Prediction Polymer Flow Rate through Machine Learning

Md Zaffar Zesshan

234107202

Submission Date: April 25, 2024

Final Project submission

Course Name : Applications of Al and ML in chemical engineering

Course Code: CL653

Contents

1 Executive Summary

This project focuses on addressing the critical challenge of modeling and controlling polymer melt flow rate (MFR) in industrial processes within the realm of Chemical Engineering. MFR, which denotes the flowability of a polymer melt, significantly impacts the quality and efficiency of production processes in industries such as extrusion, injection molding, and film blowing. Inconsistent MFR can lead to variations in product quality and increased manufacturing costs.

The project aims to develop robust mathematical models to predict MFR under varying processing conditions and implement effective control strategies to maintain MFR within desired ranges. By investigating the factors influencing MFR, including temperature, pressure, polymer characteristics, and processing parameters, the project seeks to enhance understanding and provide insights into the dynamics of MFR in polymer processing.

2 Introduction

2.1 Background

Polymer melt flow rate (MFR) plays a pivotal role in the realm of Chemical Engineering, particularly within the domain of polymer processing industries. MFR, defined as the measure of a polymer melt's flowability, profoundly influences the manufacturing process and the quality of the end products. Across various industrial applications such as extrusion, injection molding, and film blowing, precise control of MFR is imperative to ensure consistent product properties and optimize production efficiency. Consequently, understanding the factors affecting MFR and developing effective control strategies has been a subject of significant interest and research in Chemical Engineering.

2.2 Problem Statement

The specific problem addressed by this project revolves around the need to accurately model and control polymer melt flow rate in industrial processes. Variations in MFR can lead to inconsistencies in product quality and production efficiency, resulting in increased manufacturing costs and potential product defects. The challenge lies in developing robust

mathematical models that can predict MFR under different processing conditions and in implementing effective control strategies to maintain MFR within desired ranges. By addressing this problem, the project aims to enhance the understanding of MFR dynamics and contribute to the development of advanced control techniques for polymer processing industries.

2.3 Objectives

The main objectives of the project include:

1. Investigating the factors influencing polymer melt flow rate, including temperature, pressure, polymer characteristics, and processing conditions.

2. Developing mathematical models to accurately predict MFR based on input parameters.

3. Evaluating the performance of existing control strategies for MFR regulation and identifying areas for improvement.

4. Designing and implementing advanced control techniques, such as model predictive control or adaptive control, to maintain MFR within target ranges.

5. Validating the effectiveness of the developed models and control strategies through simulations and experimental studies.

6. Providing insights and recommendations for optimizing MFR control in industrial polymer processing applications.

3 Methodology

3.1 Data Source

The dataset has been taken from **kaggle**. Link to the dataset is given below.
Link. The data is taken as an excel file for jupyter notebook for further progress of the project.
Characteristics of the dataset:

i. **Volume**: Data has 9 columns and 2561 rows in total (2561*9 = 23049 data points)
ii. **Variety**: Data present is mainly numerical except one column which has Time and date.

iii. **Velocity**: Since the data is not fetched from any API or real-time, there is no velocity for the data.

Description of the Data

The data file contains tags from process instruments. The specific meaning of the tags is unknown but the follow table is an educated guess based on the name and range of values:

LABEL	DATA FILE TAG	DESCRIPTION
Time		Timestamp of the measurements
C3=	513FC31103.pv	Propylene (C3=) Feed Rate (kg/hr)
H2R	513HC31114-5.mv	Hydrogen to C3= Ratio
Pressure	513PC31201.pv	Reactor Pressure (bar)
Level	513LC31202.pv	Reactor Bed Level (m)
C2=	513FC31409.pv	Ethylene (C2=) Flow (kg/hr)
Cat	513FC31114-5.pv	Catalyst Feed Rate (kg/hr)
Temp	513TC31220.pv	Reactor Temperature
MFR	MFR	Melt Flow Rate (gm/10min)

3.2 Data Preprocessing

Data pre-processing 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.

