Exploring Iterative Applications of Machine Learning on Pyrolysis of Plastics

A Major Qualifying Project Report:

Submitted to the faculty of

Worcester Polytechnic Institute

In partial fulfillment of the requirements

For the degree of Bachelor of Science

In Chemical Engineering

By

Owen Ferrara

Eric Himebaugh

Matthew Rando

Christopher Skangos

Date: May 6th, 2021

Project Advisor: Michael Timko

Table of Contents

Acknowledgements

The team would like to thank our advisor Professor Michael Timko for his consistent guidance throughout the course of this project, as well as the WPI Chemical Engineering Department. We also thank Feng Cheng for assisting the team in accommodating to using Python and Scikit-learn. Our team would also like to thank Elizabeth Belden for her help and feedback on developing models and presentations. We also would like to thank Professor Randy Paffenroth, Wenjing Li, and Rasika Karkare from the WPI Data Science Department for sharing their insight and experience with machine learning algorithms with us.

Abstract

The world estimate for plastic pollution is expected to rise above 300 million tons annually. This prompts the need for alternative chemical recycling solutions, such as pyrolysis. Pyrolysis, high temperature, high pressure reactions, could prove to be a sustainable method to recycle plastics and produce fuel oil. Collecting waste plastic to convert it to fuel via pyrolysis could help significantly reduce the number of waste plastics, though accurately predicting the oil yield remains a challenge. One way to predict the outcomes of these reactions is through machine learning. In this work, 310 datapoints were collected of plastic pyrolysis data already existing in the literature to create models that accurately predict the oil yield of a reaction based on the reaction conditions. These models were created using Scikit-learn's random forest regression and classification methods. Due to the modest size of the compiled literature data set, emphasis was placed upon incrementally improving the methods over iterations by variable selection, constraining input variables and the output oil yield, and by selectively removing error-prone, outlier data. From the models' results, it was concluded that machine learning methods could provide a viable way to predict pyrolysis oil yields.

Introduction

Plastic's increased usage throughout the world, especially within the industry, has increased global pollution. As the world estimate for plastic pollution is expected to rise, there is a need for more sustainable methods of its disposal. One promising sustainable option for disposing and recycling plastics is through pyrolysis, a chemical process that decomposes plastics at high temperatures in the absence of oxygen. Pyrolysis can effectively degrade plastics and poses as a viable method of recycling to be used for future plastic materials or the production of oil.

Having more sustainable methods for disposing of plastic would decrease the accumulation of such pollutants. Finding these alternatives will create a more circular process that reduces excess waste products and promises a cleaner environment. However, one roadblock preventing the widespread use of pyrolysis is its chemical complexity. Kinetic modeling of reactions is difficult and time-consuming, especially due to the presence of multiple sub-reactions and vast amounts of feed compositions. An understanding of such kinetic reaction models requires extensive development and knowledge of theory.

Emerging technology in data science and machine learning can expedite the optimization process. The application of machine learning allows for reaction data to be analyzed to discover patterns and model processes faster than experimentally producing kinetic models. Machine learning algorithms are typically trained on a large collection of data and then tested on new data to independently determine their accuracy and validity. The objective of this research was to explore and effectively use machine learning methods to predict pyrolysis oil yields. From this novel approach, machine learning shows promise for future success and optimization.

Background

Pyrolysis

Currently, global plastic production is estimated at 300 million tons per year and is expected to rise.[1] Since plastics are petrochemical hydrocarbons that include other additives that make them difficult to biodegrade, the acceleration of plastic production paired with its inability to decompose poses a problem for the future. Fortunately, sustainable methods of plastic disposal are being developed to recycle millions of tons of accumulated plastic waste. In the past decade, chemical recycling processes through pyrolysis have been explored to decompose polymer plastics into monomers of a char or oils.[1] These monomers of the plastic are then repolymerized to form into other products including aromatic compounds, alternative fuels, or raw polymers that could be reformed into another generation of plastic products.

While pyrolysis fuels are high in energy, the process is energy-intensive.[2] Currently, it takes 1.047 MJ to convert 1 kg of polyethylene (PE) plastic into liquid oil. Moreover, further energy is then required to refine this oil into usable products which in turn releases similar levels of emissions to conventional fossil fuels.[1] The large energy demand for the operation of these reactions on an industrial scale, paired with the irregular quality of plastic feedstock available creates setbacks towards pyrolysis' economic feasibility.[3] However, there is potential for greater success for future pyrolysis refineries that utilize other renewable processes such as solar and

hydroelectric to assist in producing oils. Through integrating renewable technologies, the high energy required to operate pyrolysis reactions could be mitigated and reduce the overall emissions of the process. Thus, these refineries would maximize economic and environmental benefits with minimal waste production.[1]

Another method toward building the feasibility of industrial pyrolysis processes is through the utilization of catalysts. Catalysts facilitate and speed up pyrolysis reactions ultimately making these processes more efficient. It has been recorded that catalytic pyrolysis carries greater potential to convert plastic to liquid oil, and through the lowered operating temperature and reaction time, increases the oil's quality.[1] Also, the addition of catalysts alters the liquid product's physical properties such as viscosity and decreases the oil's density, flash point, boiling range, and high heating value (HHV). Some examples of common catalysts utilized in pyrolysis are Iron (III) oxide, Calcium hydroxide, fluid catalytic cracking (FCC), natural zeolite, or synthetic zeolite.[1] Through a given catalyst, its BET surface area, pore size, pore-volume, and acidity will affect its impact on the reaction. Catalyst acidity has been researched to be the most influential in pyrolysis through its ability to remove impurities from liquid oil and promote more catalytic cracking.[1] Moreover, this acidity increases the gasoline range of hydrocarbons within the produced liquid oil. Moreover, innovations into nano-catalysts are in development that could ensure reliable yield and performance of these reactions.[1]

Understanding the kinetics of pyrolysis reactions is essential to further optimize the process. The primary modeling utilized for pyrolysis is radical chain kinetics, which focuses on the thermal degradation of the reactants.[4] The decomposition of polymers is complex as there are multiple intermediary steps and reactions.[5] In one study, to completely model the gas products of polyethylene, the model contained equations for: the chain initiation to form two primary radicals, the chain initiation to form one primary radical and one allyl type radical, termination, the primary radical abstracting, the secondary radical abstracting, β-scission, and had to account for the H-abstraction which is illustrated in Figure 1 below.[4]

$$BBP_n = +KPR12\sum k=5\alpha_k \sum j=n-1-k-1\ 3\infty P_j + 12\sum k=5\alpha_k 2c_{all} O_{n+k+4} + \sum j=n-1-k-1\ 5\infty O_j$$
$$+\alpha_1 2\sum k=1-1\ 2\infty \sum j=k+3\infty P_j - 1\ \alpha_1 \sum j=1+4\infty P_j + \sum j=1+3\infty P_j - 1\ \alpha_1 2\sum j=21\infty P_j$$
$$+12\alpha_1 2\sum k=1+2\infty \sum j=k-1\ 5\infty O_j - 1\ c^*_{all} \sum j=1-1\ 6\infty O_j + \alpha_1 2c^*_{all} O_{21+1} - 1\ \sum j=21-1\ 2\infty O_j$$
$$+12\alpha_1 c^*_{all} O_{1+5} + \sum j=1+6\infty O_j + \alpha_1 c^*_{all} O_{1+4} + \sum j=1+5\infty O_j$$
$$+12\alpha_1 2\sum k=1+4\infty$$
$$\sum j=k+3\infty O_j + \alpha_1 c_{all} c_{all} + 1\sum j=1+6\infty O_j + \alpha_1 \sum j=1+5\infty O_j + \alpha_1 \sum j=1+4\infty O_j$$
$$+\alpha_1 2\sum k=1+4\infty$$
$$\sum j=k+5\infty D_j + c_{all} \sum j=1+8\infty D_j + \alpha_1 1c_{all} + 1c^*_{all} D_{1+7} + c^*_{all} \sum j=1+8\infty D_j$$
$$+\alpha_1 c^*_{all} D_{1+6} - 1\ \sum j=1+7\infty D_j + c^*_{all} D_{1+5} + \sum j=1+6\infty D_j$$

$$BBO_n = +KPR12\sum k\neq 1\alpha_k \sum j=1+k\infty P_j + 12\sum k\neq 1\alpha_k 2c^*_{all} O_{1+k+4} + \sum j=1+k+2\infty O_j - 1\ \sum j=1-1-k-1\ 3\infty O_j$$
$$+\sum k\neq 1a_k 2c^*_{all} D_{1+k+4} + \sum j=1+k+5\infty D_j + a_1 2\sum k=1+2\infty \sum j=k+3\infty P_j + \sum j=21\infty P_j$$
$$+12\alpha_1 2\sum k=1+2\infty \sum j=k+5\infty O_j + c^*_{all} \sum j=1+6\infty O_j + \alpha_1 2c^*_{all} O_{21+1} - 1\ \sum j=21-1\ 2\infty O_j$$
$$+12\alpha_1 2\sum k=1+4\infty \sum j=k+3\infty O_j + \alpha_1 c_{all} c_{all} + 1\sum j=1+6\infty O_j + \alpha_1 2\sum j=21+3\infty O_j + \alpha_1 2c^*_{all} D_{21+4} - 1\ \sum j=21-1\ 5\infty D_j$$
$$+\alpha_1 2\sum k=1+4\infty \sum j=k-1\ 5\infty D_j + c^*_{all} \sum j=1+8\infty D_j + \alpha_1 c_{all} c_{all} + 1c^*_{all} D_{1+7} - 1\ c^*_{all} \sum j=1-1\ 8\infty D_j$$

$$BBD_n = +KPR14\sum 1\alpha_1 \sum j=n+1\infty O_j + 12\sum 1\alpha_1 c^*_{all} D_{n+1+1} + \sum j=n+1-1\ 2\infty D_j$$

Figure 1: Thermal kinetics responsible for H-abstraction sub reactions within pyrolysis.

While the complexity of these models is evident, research on the utilization of pyrolysis kinetics has been conducted especially for determining the efficiency of operating conditions and feedstock purity. In 2011, a study by Khaghanikavkani and Farid observed the activation energies and pre-exponential factor for the pyrolysis of polyethylene plastic when operating conditions and feedstock composition were varied.[6] Under isothermal and non-isothermal conditions within a semi-batch reactor, two different kinetic operating parameters and the activation energies of the reactions were determined. The non-isothermal trial recorded the reactor temperature, the respective activation energies, and pre-exponential factor values. The results revealed a correlation between the carbon numbers and the activation energies, while also revealing that major discrepancies may have been caused to unpredicted heat transfers from the reactor.

Another study measured the kinetics of the thermal degradation of polyethylene and polystyrene.[5] The kinetic model focused on: the initiation reactions to form the first radicals, the propagation reactions of intermediate radicals, β-scission of radicals to form unsaturated molecules and smaller radicals, the Alkyl radical isomerization via (1,4) and (1,5) H-transfer, the H-abstraction reaction (H-metathesis) on the polymer chain, and the termination reactions. By testing different mixtures of polyethylene and polystyrene feedstocks, and running at various temperatures, more accurate kinetics on how long these polymer residues last at a given temperature were determined. While time and resource-intensive, understanding the thermal kinetics of polyethylene pyrolysis could be utilized to increase the accuracy of predictive modeling.

One of the many roadblocks that are hindering the implementation of industrial pyrolysis processes is the complex reaction kinetics. While kinetic models for pure-fed plastic streams prove to be difficult to model with kinetics initially, especially considering Figure 1 above, introducing

a mixed plastic stream drastically increases the complexity and intricacy of modeling the process's kinetics. While kinetic models for polymers have been developed, due to the variance in feed components when applied to real-world scenarios, the model's accuracy is drastically altered.[4] In addition, the wide variety of factors of pyrolysis make comparison of kinetics difficult between differently designed reactions.

While experiments to optimize these reactions are time-consuming and expensive, kinetic modeling research has offered the opportunity to utilize machine learning to model these reactions and avoid the complexities inherent in the application of pyrolysis to an industrial, real-world process. Through this analysis, the operating conditions that yield the best products can be determined and cut through the traditional complexities of kinetic modeling. Already, preliminary experiments have been conducted around the validity of machine learning models as opposed to laboratory experimentation.

