters, we can use techniques such as grid search or randomized search. These techniques involve trying out different combinations of hyperparameters and selecting the combination that gives the best performance on a validation set.

The model was trained using the scaled training dataset.

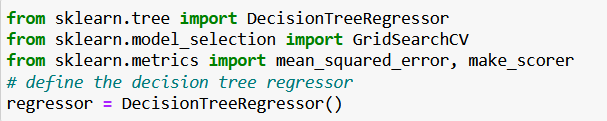


Figure : Defining the Decision Tree regressor

Figure 18: Best hyperparameters for the Decision Tree model

The true vs predicted values, residuals plot and the histogram of the residuals were plotted and are shown below:

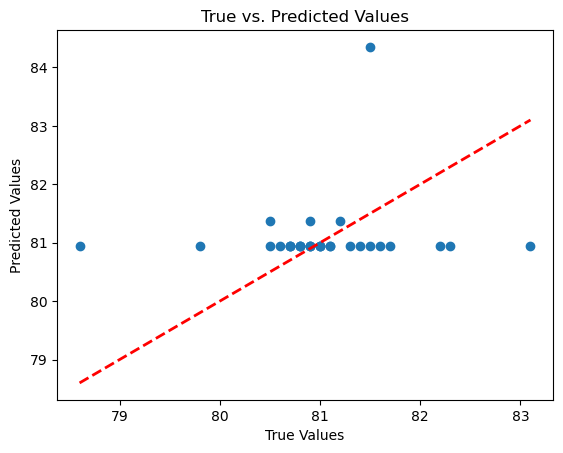
****

Figure 19: True vs Predicted values (DT model)

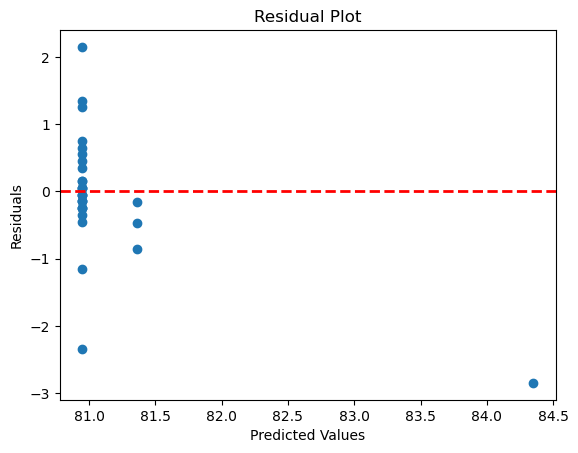


Figure 20: Residuals Plot (DT Model)

The performance metrics of the DT model are as follows:

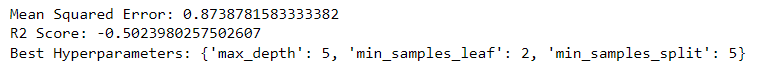


Figure : Performance metrics of DT model

**Random Forest Model**:

Random Forest Regression is a variant of the Random Forest algorithm used for regression problems. It uses an ensemble of decision trees to predict the continuous numerical target variable.

The Random Forest Regression model in sklearn works similarly to the Random Forest Classification model. The main difference is that the target variable is continuous rather than categorical. The algorithm creates many decision trees using different subsets of the training data and features. During training, the algorithm selects a random subset of features and data points for each tree, reducing the risk of overfitting and increasing the model's generalization ability.

To make a prediction with the Random Forest Regression model, the algorithm averages the predictions of all the decision trees in the forest. The final prediction is therefore a more stable and accurate estimate of the target variable than any individual decision tree.