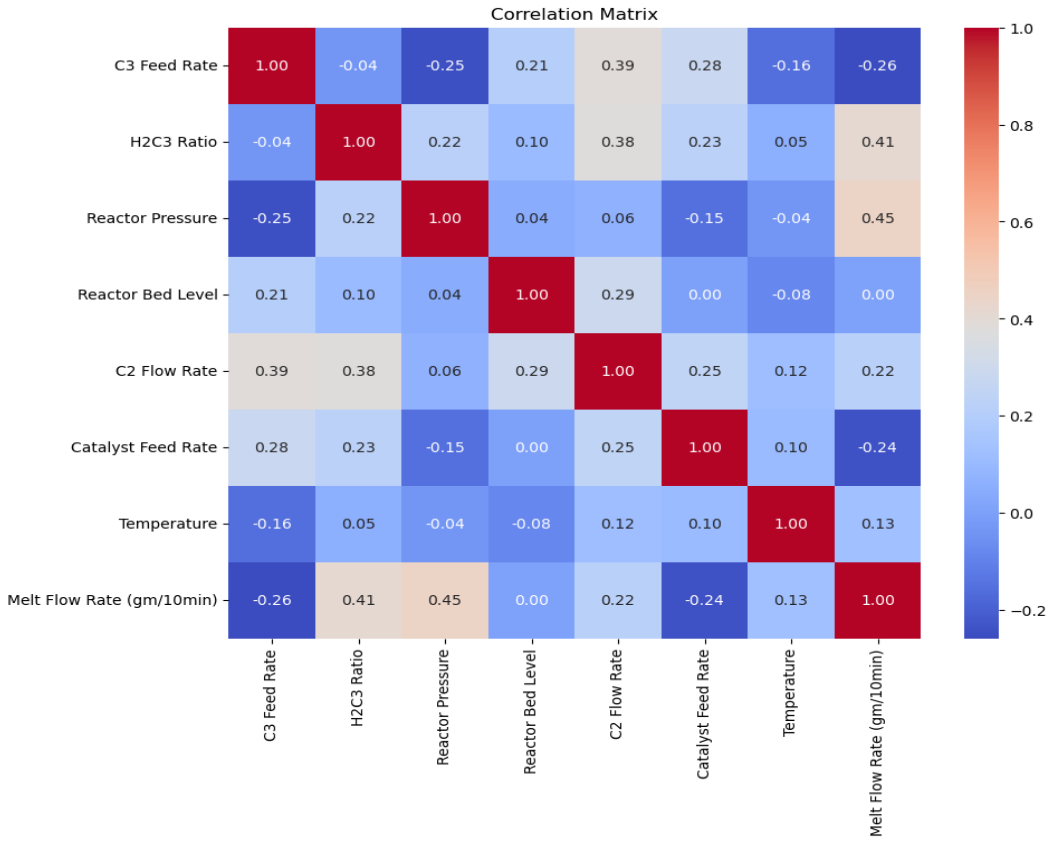


Figure 1: HeatMap of Data

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

a) *Data Cleaning* : The first step in data pre-processing 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. Here in this project, the first step in data cleaning is changing the column names to actual feature names. Removing the null values as it is only a small fraction of the whole dataset. And also removing the unnecessary columns.

b) *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.
Feature Scaling for this dataset is very important as it contains features in different ranges. To make model fitting more robust and, it is necessary to scale the features in the same range.

c) *Encoding Categorical Variables* : ML algorithms cannot work with categorical variables directly. Therefore, categorical variables need to be converted into numerical ones using one-hot or label encoding techniques. The `sklearn. preprocessing` module includes the `OneHotEncoder` and `LabelEncoder` classes for this purpose. Since the dataset does not contain any categorical column or feature, this step has not been implemented.

d) *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`. This step also has not been implemented in this current dataset.

e) *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.