Machine Learning

Currently, the machine learning literature in the pyrolysis space is sparse. One paper compared applying the machine learning methods of decision trees and neural networks toward reducing the computational time of detailed kinetic models of biomass pyrolysis.[7] From the research performed by Hough et al., two different types of multi-layer feed-forward networks looked at the efficiencies and performance from each of the learning pathways. One of these networks worked as a "full net" by using a single neural net and predicting 30 outcomes.[7] The other utilized 30 independent nets that all predicted single results. In comparison to a kinetic ODE model developed by the same group, the ODE took 4.7 seconds of code execution time compared to 1.1×10^4 s for decision trees and 1.7×10^4 s for the neural nets. While the neural network model proved effective, decision trees were ultimately favored picked over them for faster training times and simplicity. From the results, impressive accuracy in predicting the kinetic models for lignin pyrolysis was discovered.

Further literature has focused on developing predictive models for biomass yields.[8] The success of these models provides hope that while novel, machine learning research can continue and be built upon. With machine learning processes, data is split into training and test sets where the model interprets the training set to predict the test set. One of the more popular machine learning techniques is called Random Forest (RF). RF utilizes decision trees to parse the dataset for patterns. RF is popular within research due to its ability to work with small data sets, understand higher dimensional relationships, and has been found to be more accurate than linear regression methods for pyrolysis modeling.[8] Therefore, the RF algorithm has become one of the primary methods for early research on pyrolysis machine learning. With RF, either regression or classification can be performed. Regression predicts an exact number, for example, an oil yield from 0 to 100%. Classification predicts categorically, for example predicting either a 0 or 1 representing two groups above and below a certain oil yield. The current literature has solely reported random forest regression results.

As part of establishing a working algorithm, the dataset must be pre-processed. One method of pre-processing is imputing missing data from a dataset. K Nearest Neighbors Imputation (KNN) is a popular type of imputer used to estimate ranges of data. The imputer looks at data that is similar to the point that it is trying to predict, or its nearest neighbors, and forms an educated guess as to what the missing data could be.[9] These missing sections or parts of data are a common problem in machine learning and data science. KNN is dubbed a lazy learner because it waits until classification is instructed to it rather than having prior knowledge about the data set. Other imputation methods such as using the mean, median, and mode of data are also common practice.[10]

Previous biomass studies such as one done by Tang et al. observed success using RF algorithms when comparing it to multiple linear regression (MLR).[8] To determine the accuracy of each model, R^2 and the root mean square error (RMSE) metrics were used.[8] R^2 is a statistical measurement to see how much the dependent variable is affected by the change in the independent variable. RMSE determines how far away certain data is from the line of best fit by measuring the difference between predicted values and the actual values.[11] Below, Equations 1 and 2, show how R^2 and RMSE are calculated. The goal was to predict bio-oil yield and hydrogen content using datasets with 137 and 264 datapoints, respectively. For inputs, pyrolysis conditions such as the heating rate, particle size, and temperature were included. In addition, the paper included either the proximate or ultimate composition information as inputs. The proximate information was fixed carbon, ash, and volatiles. The ultimate compositions were the C-H-O-N compositions. For the MLR method, the R^2 was measured at 0.166 for the proximate yield, and 0.284 for the ultimate yield. Moreover, the RMSE's were measured at 7.45 and 7.96 respectively. When compared to RF, the accuracy massively improved, with an R^2 of 0.92 for the proximate yield and 0.87 for the ultimate yield. The RSME's measured 2.13 and 3.05 respectively. This report showcased the impressive accuracy that RF regression models could produce in correspondence to biomass reactions.

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(Y_i^{exp} - \widehat{Y}_i)^2}{\sum_{i=1}^{N}(Y_i^{exp} - Y_{ave}^{exp})^2} \quad (1)$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(Y_i^{exp} - \widehat{Y}_i)^2} \quad (2)$$

In another study, bio-char yield and the carbon contents of bio-char (C-char) based on the pyrolysis data of lignocellulosic biomass were modeled. In total, the research group used 245 datasets for bio-char and 128 datasets for C-char all comprised from previous publications.[12] Through a random forest (RF) algorithm, the study separated input data into four categories: Biomass Structural Components (Lignin, Cellulose Hemicellulose, (L-C-H)) and Ash, Element Compositions (C-H-O-N), the particle size of biomass, and pyrolysis conditions (Heating Rate, Highest Treatment Temp (HTT), residence time (RT)). Then, through 5-fold cross-validation following the training and testing of the model, three methods were run twice: One with all data

in except C-H-O-N, another with all inputs except L-C-H, and finally with all inputs.[12] Figure 2 showcases the resultant graphs from the respective trials. The idea to test different inputs helped build this study's methodology and approach to the developed pyrolysis of plastics dataset.

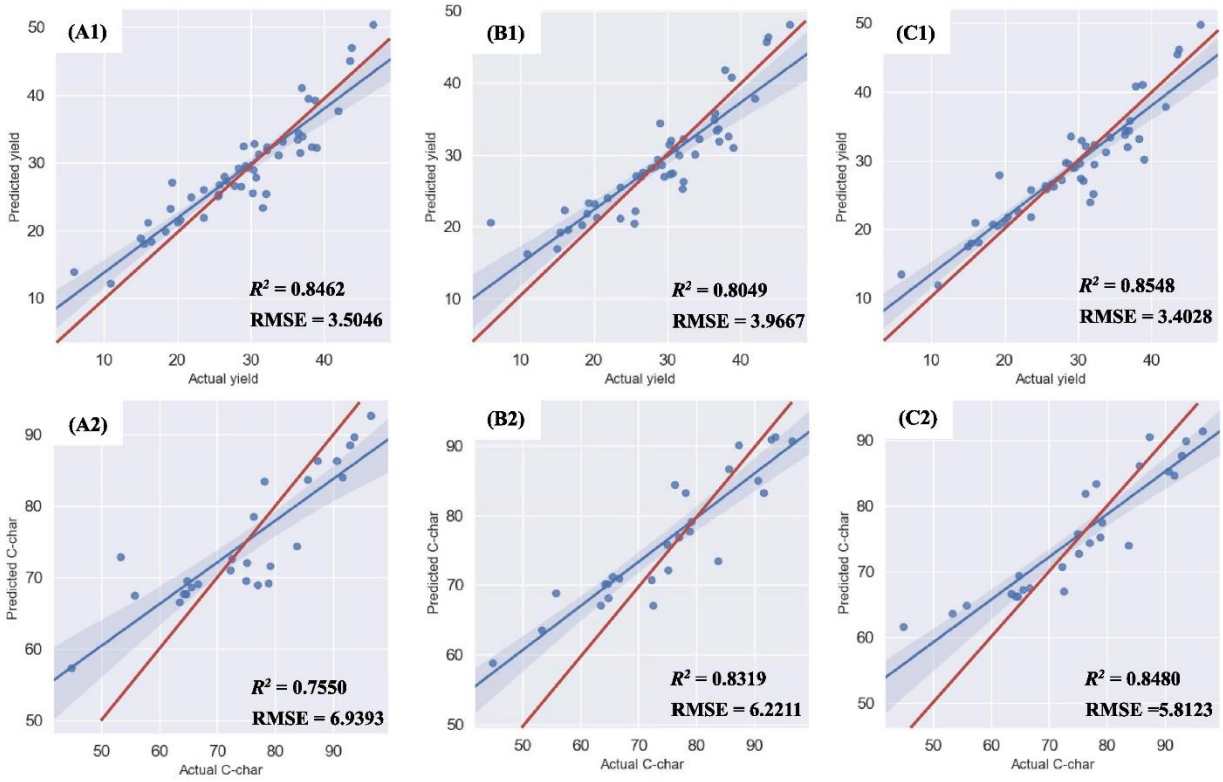

Figure 2: Comparison of predicted bio-char yield/C-char and actual values using test data with different inputs: (A) all inputs except C-H-O-N, (B) all inputs except L-C-H, (C) all inputs. The red lines refer to the line y = x (i.e. predicted values = true values).[12]

In the methods of Zhu et al.'s research, the bio-char yields prediction ranged from R^2 values 0.8 to 0.85 with RMSE between 3.4 to 4.0. Both the models with all inputs and all inputs but C-H-O-N performed the best. These results showcased the models in predicting the yield with a comfortable accuracy measurement. Furthermore, in the tests focused on predicting C-char, the model showcased its accuracy with R^2 values from 0.75 to 0.85, and RMSE's from 5.8 to 6.9.[12]

In one of the only published studies on pyrolysis of plastics with machine learning, a feed-forward neural network model utilized a small 24 point dataset of several mixed consumer plastics compositions (HDPE, LDPE, PP, and PS) to determine the oil yield of non-recycled plastics.[13] These four compositions were used as the only inputs for predicting yield rather than using pyrolysis conditions. However, the paper only included data between 400 and 500℃. The paper assumed that the plastic inputs were between 2 to 3 mm in particle size, and all had reaction times of 30 minutes. Through the model, a mean squared error (MSE) of 0.11 was computed. When comparing the experimental results from the dataset to the predicted curve, the MSE was found to be 2.64×10^{-4}. This reported value of MSE reported that the model fits almost perfectly to the

predicted yields. Reported R^2 values were above 0.9 in both the training and validation datasets only further validating their model.[13]

Through the success of the predictive power from such as small data set size, the power of optimization through machine learning was realized. The success of this paper and others inspired this team to attempt to test the random forest algorithms of plastics pyrolysis, which has shown success for biomass pyrolysis. To expand the size of the training and test sets from the Abnisa et al. paper, the collection of pyrolysis of plastics dataset was necessary.[13] Instead of testing a model on only plastic mixtures, all plastic feeds would be included along with reaction conditions in the literature search. Overall, the goal was to explore a synthesis of methods utilized by the limited literature as well as discover new information from the novel approach of applying random forest on a plastics dataset.

Methods

Primary Dataset Creation and Characteristics

The primary data set, which was used to create all the machine learning models, as well as to create subsets of data, was compiled from an analysis of relevant literature. Variables collected included the plastic composition percentages, being made up of some amount of high-density polyethylene (HDPE), low-density polyethylene (LDPE), polypropylene (PP), polystyrene (polyS), polyvinylchloride (PVC), and polyethylene terephthalate (PET). The other variables included the reaction conditions with which pyrolysis was carried out, as well as the quantifiable results of the process. These variables include reaction temperature (T) in Celsius, heating rate (HR) in Celsius per minute, particle size (PS) in millimeters, feed size (FS) in grams, residence/reaction time (RT) in minutes, nitrogen sweep flow rate (N_2 Rate) in nitrogen per minute, the presence of a catalyst (Cat), reactor type (R type), and included three different dependent variables being oil yield, gas yield, and char yield. The inclusion and search of these variables were inspired by the papers that utilized machine learning to predict biomass pyrolysis using feed characteristics and pyrolysis conditions. In addition, the utilization of the plastics compositions was inspired by the Abnisa et al. paper which achieved successful results from only 24 data points.[13] After a literature search, the primary dataset consisted of 310 unique data points, amassed from 37 different articles.

In some cases, estimation had to be used to determine the value for a specific variable when making the dataset. This was due to the source of that datapoint perhaps being conveyed through graphical means, given with a certain amount of uncertainty attached to it, or given as a range of values. The lattermost of these was typical for feed and particle sizes. For cases using uncertainty, these were ignored, and when ranges of values were given, an arithmetic average was taken for use in the primary dataset. Additionally, not every data source included each type of independent variable. For example, many studies did not include a nitrogen sweep gas, nor did some report a reaction time or residence time. This problem existed for numerous independent variables and is illustrated by Table 1. All the sources used to create the primary set reported feed composition, temperature, yield data, and no mention of a catalyst was taken to mean that no catalyst was present during the reaction. All 37 papers from which data were collected can be found and referenced in Appendix A. Catalyst characteristics such as the catalyst type, particle size, amount, and surface area were collected but not used for model information due to lack of consistency between the reporting of catalyst information. When being input into the model, catalysts were denoted with either a 0 or 1. Zero indicated that no catalyst was present in the reaction and one indicated that there was a catalyst. For reactor type, several scenarios arose for determining the appropriate category. To limit the variables for reactor type, five categories were decided on for the model: batch, fixed bed, fluidized bed, horizontal tube, and semi-batch. Literature papers either included one of these categories directly or provided a description that was used to sort the data points towards one of these five categories. For example, one paper by Palafox-Ludlow & Chase from 2001 used a modified microwave oven, which was sorted into the batch category for the model.[14] Reactor type was modeled similarly to catalyst. Each reactor type (batch, fixed bed, fluidized bed, horizontal tube, and semi-batch) was given its own column in an input excel csv file. These columns had a 1 for the reactor used for the reaction and a zero in place of all the other reactors.