*Hyperparameters* in Random Forest Regression Model:

* 1. n\_estimators: It determines the number of trees in the random forest. A larger number of trees can improve the performance of the model, but also increase computation time.
  2. max\_depth: It determines the maximum depth of the tree. A deeper tree can lead to overfitting, whereas a shallow tree may not capture the complexity of the data.
  3. max\_features: It determines the maximum number of features that can be used to split a node. A smaller number of features can reduce overfitting, whereas a larger number can increase model performance.
  4. min\_samples\_split: It determines the minimum number of samples required to split an internal node. A larger value can prevent overfitting but a smaller value can capture more information.
  5. min\_samples\_leaf: It determines the minimum number of samples required to be at a leaf node. A larger value can prevent overfitting but a smaller value can capture more information.
  6. bootstrap: It determines whether to use bootstrapping when constructing trees. If True, each tree is trained on a bootstrapped sample of the data, which can improve model performance.
  7. random\_state: It determines the random seed used by the random number generator. It can be used to ensure that the results are reproducible.

Defining the Random Forest regressor:

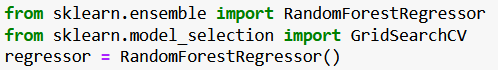


Figure 23: Defining the Random Forest regressor

The best hyperparameters (n\_estimators, max\_depth, min\_samples\_split and min\_samples\_leaf) are tuned according to the following code:

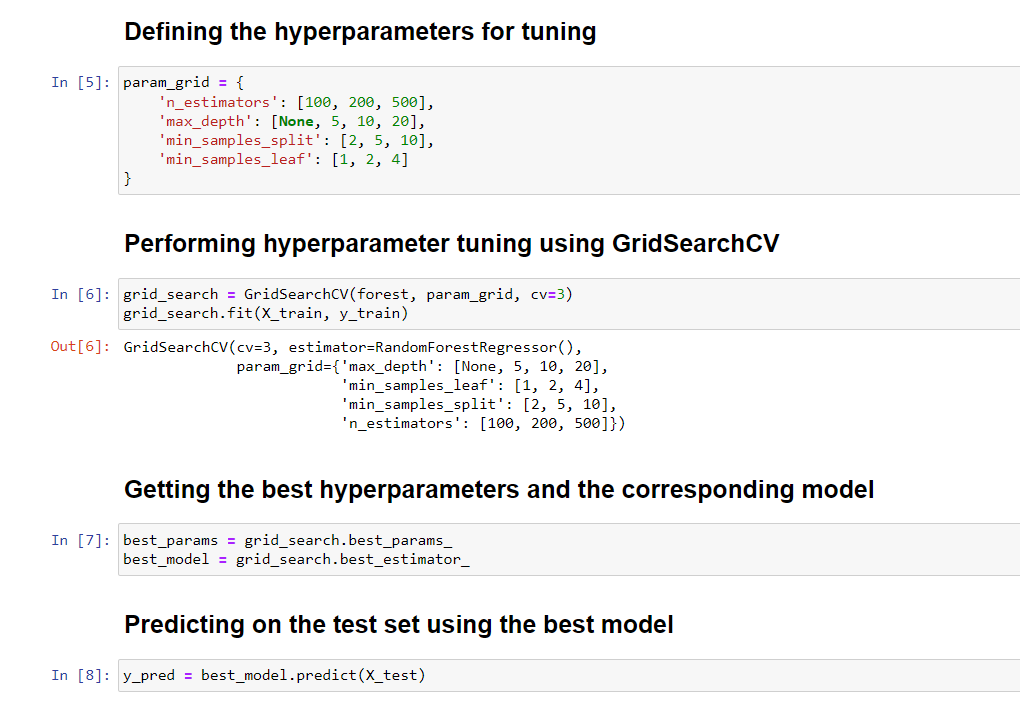


Figure : Best hyperparameters for the Random Forest model

The true vs predicted values, residuals plot and the histogram of the residuals were plotted and are shown below:

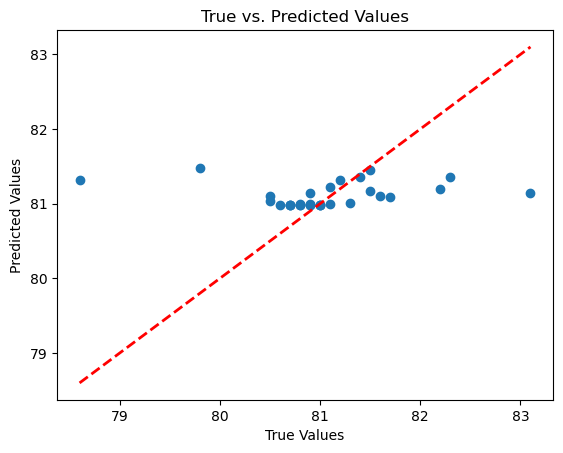


Figure 25: True vs Predicted value for Random Forest model

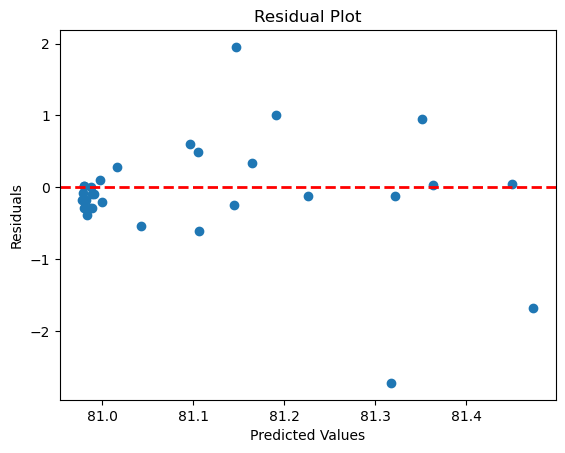


Figure 26: Residuals plot for Random Forest model

The performance metrics of the Random Forest model are given below:

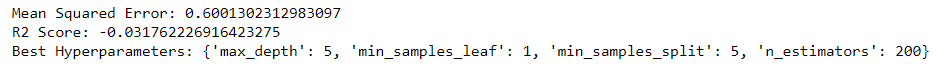


Figure 28: Performance metrics of Random Forest model

The Scoring Metrics of each models are given below:

|  |  |  |
| --- | --- | --- |
| **Model** | Mean Square Error (*MSE*) | R2 score |
| Multilinear Regression Model | 0.570876 | 0.0185326 |
| SVR Model | 0.558761 | 0.0393596 |
| Decision Model | 0.873878 | -0.5023980 |
| Random Forest Model | 0.600130 | -0.0317622 |

Thus, out of the 4 models developed the *Random Forest Model* works the best.

1. **Conclusion**

Based on the given metrics, it can be concluded that the **Support Vector Regression Model** performed the *best* in predicting the heat transfer coefficient. The model has significantly lower Mean Square Error (MSE) values and higher R2 scores compared to other models.

The Decision Model has the lower MSE value but negative R2 score among all models.

The Random Forest Model also performed well, with a lower MSE value but much lower R2 score than the Decision Model. Also R2 score is negative.

There are *several limitations* to the project that can be addressed in future work to improve the results:

1. *Data quality*: The accuracy and reliability of the model is heavily dependent on the quality of the data used to train and validate it. Future work could involve collecting more accurate and comprehensive data on the relevant variables and parameters.

2. *Limited variables*: The current model is limited to a few variables, and there may be other important variables that have not been considered. Future work could involve incorporating additional variables that may affect the heat transfer process.

3. *Model assumptions*: The models used in this project assume a linear relationship between the dependent and independent variables, which may not always hold true. Future work could involve exploring nonlinear models or other machine learning techniques to improve the accuracy of the model.

4. *Generalization*: The models developed in this project may not be applicable to all types of heat transfer problems. Future work could involve developing models for specific applications or situations.

1. *Computational efficiency*: Some of the models used in this project, such as the random forest model, may be computationally expensive and time-consuming to train.

*Future work* could involve developing more efficient algorithms or using parallel computing to speed up the training process.

In summary, there are several limitations to the project, but there is also potential for future work to address these limitations and improve the accuracy and applicability of the models.

1. **References**

The references used for developing the models are listed below:

1. Scikit-learn documentation: https://scikit-learn.org/stable/documentation.html

2. Introduction to Machine Learning with Python by Andreas C. Müller and Sarah Guido

3. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems by Aurélien Géron

4. Machine Learning Mastery by Jason Brownlee

5. Python Machine Learning by Sebastian Raschka and Vahid Mirjalili