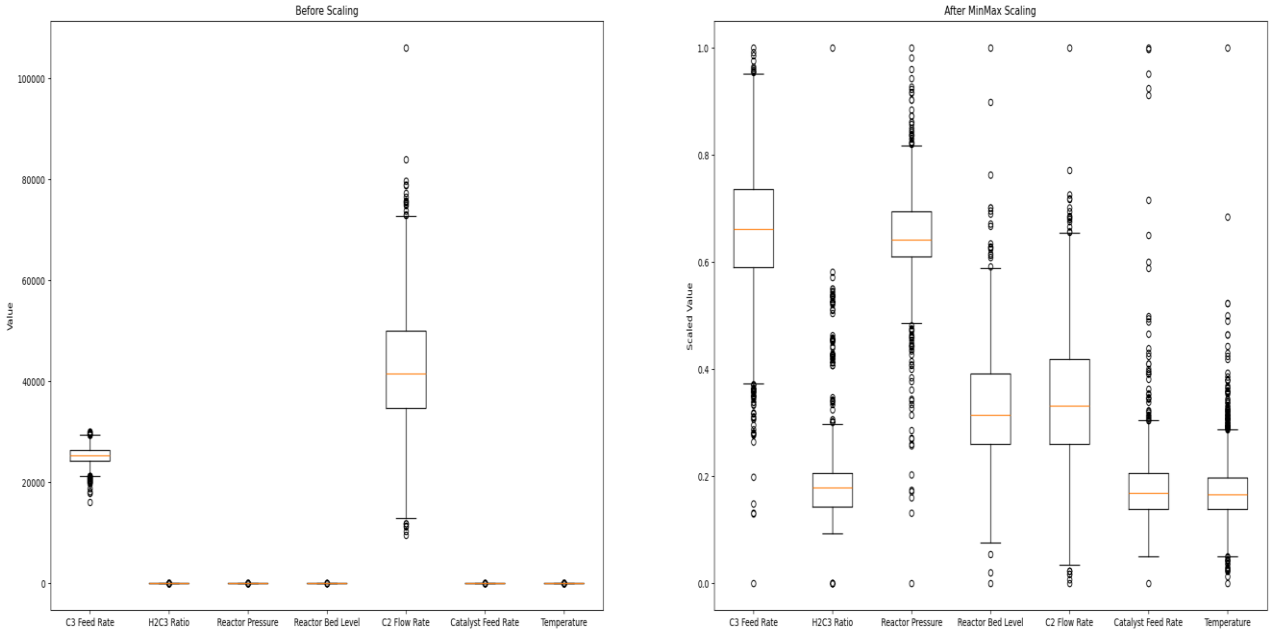


Figure 2: Comparison before and after scaling

One of the pre-processing module used for the models is the `*train_test_split*` module from `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:

i) 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.
ii) 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.
iii) random_state : The random seed to ensure the same random split occurs each time the code is executed.
iv) shuffle: Whether to shuffle the data before splitting. By default, shuffle is set to True.
v) 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.

3.3 Model Architecture

Before Training lets define the different models and their architecture to be used in this project

3.3.1 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.

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

```
model.fit(X_train,y_train)
```

```
LinearRegression()
```

Figure 3: Code snippet for Linear Regression

3.3.2 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.

```
[7]: model_svm = SVR()
     model_svm.fit(X_train,y_train)
     # Predict on test set
     y_pred_svm = model_svm.predict(X_test)
```

Figure 4: Code snippet for SVR

Hyperparameter tuning

[11]:
```
svr = SVR()

# Define parameter grid for grid search
param_grid = {
    'kernel': ['linear', 'rbf','poly'],
    'C': [0.1, 1, 10, 100],
    'gamma': ['scale', 'auto']
}
```

[12]:
```
grid_search = GridSearchCV(estimator=svr, param_grid=param_grid,refit=True)
```

[13]:
```
# Fit grid search
grid_search.fit(X_train, y_train)

# Get best model
best_model = grid_search.best_estimator_
```

[14]:
```
print(grid_search.best_params_)
print(grid_search.best_estimator_)
```

```
{'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
SVR(C=10)
```

Figure 5: Code Snippet for Hyperparameter tuning of SVR

3.3.3 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 predict 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.

```
[12]: modeldt = DecisionTreeRegressor()
      modeldt.fit(X_train, y_train)

      y_pred_dt = modeldt.predict(X_test)
```

Figure 6: Code Snippet for DT

With hyperparameter tuning

```
[17]: param_grid = {
          'max_depth': [3, 5, 7, None],
          'min_samples_split': [2, 5, 10],
          'min_samples_leaf': [1, 2, 4]
      }

[18]: dthyp = DecisionTreeRegressor(random_state=42)

[19]: grid_search = GridSearchCV(dthyp, param_grid, cv=5, scoring='neg_mean_squared_error')

[20]: grid_search.fit(X_train, y_train)

[20]: GridSearchCV
      estimator: DecisionTreeRegressor
      DecisionTreeRegressor

[21]: best_params = grid_search.best_params_
      print("Best Parameters:", best_params)

      Best Parameters: {'max_depth': 7, 'min_samples_leaf': 4, 'min_samples_split': 10}

[22]: best_regressor = DecisionTreeRegressor(**best_params)
      best_regressor.fit(X_train, y_train)

[22]: DecisionTreeRegressor
      DecisionTreeRegressor(max_depth=7, min_samples_leaf=4, min_samples_split=10)
```