This is a process called one-hot encoding which is common in machine learning.[15] The two datasets, one being the raw unedited version and the model ready dataset can be found in the supplemental materials submitted alongside this paper. Note that neither N_2 flow rate nor residence time data were used to make the models. This was due to low reporting with residence time appearing in roughly 22% of the papers and 54% reporting the N_2 flow rate.

Table 1: Percentage of papers that reported pyrolysis conditions. Note: 100% of papers included temperature and compositions.

	Heating Rate	Particle Size	Feed Size	Residence Time	Reaction Time	N2 Flow Rate
Data Points	238	217	207	69	177	126
% of Entries Reporting	84	70	78	22	65	54

Dataset Visualization

Throughout the 310 data point dataset, it was necessary to visualize the features within it to graphically understand the data that had been collected. Before running and choosing models, these investigations helped guide testing. Even though the dataset was small, there were many ways in which the dataset could be manipulated to create a vast number of permutations. Because of this, the dataset needed to be constrained in certain ways to focus on research. One of the primary differentiations between data points was whether the collected point was influenced by a catalyst or not. Overall, there were 99 catalyst datapoints and 211 non-catalyst data points as visualized in Figure 3 below.

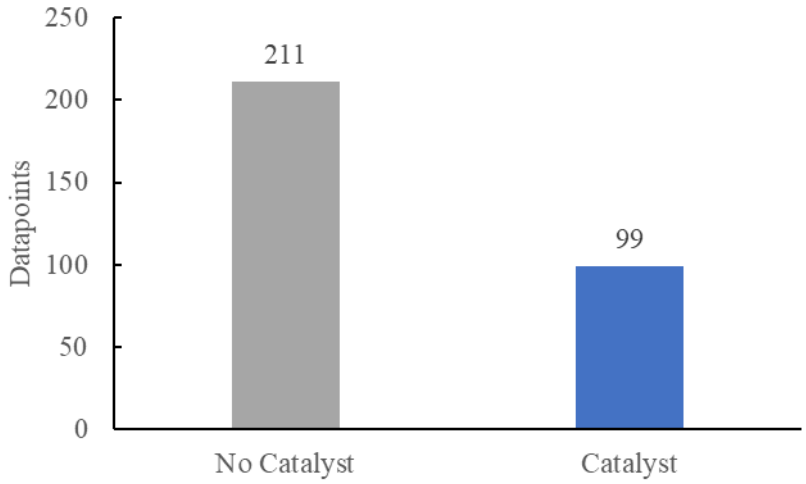

Figure 3: Datapoints with catalysts and without catalysts.

Another key part of the dataset was the composition of the plastic feed whether it was HDPE, LDPE, PP, PolyS, PVC, or PET. A chart indicating the distribution of pure feeds and mixed feeds with respect to temperature can be seen in Figure 4. Overall, there were 209 pure feeds of

plastics and 101 mixed feeds of plastics reported from the research articles. Ideally, more mixed feed data would be useful, especially for future applications to plastic wastes, however, the current literature was focused on pure feeds. Figure 4 displayed that mixed feeds were most prevalent within the 400 to 550 °C range, whereas pure feed data expanded uniquely into the higher temperature data points above 700 °C. For this reason, a temperature range of 400 to 550°C included a high combination of mixed and pure feeds. This was an important development because a primary goal was to develop a model that could handle mixed feeds.

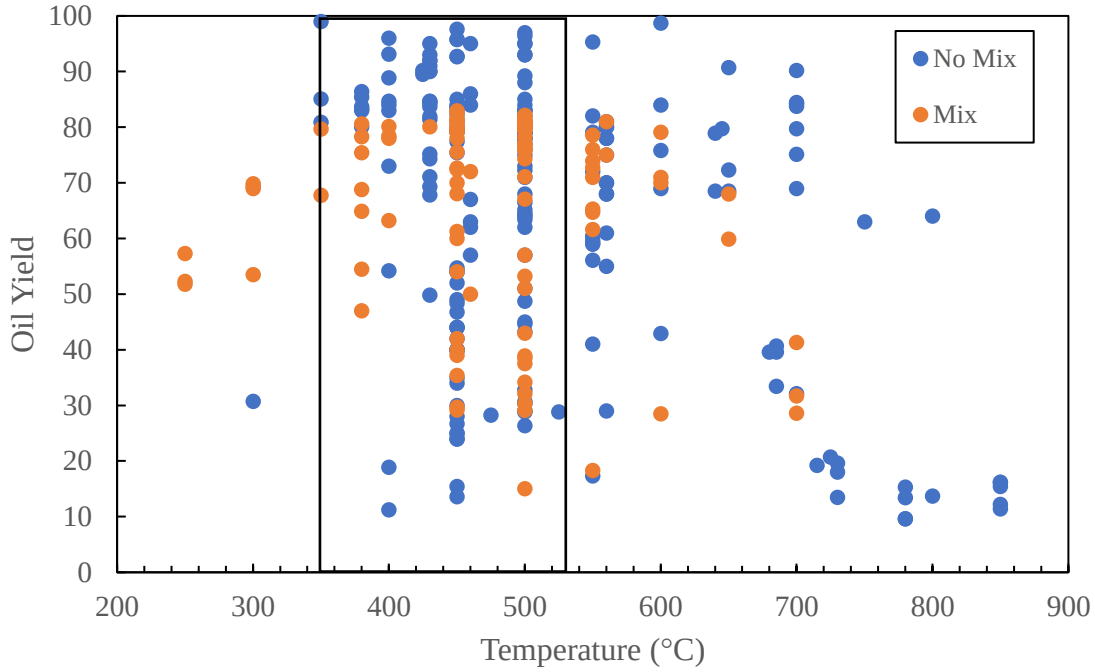

Figure 4: Distribution of pure feeds (Blue) and mixed feeds (Orange) across various temperatures.

Concerning the pure feeds themselves, HDPE, LDPE, and PP made up the majority with 65, 53, and 56 data points, respectively. There were 21 pure PolyS points, 13 pure PET points, and only one pure PVC point. The spread of composition percentages was also visualized using Figure 5 below. For the compositions of HDPE, LDPE, PP, and PolyS, there were similar spreads. LDPE, PP, and PolyS themselves appeared the most similar. PVC and PET data were sparser between the compositions of 40 and 90%.

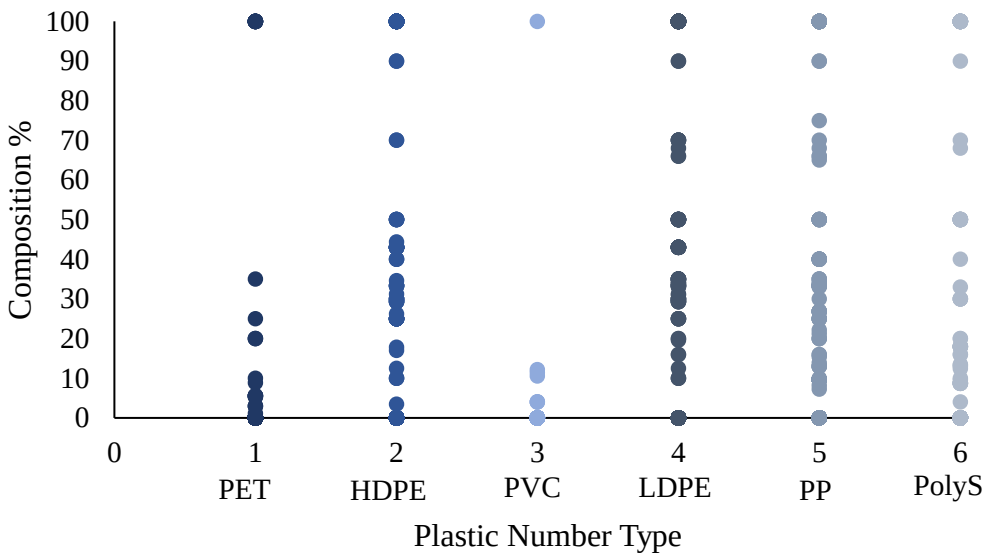

Figure 5: Plastic composition spreads for each plastic type organized by plastic number.

Similar to catalyst data, the reactor type was considered as an input variable and was recorded from literature data. Figure 6 displays the distribution of reactor types for all data points. Most of the data were from batch reactors, consisting of 200 total data points. This indicated that potentially a model with only batch data could be a good focus for an algorithm. The next most common were fixed bed and fluidized bed at 50 and 42 data points, respectively. Horizontal tube reactor data had 15 data points, all from one study from Quesada et al.[16] There were only three semi-batch points from one paper by Kumar & Singh.[17]

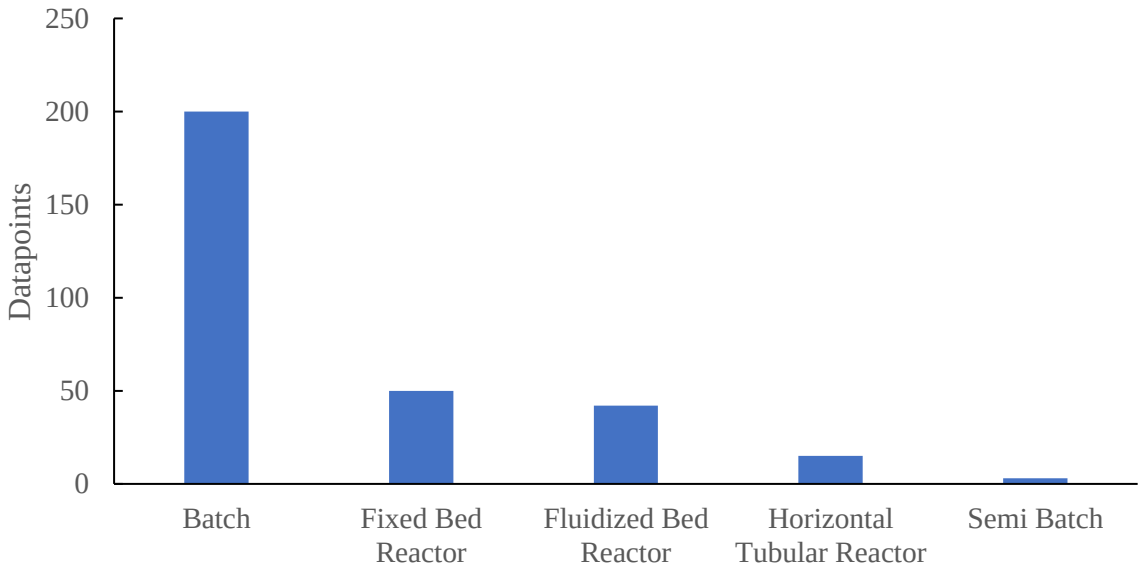

Figure 6: Reactor type dataset distribution for the five reactor types.

In addition to observing the features collected for the model, understanding the output of the model was also important. Plotted in Figure 7 was the number of datapoints for each oil yield.

To better visualize this, oil yields were rounded to the nearest 5. For example, 91.2 would become 90, and 78.7 would become 80. The most common oil yields were around 70 to 85%. When observing the split of the data, around 89 datapoints were below 50% oil yield and 221 were above 50% oil yield. The importance of this was that the model would have more information on data above 50% yield based on data points alone. Furthermore, it indicated that performing oil yield cutoff testing and classification of oil yields above or below a certain threshold could be useful and was worth testing.

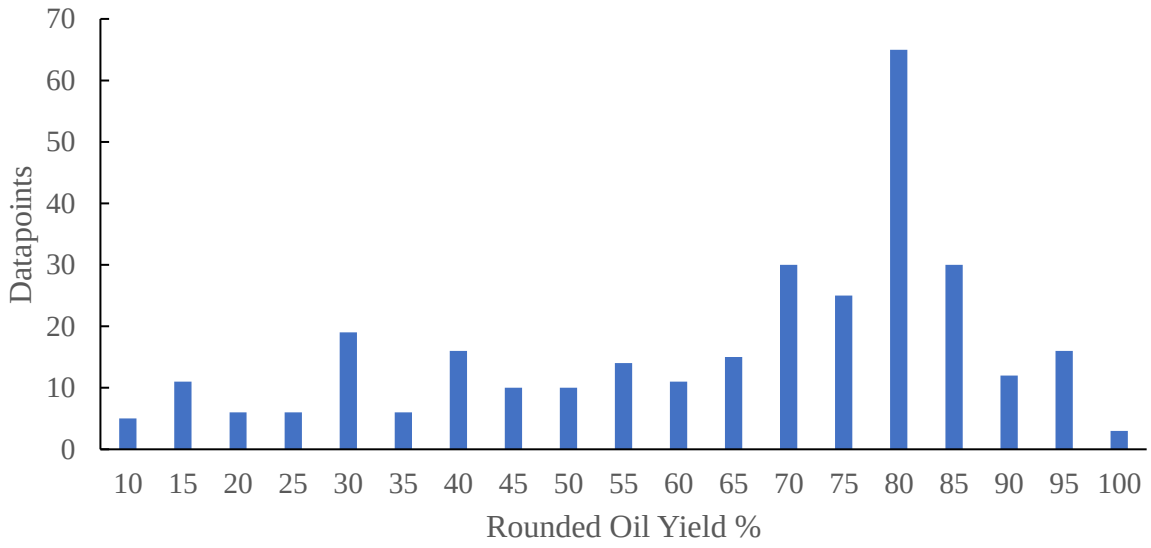

Figure 7: Oil yield distribution of datapoints. Note: Oil yield was rounded to the nearest 5 to make for a cleaner visual.