Figure 7: Code Snippet for hyperparameter tuning of DT

3.3.4 Random Forest Model (RF):

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.

```
[6]: modelrf =  RandomForestRegressor(n_estimators=100, random_state=42)
     modelrf.fit(X_train, y_train)

     y_pred_rf = modelrf.predict(X_test)
```

Figure 8: Code Snippet for RF

With Hyperparameter Tuning

```
[10]: param_grid = {
          'n_estimators': [50, 100, 150],
          'max_depth': [None, 5, 10, 20],
          'min_samples_split': [2, 5, 10],
          'min_samples_leaf': [1, 2, 4]
      }
      rfhyp = RandomForestRegressor(random_state=42)
```

```
[11]: grid_search = GridSearchCV(rfhyp, param_grid, cv=5, scoring='neg_mean_squared_error')
      grid_search.fit(X_train, y_train)
```

```
[11]: GridSearchCV
      estimator: RandomForestRegressor
          RandomForestRegressor
```

```
[12]: best_params = grid_search.best_params_
      print("Best Parameters:", best_params)
```

```
Best Parameters: {'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 150}
```

```
[13]: best_regressor = RandomForestRegressor(**best_params, random_state=42)
      best_regressor.fit(X_train, y_train)
```

```
[13]: RandomForestRegressor
      RandomForestRegressor(max_depth=20, min_samples_leaf=2, n_estimators=150,
                            random_state=42)
```

Figure 9: Code snippet for hyperparameter tuning of RF

3.4 Tools and Technologies

Language Used : Python

Algorithm	Python Package	Package Version	Website
Support Vector Machine	sklearn.svm.SVR	1.4.2	https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html
Decision Tree Regressor	sklearn.tree	1.4.2	https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html
Random Forest Regressor	sklearn.ensemble	1.4.2	https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