Machine Learning Models

From the dataset, various models were created using several different strategies to investigate how different portions of the dataset would react to the algorithm. For making models, missing data was a common problem. This was due to differences between reported pyrolysis conditions in the literature. There were two approaches towards this problem, omitting data with missing conditions or imputing the data. Several of the models included imputed data and others did not. These models were named Model A through Model I. A description of each is present in Table 2. Model A was based on the hypothesis that reaction time data was a critical feature in conjunction with temperature and composition to predict oil yields. For this reason, this model included every literature data point that reported a reaction time. The rest of the data for this model was imputed with the KNN method. Model B included every data point and was chosen to see how the model reacted to imputing all the missing data not reported by literature. Model C was developed based on a strategy from Abnisa et al. where a neural net model was used to predict oil yields. For this model, only composition data was used as the input features.[13] Furthermore, only compositions between 400 to 500 °C and 0 to 3.5mm particle size were included. Model D included all the data reported from batch reactors. This was considered because most of the data, around

two-thirds, were from batch processes. Model D, similarly, with A and B had missing data imputed with KNN. Models E and F were attempts at modeling the process without data imputation. Model E did not include reaction time as a feature. Whereas Model F did not include feed size nor reaction time. Reaction time was not included in either Model E or F to preserve as many data points as possible compared to the imputed model. If unimputed reaction time was included, then the Model E and F would be reduced to 82 and 101 datapoints, respectively. Models A through F were tested with the random forest regressor. In addition to these, three different models were tested with a random forest classification method. These were Model G, H, and I. Model G was Model B adapted for classification. Similarly, Model H and I were based on Model F and adapted for classification. Model I used a simple binary classification, either 0 or 1, and Model H used the grouping method where oil yield ranges were assigned numbers, either 1-4 or 1-7 depending upon the test being performed.

Table 2: Different models developed for regression and classification random forest algorithm with data points and descriptions

Model	Data Points	Description
A	182	Imputed, Unimputed Reaction Time Data
B	310	Imputed, All Data
C	199	Unimputed, Based on Abnisa et al. 2019[13]
D	200	Imputed, Batch Reactor Data Only
E	132	Unimputed, Feed Size included
F	171	Unimputed, Feed Size not included
G	310	Imputed, Based on Model G for classification
H	171	Unimputed, Based on Model F for classification, Grouping Method
I	171	Unimputed, Based on Model F for classification

Data Model Subset Creation

From the original 310-point data set, numerous subsets of models were created for different machine learning analyses. While there were many ways to segregate the data, there were three main categories of these subsets:

I. Removal of specific independent variables
II. Variable constraints
III. Oil yield constraints

From a chemical kinetics perspective, not every variable that was included in the primary dataset was as useful as the other. For example, the mass of the feed placed into a reactor surely does not affect the yield of the reaction as much as the temperature or the heating rate. For this reason, many trials were included that removed certain independent variables from contention within the machine learning algorithm. Another important reason to remove certain variables was for trials that did not include data imputation; any variable missing a very large amount of data would need to be removed, such as N_2 flow rate, as indicated by Table 1. In addition, variables would also be

added piecewise to the model where compositions would be tested alone, then add temperature, and more to observe the effect on the model's results.

Some variables were constrained, meaning that only values that appeared in a certain range would be taken and used in the machine learning analysis. For example, Figure 8 displayed that most of the temperature data exist between 400°C and 600°C, with some outlier data points. With the hypothesis that removing certain outlying datapoints would increase the accuracy of resulting models, some variables for specific trials were constrained within a range of values.

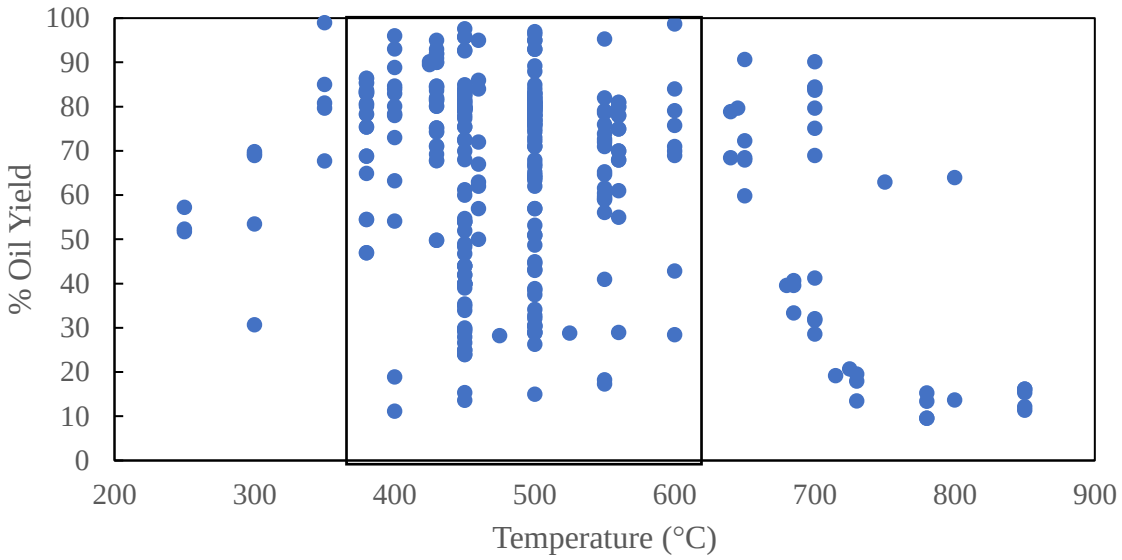

Figure 8: Distribution of temperature datapoints compared to the oil yield, for the primary dataset.

Because the RF model requires a sizable amount of information to accurately predict output variables, outlier input variables could cause problems. This can be more easily visualized at temperatures around 200 or 300°C, where the model has very little information to draw from, only 5 data points. It was unreasonable to expect that the model will have enough information to make an accurate prediction with such a small amount of information, so the choice was made to test how constraining variables would affect the performance of the models. Similarly, the dataset had a wide range of particle sizes reported from literature ranging from 0.225 to 50 mm. Figure 9 displays the particle size distribution throughout the dataset. Out of the 218 data points reported from the literature with particle size included, 174 of them were below 5mm and 133 were below 3.5mm. Based on this distribution and the constraints imposed by the Abnisa et al paper, another feature cutoff was tested in between 0 and 3.5mm particle size.[13]

The dependent variable of the dataset, the oil yield, was another variable that was constrained in the machine learning analyses. Due to the large distribution of yields, seen in Figure 7, the decision was made to focus some machine learning algorithms on certain ranges of oil yields. This was tested for both the regression and classification models. For the regression models, tests

were conducted to predict data within oil yield ranges of 60 to 100% for example. For classification, the decision was typically made to focus the oil yield on areas above 60%. This cut-off was also increased to 70% and 80% to view the differences inaccuracy. Since the area of the highest importance to the practical use of pyrolysis is areas of high yield, the focusing of the model in those areas should help to improve its accuracy.

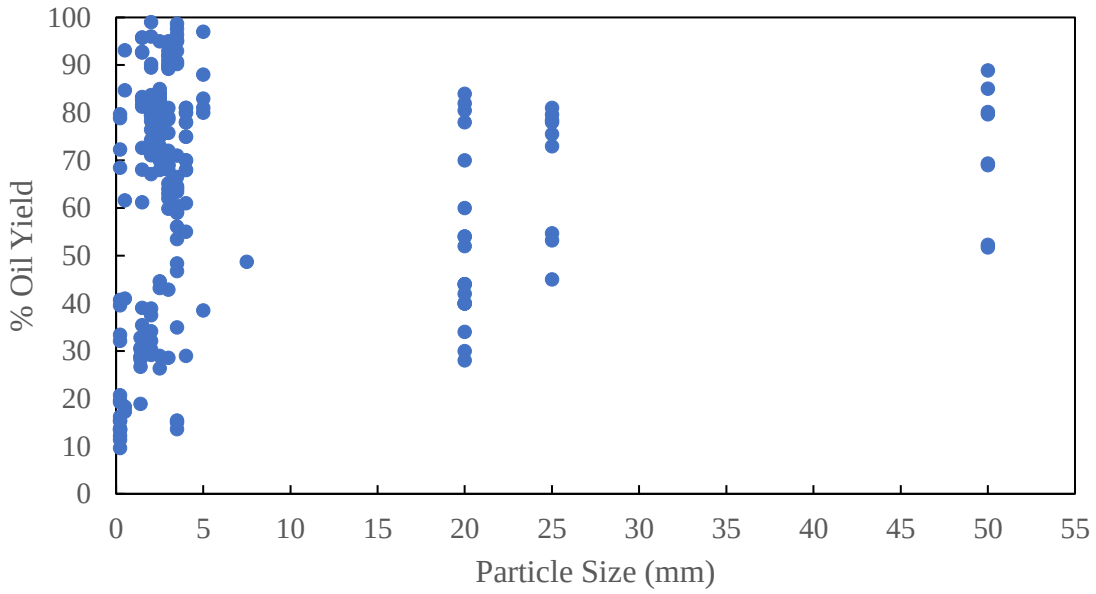

Figure 9: Particle size distribution throughout the primary dataset.

Application of Machine Learning

For the machine learning algorithm, both Random Forest regression and classification using Scikit-Learn's library were used. This was chosen due to random forest's popularity for regression problems, its simplicity, and its application in biomass pyrolysis literature. The inputs used to train the model were called features. The features used depended on the model being run. Oil yield was the output of each model. For each model run, a dataset was broken down into a training and test dataset with a split of 80% of the data in the training set and 20% of the data in the testing set. The model trains on the 80% and cannot interact with the separated 20% testing dataset. The random forest regressor and classification functions were run with $n\_estimators = 1000$ and the remainder of settings were left as default from the scikit-learn library.[18] After training was complete, the trained algorithm was run on the test set to see how well the model performed. Statistics for assessing regressors mean absolute error (MAE) and root mean absolute error (RMSE) was used. For classification models, accuracy % was used as a metric. To collect data as fast and complete as possible, models were run for 100 simulations. Each simulation would have a unique train and test split as well as metrics. These metrics were averaged throughout the 100 simulations to use one number to describe how well the algorithm performed for that model. The code for these simulations can be found in Appendices D and E. For missing data that needed to be imputed, the KNN method was utilized. When running the KNN code the number set for N

nearest neighbors was determined by taking the square root of the dataset size.[19] If an even number was the result, the number would be rounded up to the nearest odd number. This was done because KNN requires an odd number to decide which number to impute in case of a missing value.[9]

Regardless of whether imputed data was used or not, cross-validation was used as a method to test the robustness of the model. If the difference between the validation metric and test metric was high, then the model may not be considered robust no matter how well the test set performed. Cross-validation is a method where data in the training set is divided into subsets called fold or bin. All subsets besides one are then used to train the model and the last is used for a validation test. After each subset is tested once, a validation score can be averaged. Two different methods were attempted for cross-validation: one supplied by the scikit-learn library and a stratified method developed using Python.[18] The code for these can be found in Appendices F and G respectively. The scikit-learn method was a function named 'cross_val_score' which separated the model training set into the desired cross folds and outputs a validation score for the chosen metric. The number of cross folds was set to five and the rest of the parameters were left as the defaults from the scikit-learn library. The stratified method involved pre-separating a dataset into 5 equally representative excel prep sheets of oil yields. This was done due to the uneven distribution of oil yields within the collected dataset. An example of this split can be seen in Appendix C. After splitting the data into five different prep groups, these were inputted into a python code that shuffled each prep sheet. From there, 20% of each prep sheet was removed randomly to represent a test set. The remaining data in each of the five prep sheets would be split into five new groups. These new groups were known as the cross-validation bins. Each cross-validation bin would contain 20% of the data from each prep sheet. From here, the random forest algorithm was run with four of the cross-validation bins as the train set and the last as the validation set. This would repeat four more times so that each cross-validation bin would be the validation set once. From the five simulations, the metrics of MAE and RMSE were gathered for each. These were then averaged to determine the validation score of a particular model. This method was visualized with Figure 10 below.

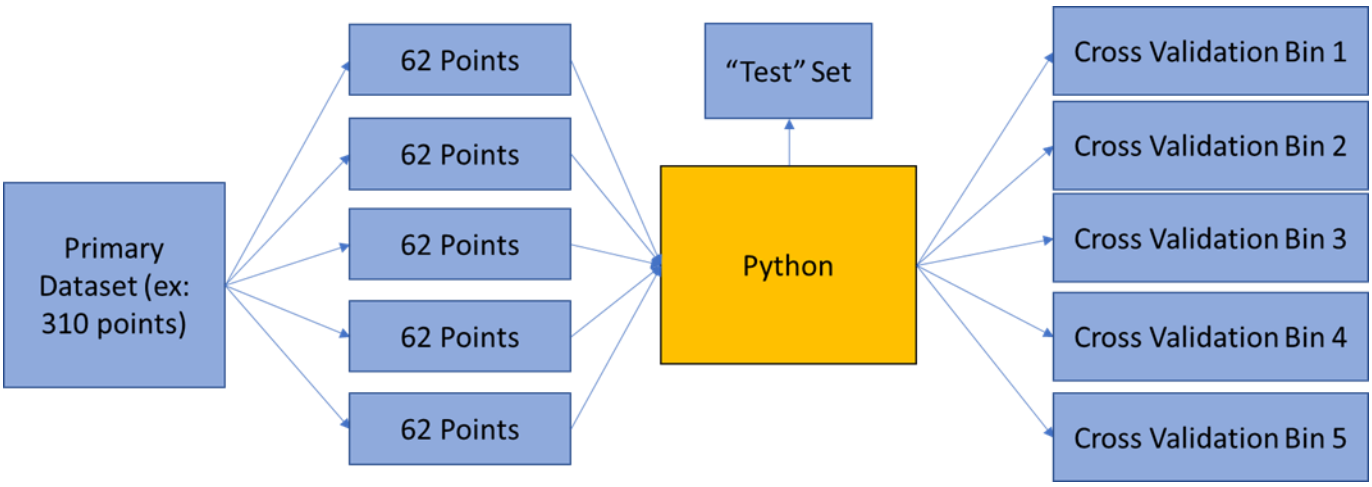

Figure 10: Stratified cross validation diagram flowchart.

Model Evaluation

Numerous parameters were calculated for several different RF regression and classifier models to test their accuracy. For regression, these metrics were the mean absolute error (MAE), and the root-mean-square error (RMSE), defined as follows,

$$MAE = \frac{(\sum_{i=1}^{n} |e_i|)}{n} \quad (3)$$

$$RMSE = \sqrt{\frac{(\sum_{i=1}^{n} |e_i|^2)}{n}} \quad (4)$$

where e_i represents the difference between measured and actual values. MAE was a metric where all the errors were summed and divided by the total data points predicted. RMSE, as defined in the background, highlighted how far from the line of best fit predicted data was. Throughout the process of creating and running different random forest models on the pyrolysis dataset, all results for regressors were compared via these two metrics to determine broadly which performed the best. For classification, accuracy was a measure of the percentage of times the model predicted the correct category.

Results

Random Forest Regression

Random forest regression was the primary machine learning method used to make predictions on pyrolysis oil yield. Within RF itself, there were many ways to tweak the model to produce more valuable and accurate results. The next four sections detail the results of these methods. For reference, Appendix B contains all results from this section.

Piecewise Addition of Variables

For piecewise models, the first test was run using only compositions as the input variables to predict oil yield. This was run as an initial base test since all the composition data was reported from literature and no data needed to be directly imputed. After that, the temperature was added as a variable since all literature sources reported temperature as well. The temperature was also tested on its own. As a result of literature reporting all these values, each of these tests was 310 data points. The results from this test are represented in Figure 11 below for MAE and RMSE. Using only compositions, the model performed around 14 and 18 for MAE and RMSE respectively. When using temperature only as an input feature, the prediction of the algorithm worsened to 16.4 MAE and 20.5 for RMSE. The combination of the two improved the prediction to 11.3 and 16.3 for MAE and RMSE respectively. This indicated that the addition and removal of features did play a critical role in determining model performance.

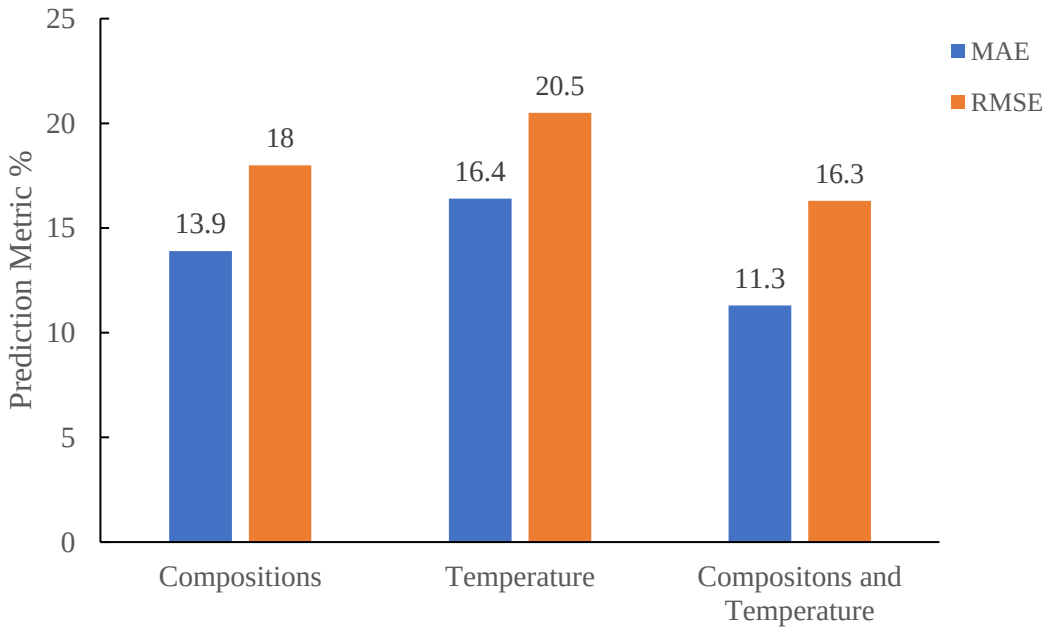

Figure 11: Initial testing of using only compositions, temperature, and a combination of the two to predict oil yield. The result is the % for MAE and RMSE for the Oil Yield

It was also of interest to see if the addition of other variables affected and improved the model on top of compositions and temperature. These two variables were the catalyst and reactor type. Both were one hot encoded into the algorithm. This was tested stepwise, first with a catalyst

added on top of compositions and temperature and then reactor type was tested with compositions and temperature separately. Finally, the two were tested together. The results can be seen in Figure 12 below. Overall, reactor type had more effect than catalyst on its own bringing the MAE down from 11.3 to 10.5 and RMSE down from 16.3 to 15.1. Adding both together resulted in slightly similar results to only adding the reactor type feature with a change in MAE from 11.3 to 10.3 and RMSE 16.3 to 14.8. However, the overall effect was not as large as it was when the temperature was added to the model.

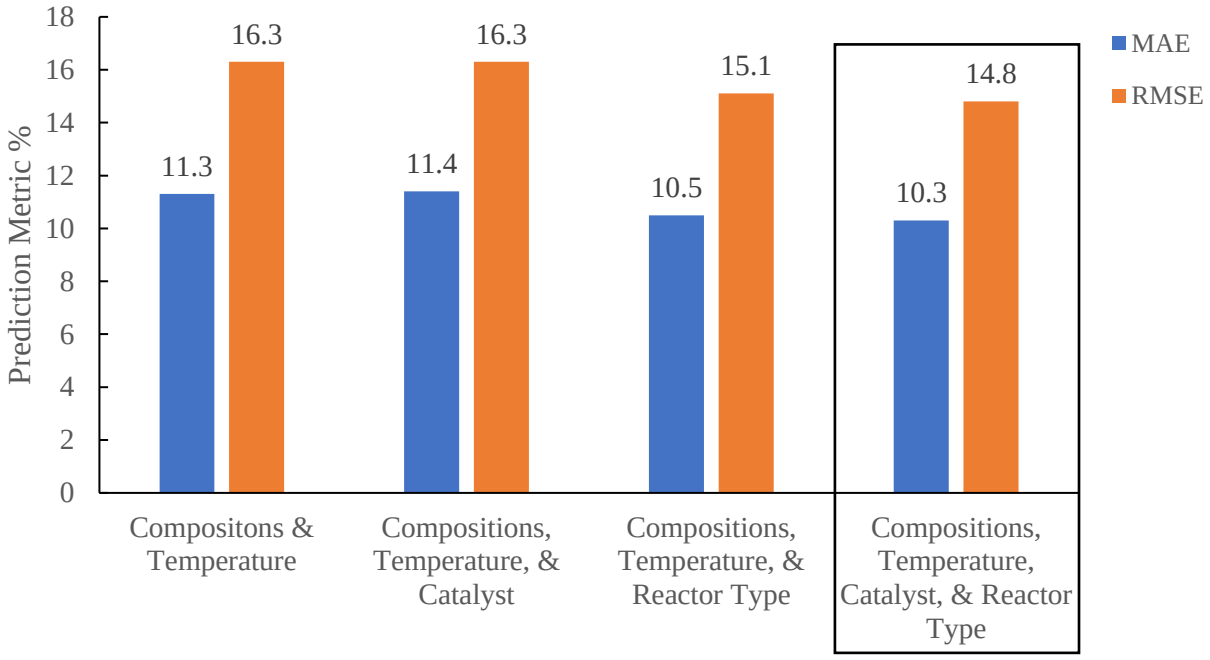

Figure 12: Observing if adding catalyst and reactor type improved the model from having only the compositions and temperature.

Nevertheless, this further showed that the addition of features improved model performance overall. From this, the effect of removing features from the developed models described in the Machine Learning Models section was attempted. This involved removing features that were not reported for all 310 data points such as feed size.

Removal of Variables

Model F, derived from the primary data set, was an unimputed model, including all the features of the primary set except for the feed size and reaction time. This model was run three times after the base model to test removing specific features from the model. The specifics of each run for the model can be seen below in Table 3.

Table 3: Model F (171 Data Points) random forest results.

Model	MAE	MAE Standard Deviation	RMSE	RMSE Standard Deviation	Features				
F.1	8.8	1.5	13.0	2.1	T	HR	PS	Cat	R type
F.2	8.9	1.4	13.4	2.2	T	HR	PS	Cat	-
F.3	9.0	1.4	12.9	2.0	T	HR	-	Cat	R type
F.4	9.5	1.4	13.7	2.0	T	HR	-	Cat	-

The progression from Model F.1 to Model F.4 revealed that as features were removed, the resulting errors in the model generally increase. This stood to reason as Model F.1 contained only 171 data points, meaning that a sizable percentage of raw data was lost with each iteration of feature removal. Models F.2 and Model F.3 are both missing a single feature, and both exhibit a slight uptick in MAE compared to the base model. The removal of the particle size and reactor type data, respectively, both caused similar increases in error. Of all the literature collected in this study, 65% of the reactions reported were carried out in a batch reactor. In comparison to the particle size data, values varied much more widely, between 0.225 and 50 mm, with many more unique values than the five possible reactor types. Despite this difference, the removal of the reactor type data, which contained less total information regarding the data set than the particle size data, resulted in a similar increase in MAE as compared to the removal of the particle size feature. The RMSE reported for F.3 was similar to the error reported by model F.1, which was understandable.

When both features were removed and tested in Model F.4, both error metrics increased largely compared to the base model F.1; both the MAE and RMSE increase by 0.7. It stands to reason that as more information was taken away from the model, that the model would become less accurate. The results from F.2 and F.3 seemed to suggest that this was not a perfectly linear trend and that there seemed to be a limit to the amount of data that can be removed from the model before an appreciable decrease in performance occurs. Additionally, this loss of information also appeared to compound upon itself. The removal of particle size and reactor type increased the MAE by about 0.1 and 0.2 respectively, but the removal of those features at the same time increased the MAE up to 0.7. This compounding error was important to take into consideration for the creation of any new models, as the loss of information can cause a decrease in model accuracy.