4 Implementation Plan

4.1 Development Phases

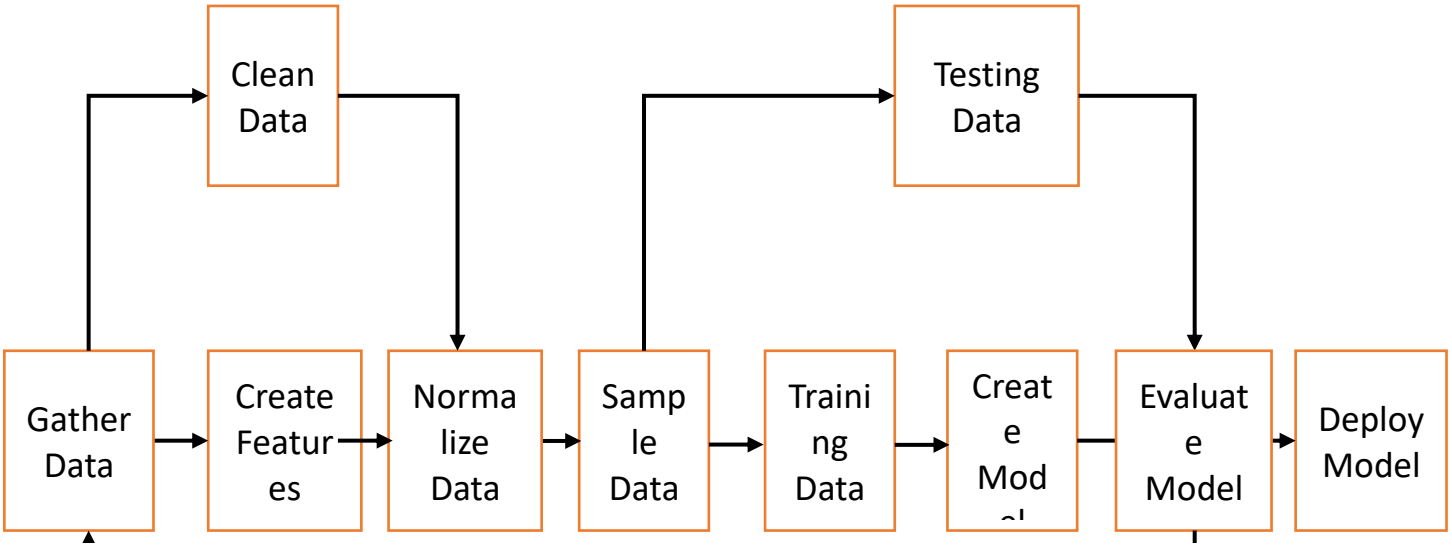


Figure 10: Flowchart of development phases

4.2 Model Training: Strategies for training the model

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.

4.3 Model Evaluation: Metrics and methods

The metrics used are:

i. **R2 Score**: The R2 score, also known as the coefficient of determination, ranges from 0 to 1, with higher values indicating better model performance. It measures the proportion of the variance in the dependent variable that is explained by the independent variables. A score of 1 indicates that the model perfectly predicts the dependent variable, while a score of 0 suggests that the model does not explain any of the variability. R2 score serves as a crucial evaluation tool in regression analysis, aiding in the interpretation of model effectiveness and predictive power.

ii. **RMSE**: Root Mean Squared Error (RMSE) is a widely used metric to assess the accuracy of a predictive model, particularly in regression analysis. It calculates the square root of the average squared differences between predicted and actual values. RMSE provides a meaningful measure of the model's prediction error in the same units as the target variable, making it easily interpretable. Lower RMSE values indicate better predictive performance, reflecting a smaller discrepancy between predicted and observed values.

5 Testing and Deployment

5.1 Testing Strategy

The model so developed can be tested for their accuracy metrics like the model with high R2 score and low RMSE can be selected and then pickled. This model can further be loaded and thus can be tested with other remote datasets.

5.2 Deployment Strategy

A separate UI can b prepared using some library such as streamlit and thus User input or someone from domain can be told to test the model with datasets and thus in real-world scenario can be tested.

I have tried to pickle the model and to make an UI using streamlit, Unfortunately, it didn't work.

5.3 Ethical Considerations

Deploying any model, including those encapsulated in pickled form, carries significant ethical implications that warrant careful consideration. Firstly, there's the concern of bias within the model itself. If the training data used to develop the model is biased or unrepresentative, the deployed model can perpetuate and potentially exacerbate existing societal inequalities. This bias can lead to unfair treatment or discrimination against certain individuals or groups, particularly in sensitive domains such as finance, healthcare, and criminal justice.

6 Results and Discussion

6.1 Findings

I have tried to fit four regression models:

i) Linear Regression
ii) Support Vector Machine
iii) Decision Tree
iv) Random Forest

Out of these four models, Random Forest 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.

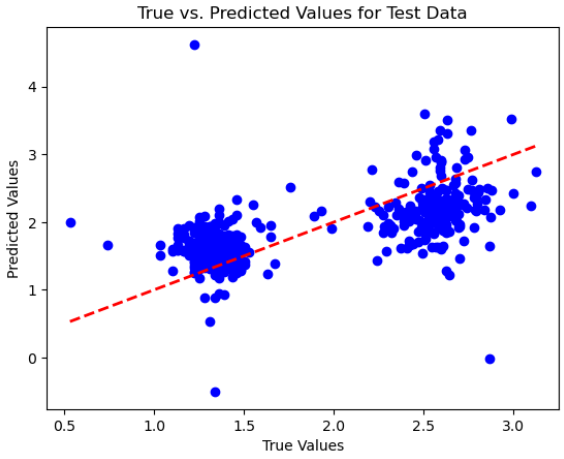
The data has nonlinearity . Also it have clusters. Some unsupervised learning would have given better results.