Independent Variable Constraints

Like the results above, other trials were created and run to test how constraining variables to be within a specific range of values would affect the overall performance of the RF regression model. Results for these trials are in Table 4.

Table 4: Variable constraint analysis on models.

Model	MAE	Std Dev	RMSE	Std Dev	Constraint(s)	Features						
A.1	8.30	1.60	12.50	2.80	None	T	HR	PS	FS	Rxn Time	Cat	R type
A.2	7.50	1.40	11.00	2.20	No >90 min Rxn Time							
A.3	8.40	1.60	12.10	2.40	No >90 min Rxn Time & T >650 C							
B.1	8.66	1.10	13.20	1.80	None	T	HR	PS	FS	Rxn Time	Cat	R type
B.5	8.45	1.46	12.32	2.30	400 to 500C							
B.6	8.14	1.73	12.32	2.90	400 to 500 C 0 -3.5 PS No 1000 FS							
D.1	8.41	1.44	12.58	2.42	None	T	HR	PS	FS	Rxn Time	Cat	
D.2	8.50	1.50	12.65	2.47	400 to 500C							
D.3	6.72	1.76	11.16	3.59	400 to 500 C 0 -3.5 PS No 1000 FS							

All ‘X.1’ models were the basis for which the models containing cutoffs were compared. Within each model, placing constraints on specific variables resulted in both increases or decreases in the error. For example, with no constraints on Model A.1, the average MAE of 100 simulations was 8.30. Excluding all reaction times greater than 90 minutes decreased the MAE to 7.50. When more constraints were added on, the model worsened in A.3 with an increase to 8.40 MAE. In this model, all reaction times greater than 90 minutes were removed, as well as all data containing temperatures larger than $650°C$.

Model B tested different constraints. In this model, the only data considered were all those data points that contained a temperature between 400 and 500°C. A small decrease from an MAE of 8.66 to 8.45 was observed but was tempered by an increase in the standard deviation. Model B.6 contained three constraints, being excluding all temperature data outside the range of 400-500°C, including only particle sizes with the range of 0-3.5 mm, and excluding feed size data of 1000g. Despite the increase in constraints, Model B experienced a slight decrease in MAE and RMSE with these constraints which contrasts with the information gathered from Model A.

Model D was like Model B, except that it did not include reactor data. This was because Model D used only batch reactor data. The first iteration of the model achieved an MAE of 8.41. Constraining the model to temperatures between 400 and 500°C produced a small uptick in error to 8.50 but adding the constraints on particle size (no values greater than 3.5 mm) and feed size (no values of 1000 g), decreased the model's MAE to 6.72, the lowest of models tested. Including constraints helped to improve certain models more than others. From these tests, it was determined that Model D.3 produced the lowest MAE and second lowest RMSE value. To determine why this was the case it was necessary to individually test specific constraints as well as to visualize the data. Using Model B.6 as an example, feed size data of 1000g, as mentioned previously, was excluded. Included below, in Figure 13, is a chart depicting the variance between oil yields and feed size within the training and testing sets. In the figure, the red points were in the train set and the blue points were in the test set.

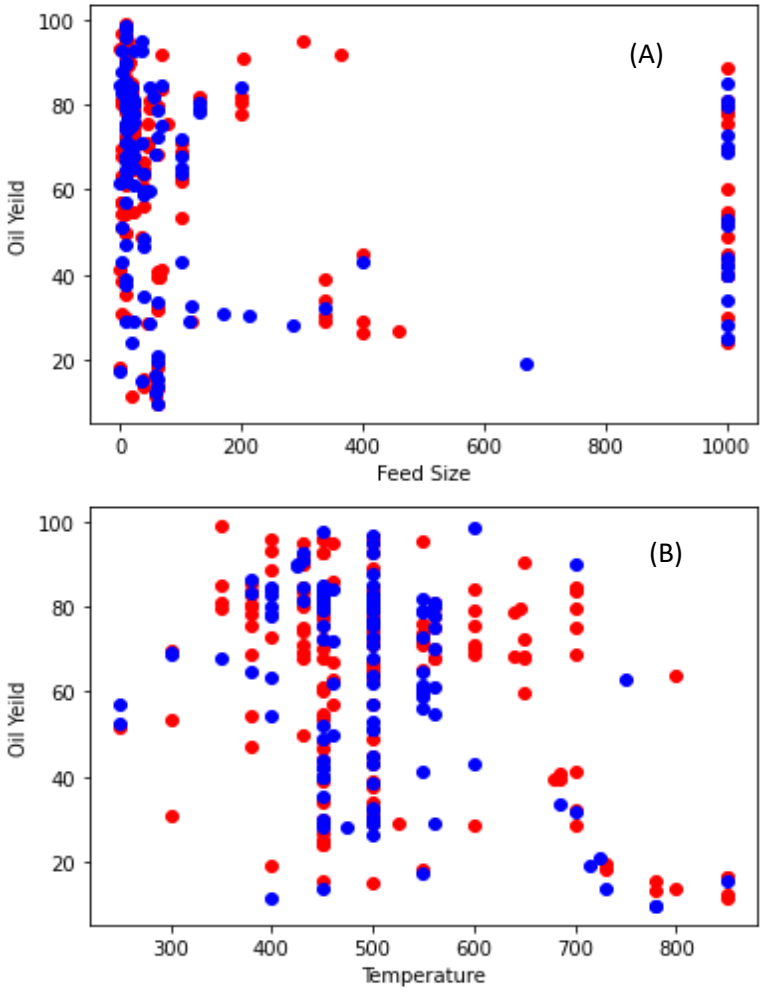

Figure 13: Charts displaying the variance between feed sizes and oil yields. (Red) points are specific points that the model used in the training set. (Blue) points are those that the regressor used in the testing set. (A) shows the oil yield distribution compared to feed size and (B) shows the oil yield distribution compared to temperature.

Figure 13 provided a rationale for understanding why removing certain data points may have been beneficial. As can be seen in the first chart (A), there was a large amount of missing information for feed sizes between 400g and 1000 g, and at 1000g there was a great amount of variance between the oil yields. For this reason, it could have been beneficial to focus the model on feed sizes below 1000g, in hopes that this would improve performance instead of trying to focus on an area with only a small amount of data. With the temperature data in the second chart (B), most of the data exist within the range of 400-500°C, with a decent chunk of data existing at about 550°C. Due to this wide range of values for both the temperatures and the oil yields at those temperatures, performance was improved in the models by constraining to an area where it may better train and test itself. Ideally, the temperature data would contain a relatively healthy distribution of red points and blue points, indicating that the model tested and trained itself on similar data points, which would lend itself to better predict oil yield. Because of the large swath of temperatures, the model may not be able to test itself for many different values. For example, in chart (B) displaying oil yield versus temperature, the model tested itself with the highest yield data points at both 600°C and 700°C, though it trained itself with lower yield data points. This could have been a source of error within the model due to the slight differences between the testing and training set; neither set of data contained a complete distribution of temperatures and yields.

Oil Yield Constraints

The fourth method of feature engineering was to test the model which included data within a specific range of oil yields. As pyrolysis is a chemical process, the most beneficial physical result of a machine learning analysis would be to provide information that helps to maximize the oil yield for future reactions. As such, in some cases, oil yields that were below a certain threshold, for example, 40% or 60%, were removed from the data set and the model was trained and tested only upon those points which contained oil yields above those values. This was additionally beneficial, as for most of the models, high yield data points were the most plentiful, and as such, the model was able to predict them more accurately. Below is Figure 14 which is a parity plot displaying the difference between high and low yield data points with respect to the error from the predictions.

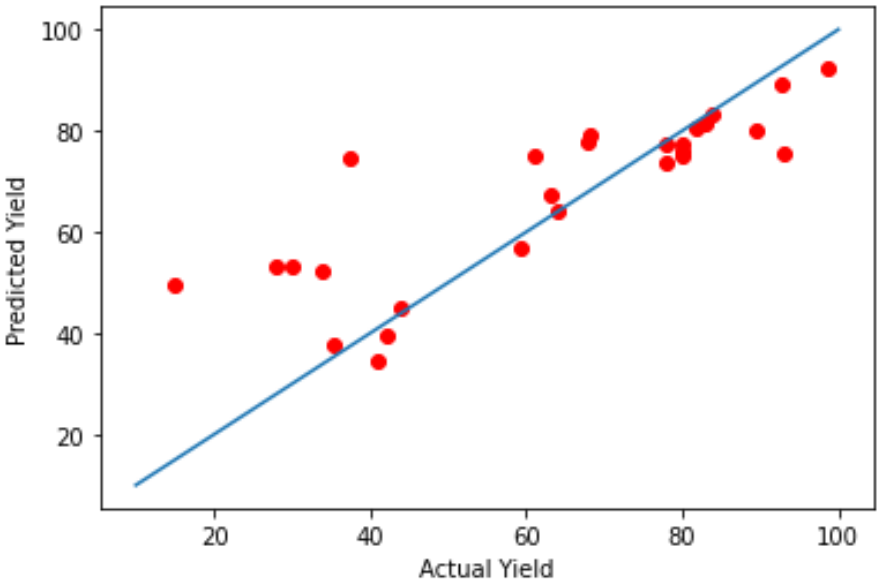

Figure 14: This chart indicated the closeness of predicted oil yield values to actual oil yield values for Model E. There was a noticeable decrease in accuracy for oil yield values between 20-60% versus 60-80%.

To include all the low yield data points would be to provide a more holistic model of pyrolysis, but from a perspective of pure utility, the most important values to predict correctly were those of high yields. Having a model that could predict well at high yields could help tailor reactions to operate as efficiently as possible and to produce as much oil as possible. There are essentially no benefits of operating at a low oil yield, and as such, it was beneficial to see how the RF models reacted to removing areas of low oil yield. Figure 15 below, displays the results of these tests on Models A and B.

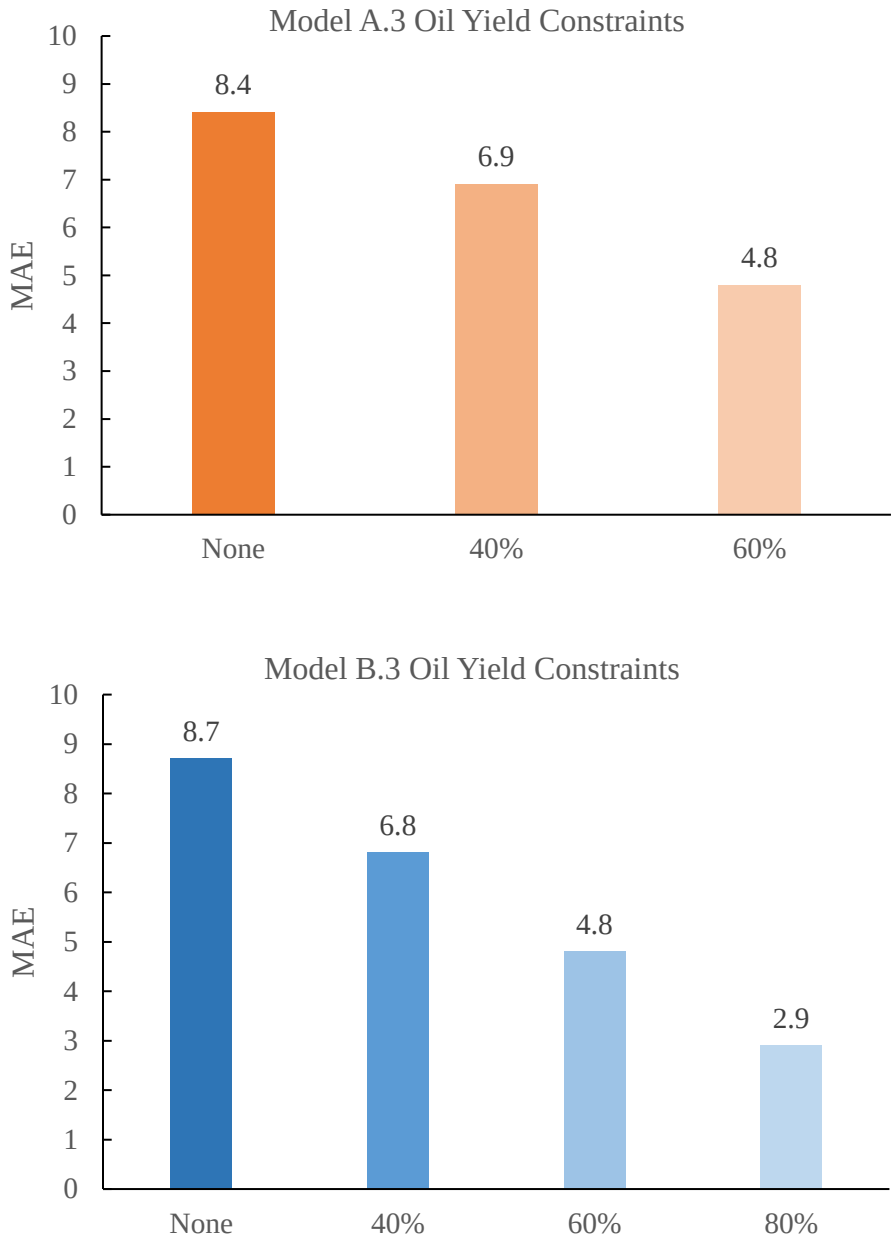

Figure 15: The charts above indicate how the MAE of a model changed by varying the oil yield cutoff. Both models have no constraint to start, and successive bars indicate that only data above a certain oil yield was used within that model. Increasing the oil yield constraints was shown to increase the accuracy of Models A and B.

For Models A.3 and B.3, both the MAE and RMSE of the model decreased drastically via the removal of low oil yield data. With no constraints on the data, Model A.3 output an MAE of 8.4. When data below 40% oil yield was removed, the MAE decreased to 6.9, and when the all-oil yields below 60% were removed, the MAE decreased further to 4.8. Model B.3 exhibited similar

behavior. With no oil yield constraints, the MAE was 8.4, but with all data below 40% oil yield removed, the model's MAE decreased to 6.8. Subsequent removal of oil yields below 60% returned an MAE of 4.8, and finally, removing data below 80% resulted in the best MAE of 2.9. Part of the removal and constraining of data included an inherent give and take within the models. For every constraint placed upon the oil yield, the dataset lost a significant portion of its information. Despite this loss, the benefit of tailoring this pyrolysis dataset to areas of high yield seemed to improve the model's performance more than the loss of data detracted from the performance. There were likely a couple of reasons why this was the case. One was due to the large concentration of high yield data points within the primary dataset. Table 5 below illustrates both the MAE results of the models shown above, as well as the number of data points included in those models. Model B.3's original run contained 265 data points, whereas its final run, at an oil constraint of >80%, including 95 data points, roughly a third of its original size. This indicated that the dataset was skewed toward regimes of high oil yield, which was understandable due to the importance of chemists and chemical engineers to operate pyrolysis reactions in these areas of high yield. Additionally, there was a level of hesitance in this analysis, as constraining the oil yield meant that the model had a smaller range of answers to report. Hence, the lower MAE was inevitable since a lower range of oil yields meant the error was constrained to that range. With this concession, the results, particularly at the 60% cutoff, were impressive, since within that range from 60 to 100% oil yield, the model could predict within 5%. This would provide enough accuracy to discern whether a model was closer to 60% or a higher 90% yield. Furthermore, if improved, these models could predict with increased clarity in the future.

Table 5: Oil yield removal results.

Model	Oil Yield Constraint	MAE	RMSE	Data Points
A.3	None	8.4	12.1	160
	40	6.9	9.9	139
	60	4.8	6.4	110
B.3	None	8.7	13.2	265
	40	6.8	9.6	251
	60	4.8	6.7	201
	80	2.9	4.1	95

Overall, the performance of these models deviated from the literature sources of Zhu et al., where bio-char was predicted with RMSE values around 3.5 and Tang et al., where bio-oil was predicted with RMSE values around 3.[8,12] This could have been due to several reasons. One of the most pressing being that these were predicted for biomass and not plastic and therefore input variables, especially for composition, differed greatly. In addition, the paper that predicted oil yields did so between 15% and 50% and the paper that predicted bio-char yield did so between 0% and 50% yield. The base models in this study predicted from 0% to 100% and therefore a larger MAE and RMSE could be expected compared to these studies. Furthermore, the predictions of yield constraints in Table 5 reflected RMSE values that were closer to the literature values for the

60% cutoff which had a range of 40%. A range of 40% more closely resembled the literature and RMSE's of around 6 for Model's A and B were not far from either Zhu et al. nor Tang et al.'s model performance. From this, it can be stated that there is promising room for improvement on plastic pyrolysis going forward with the results of this novel approach for random forest. In comparison to the Abnisa et al. paper with 24 datapoints for plastics pyrolysis, where the MSE was near zero, these models cannot reflect that level of accuracy. However, that paper used neural networks for computation not random forest. This indicated that potentially using neural networks could improve the results for this study further.

Random Forest Classification

In addition to the regression models, classification was also used to predict oil yields. For one method, oil yields were determined on a binary basis. In this case, classification was used to predict above and below a specific oil yield, giving two possible answers. Alternatively, other tests with Model H used several classifications for ranges of oil yields where numbers for each category were assigned. Classification was tested piecewise like the regression models by adding variables that were complete from literature values. In addition, tests were run on both variations of the imputed Model G and unimputed Model I. 60% was used as the base oil yield cutoff for classification models. Higher cutoffs of 70% and 80% were also tested to observe the effects of that on the model.

Piecewise Addition of Variables

Like the piecewise additions for the regression models, the classification models were tested with compositions only, temperature only, and a combination of the two to predict oil yields. The cutoff for the binary 0 or 1 was above or below 60% oil yield for these tests. All the models tested for piecewise addition contained all 310 data points. The results can be seen in Figure 16 below. Using compositions only resulted in an accuracy of 77%, whereas temperature only data resulted in lower accuracy of 69%. Adding both together increased the accuracy to 83%. This was a similar trend to what was observed for the regression models where the combination of the two variables improved the model and that only temperature as an input did not perform as well as compositions only to predict oil yields.

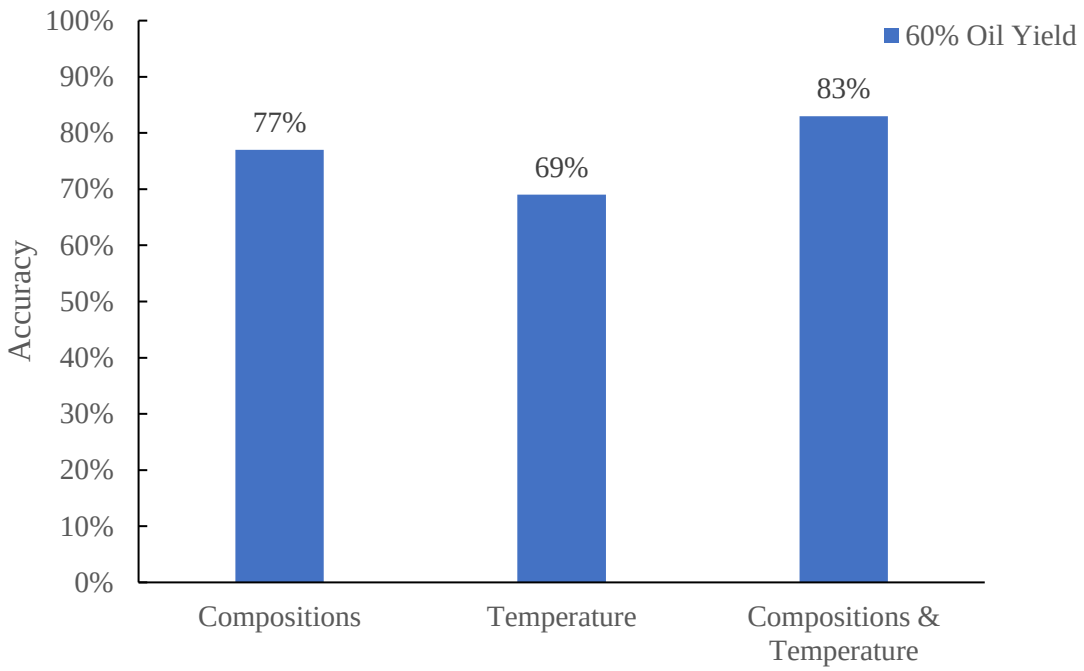

Figure 16: Piecewise addition of variables of composition and temperature to observe the performance of the random forest classification model at 60%.

For the piecewise method, the addition of catalyst and reactor type independently was also tested at the 60% yield cutoff. This followed a similar approach that was taken for the regression problems. These results can be seen in Figure 17 below. From these models, there was a slight improvement over only using the compositions and temperature to predict oil yields from 83% to 85%. Both the independent addition of catalyst and reactor type on top of compositions and temperature resulted in the same accuracy of 85%. The addition of both together improved the accuracy to 88% which was the best out of all the piecewise models. These results were slightly different from the trends seen for the regression piecewise addition where the addition of reactor type performed better than the addition of catalyst.

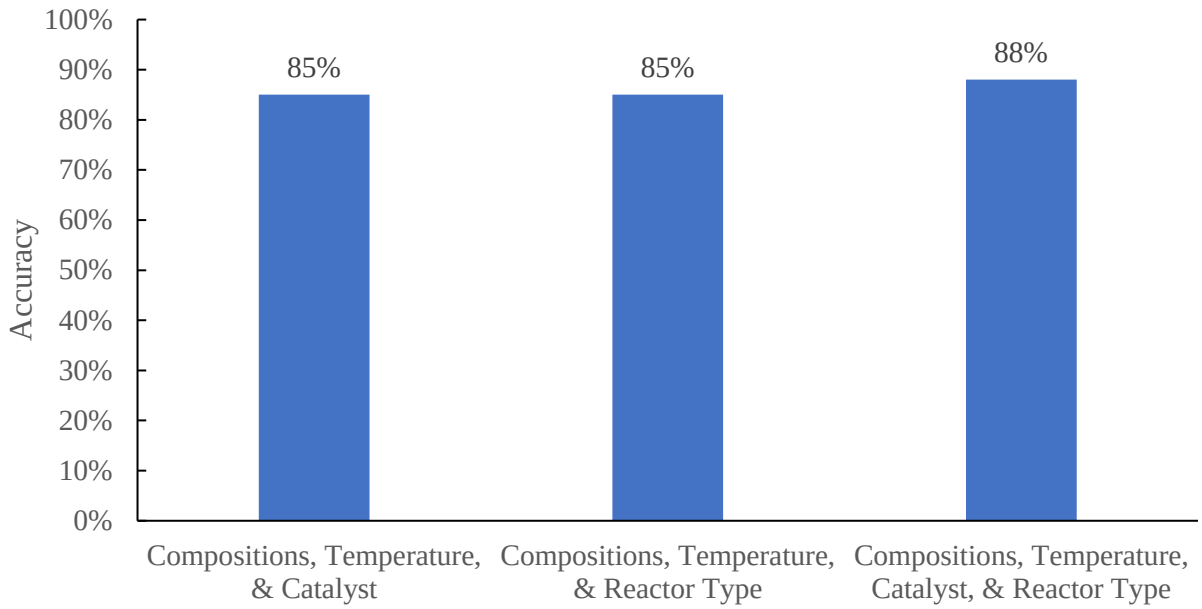

Figure 17: Piecewise addition of variables of catalyst and reactor type to observe the performance of the random forest classification model at 60%.

Oil Yield Grouping Classification

For oil yield grouping tests, the dataset was made up of 171 points (Model F) was divided into sections based on the total oil yield. This was named Model H. The goal of this method was to test to see if classifying the data into groups would yield better results than the regression models or binary classification. The groups were created by splitting up the oil yield percentages into categories. For example, the 171 data points were broken into 7 groups. The groups were created to make each group roughly the same number of data points be more representative of the whole model and not overpredict a certain group over another. The group breakdowns and the number of datapoints contained within each can be seen in Table 6.

Table 6: Oil yield grouping for classification Model H.1 and H.2

Group #	H.1 Oil Yield Range	Data Points	H.2 Oil Yield Range	Number of Data Points
1	0-30%	18	0-45%	36
2	30-50%	29	45-60%	32
3	50-65%	21	60-75%	56
4	65-75%	24	75-100%	47
5	75-80%	25	N/A	N/A
6	80-85%	32	N/A	N/A
7	85-100%	22	N/A	N/A

From Model H.1, after 100 runs the average accuracy resulted in 46%. Seeing the low accuracy result compared to all the piecewise models, an additional test was done with decreasing the number of groups from seven to four which increased the number of data points in each representative group. This was be referred to as Model H.2 and can also be seen in Table 7. The accuracy then increased to 60%. Comparing Model H.1 and Model H.2 directly showed that the smaller number of groups increased the accuracy of the model overall. However, neither accuracy was impressive compared to the models run with a binary oil yield result. Due to this, further attempts at grouping oil yields outside of the typical binary method were not attempted.