6.2 Comparative Analysis

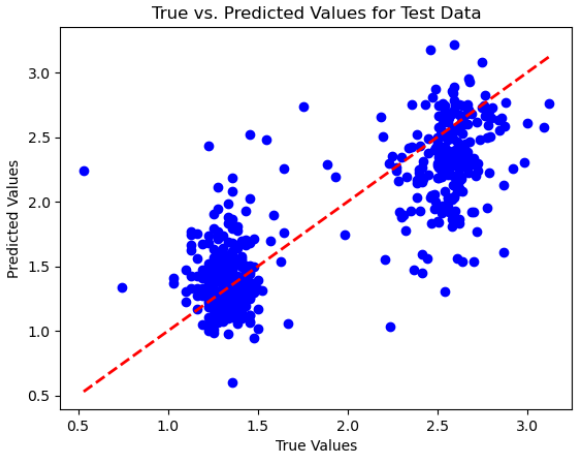
The models are compared using their metrics:

R2 Score and RMSE

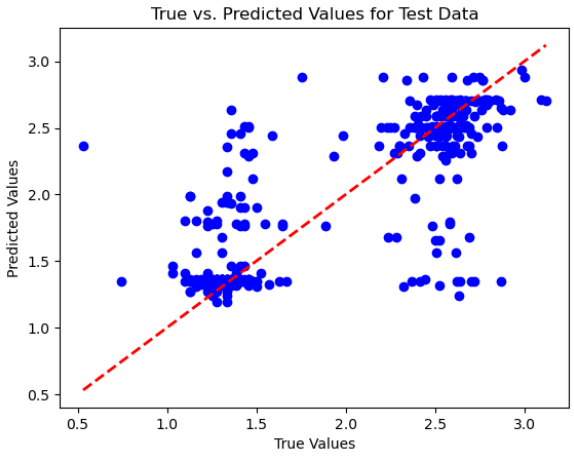
Model Name	R2 Score	RMSE Score
Multiple Linear Regression	0.365	0.503
Support Vector Regression	0.677	0.358
Decision Tree Regressor	0.708	0.341
Random Forest Regressor	0.792	0.385



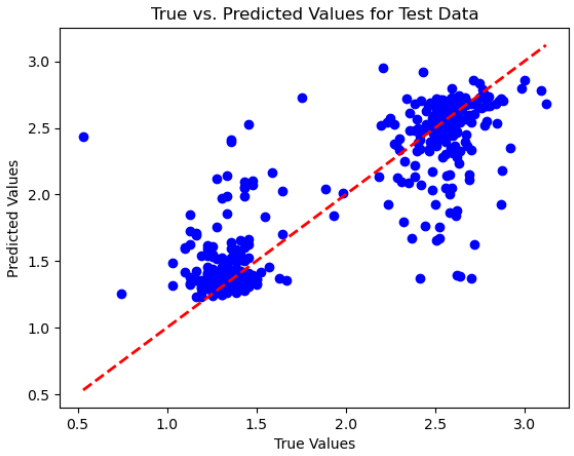
a) Scatter Plot of Linear Regression



b) Scatter Plot of SVM



c) Scatter Plot of Decision Tree



d) Scatter Plot of Random Forest

6.3 Challenges and Limitations

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.

5.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.

7 Conclusion and Future Work

This project aimed to develop predictive models for Polymer Melt Flow Rate (MFR) based on reactor conditions, leveraging regression methods and data analysis techniques. Through thorough exploration, preprocessing, and modeling, we have successfully created models capable of predicting ln(MFR) accurately. These models serve as virtual "soft sensors," supplementing infrequent lab sampling and enhancing real-time monitoring and control in polymer production processes.

The impact of this project is significant in several aspects. Firstly, it enhances process control and optimization by enabling continuous monitoring and adjustment of polymer viscosity to meet specific grade requirements. This not only improves product quality but also reduces manufacturing costs by minimizing the need for manual intervention. Secondly, it demonstrates the effectiveness of data-driven approaches in process engineering, showcasing the potential of digital twins for real-time monitoring and control.

Future work

Model Refinement: Continuously refining the regression models can enhance prediction accuracy. Exploring different feature engineering techniques, optimizing model hyperparameters, or experimenting with advanced regression algorithms can lead to improved performance

8 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

6. Roychadhury, A. (n.d.). Predict MFR of Polymer. Kaggle. Retrieved January 22, 2021.

Please add the below mentioned links.

Web link: (if deployed as live website give website link)

Data Source: https://www.kaggle.com/datasets/apek1999/predict-mpv-of-polymer/code

Python file: Attached as zip along with this document.