Table 7: Classification grouping method results.

Model Number	Variables	Accuracy (Average of 100 runs)	Std Dev	Points
Model H.1	T, HR, PS, no FS, Cat, R type	46%	8%	171
Model H.2	T, HR, PS, no FS, Cat, R type	60%	7%	171

Binary Classification of Models

The developed models for binary classification, Models G and I, were tested with several cutoffs of oil yields of 60%, 70%, and 80%. Model G was the classification version of Model B which contained 310 datapoints and Model I was the classification version of Model F which contained 171 datapoints. Model G included imputed data whereas Model I was not. The results for these simulations can be seen in Figure 18.

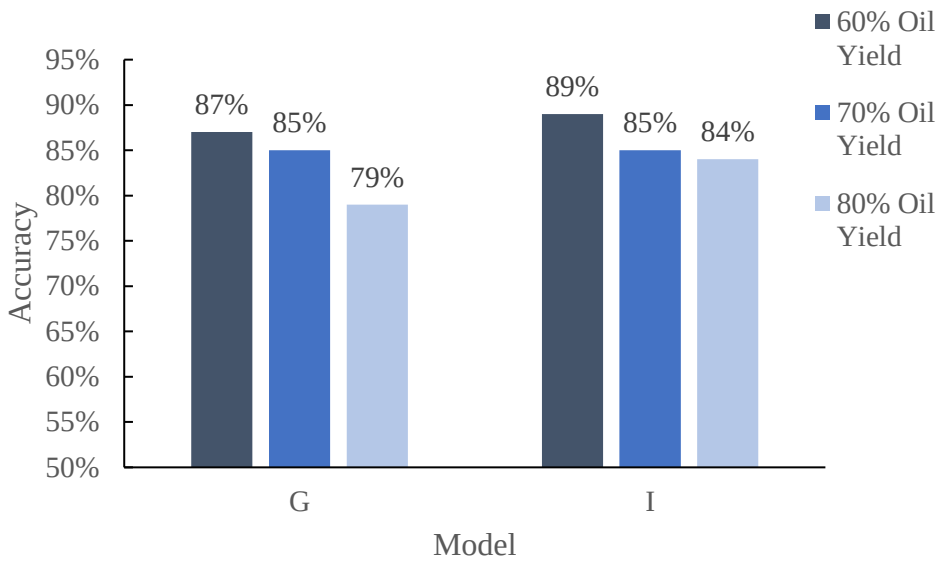

Figure 18: Classification results at various oil yield cutoffs for the imputed Model G and unimputed Model I.

The accuracies for the 60% oil yield cutoff were 87% and 89% for Models G and I respectively. Interestingly, these models did not significantly outperform the 88% accuracy found by only using compositions, temperature, catalyst, and reactor type as input variables. This indicated that the extra variables of particle size, feed size, and reaction time did not appear to play a large role in determining classifier accuracy nor were those variables needed for a well-performing classifier model. Furthermore, the results around 90% accuracy were extremely promising for a classifier's use for determining oil yield. As the oil yield cutoff increased, the accuracy expectedly decreased due to fewer datapoints within each successive increase in the cutoff. The number of datapoints above and below the cutoffs for the imputed and piecewise models is visualized in Figure 19. For 60% yield, nearly two-thirds of the data was above the cutoff. The number of datapoints nearly evened out for the 70% cutoff and of the weight shifted to around only one-third of the data above the threshold for the 80% cutoff. Hence, maintaining high accuracy across the three cutoffs showed the success of the classification models even when the data was unevenly shifted towards predicting below 80%. Figure 20 displays the datapoint numbers for the unimputed model. The spread was like the 310 datapoint set where the 60% yield had nearly two-thirds above the cutoff. At the 70% cutoff, nearly half was above, and at 80% around only one-third was above. This similarity was a product of the spread of the dataset where

many datapoints were in the 60 to 90% oil range. In addition, this indicated that the slightly better performance of the unimputed model was not solely tied to a difference in datapoints above or below the threshold.

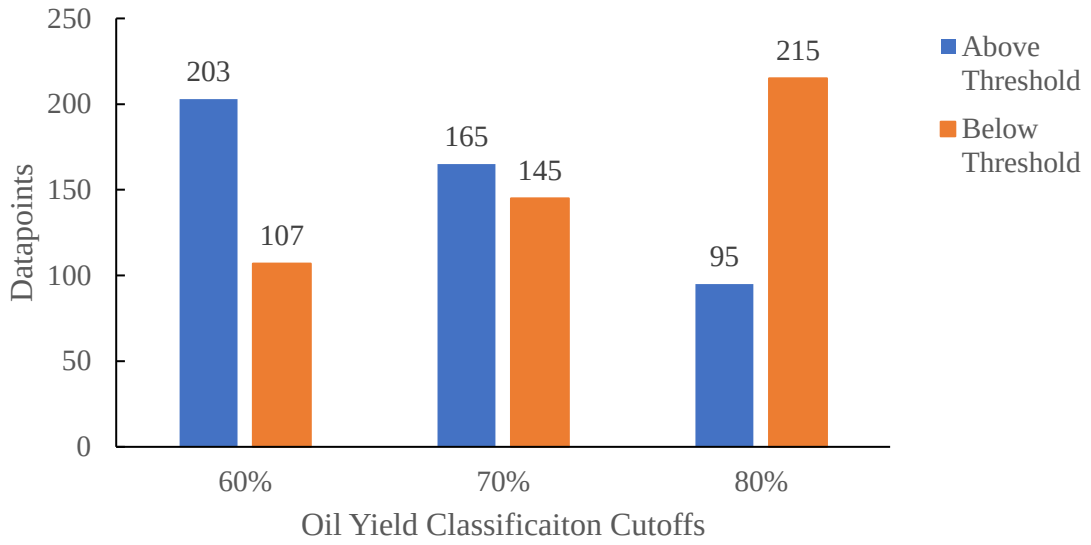

Figure 19: The number of datapoints above and below the oil yield cutoffs for classification models. This applied only to the imputed classification and piecewise models due to the inclusion of 310 datapoints.

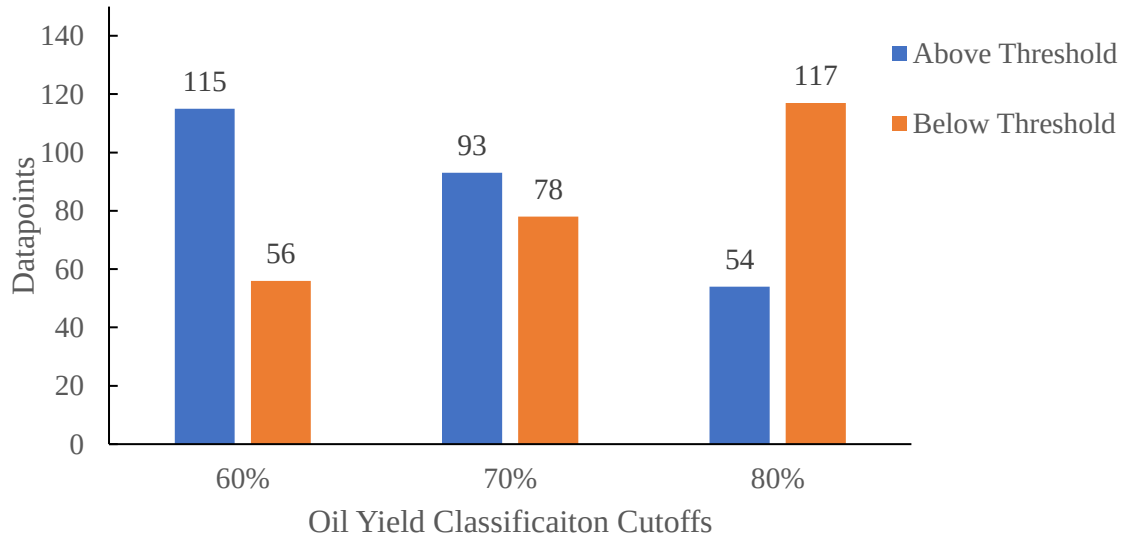

Figure 20: Unimputed model number of datapoints above and below the thresholds.

The imputed model matched the unimputed model at the 70% cutoff. However, the unimputed model performed better than the imputed model by an accuracy of around 5% at the 80% cutoff. Despite the decrease in accuracy with an increase in yield cutoff, the results from 79% to 84% at the highest cutoffs added to the promise that the classifier models were successful. Due to the comparative performance by the piecewise addition model, the last piecewise model with

temperature, catalyst, and reactor type data was run at the 70% and 80% yield cutoffs. Figure 21 shows the results of this attempt. While the piecewise model had the highest accuracy for the 60% cutoff, it did not maintain a higher accuracy compared to Model G and I for the 70% and 80% cutoffs. Both the imputed Model G and unimputed Model I perform better by 10% for the 70% oil yield cutoff. This likely meant that increasing the features such as particle size and reaction time allowed the model to maintain high classification performance at the higher cutoffs. The piecewise model's success at 60% remained impressive but was likely due to the larger number of datapoints above the cutoff since its accuracy significantly decreased when more datapoints moved below the threshold. The accuracy did not decrease largely from the 70% to 80% cutoff which resembles the trend that occurred to Model I at those cutoffs.

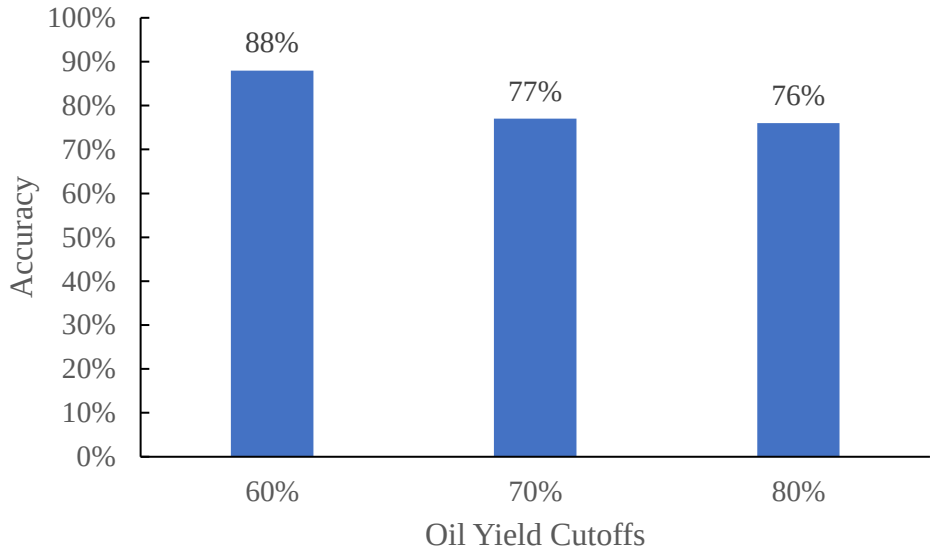

Figure 21: Piecewise composition, temperature, catalyst, and reactor type at cutoffs of 60, 70, and 80% oil yield.

Feature cutoffs were also attempted with both the imputed and unimputed models. These feature cutoffs mirrored the strategies that were used for the regression models. First, Models I and G were tested with the constraint of 400 to 500 ℃ and then these models were additionally tested with the temperature constraint and a constraint within 0 to 3.5 mm particle size. Each model was tested within the 60%, 70% , and 80% yield cutoffs. The datapoints for each of these models and associated feature cutoffs can be seen in Table 8 below.

Table 8: Datapoints for each classification model at each cutoff.

Feature Cutoffs	Model G Datapoints	Model I Datapoints
None	310	171
400 to 500 ℃	199	124
400-500 ℃ & 0-3.5 mm	152	99

These were important to consider because as the cutoffs were added the models began to lose datapoints to train and test on. While this was inevitable with this strategy of feature engineering, it was important to know how many datapoints were removed with each cutoff. The results for each cutoff can be seen in Figures 22 and 23 below. For clarity, Model G was color-coded blue, and Model I was color-coded orange. Like the regression models, the effect of feature cutoffs was largely model-dependent and in the case of classification, oil yield cutoff was dependent as well. Model G's cutoffs saw consistent trends where increasing the cutoffs increased the accuracy at the 60% oil cutoff, steady accuracy at the 70% oil cutoff, and a decreasing accuracy at the 80% oil cutoff. It was interesting that feature cutoffs improved the lower yield cutoffs and negatively impacted the higher yield cutoffs. This could have been related to the number of datapoints which was lowered by increased feature removal combined with fewer datapoints above the oil prediction.

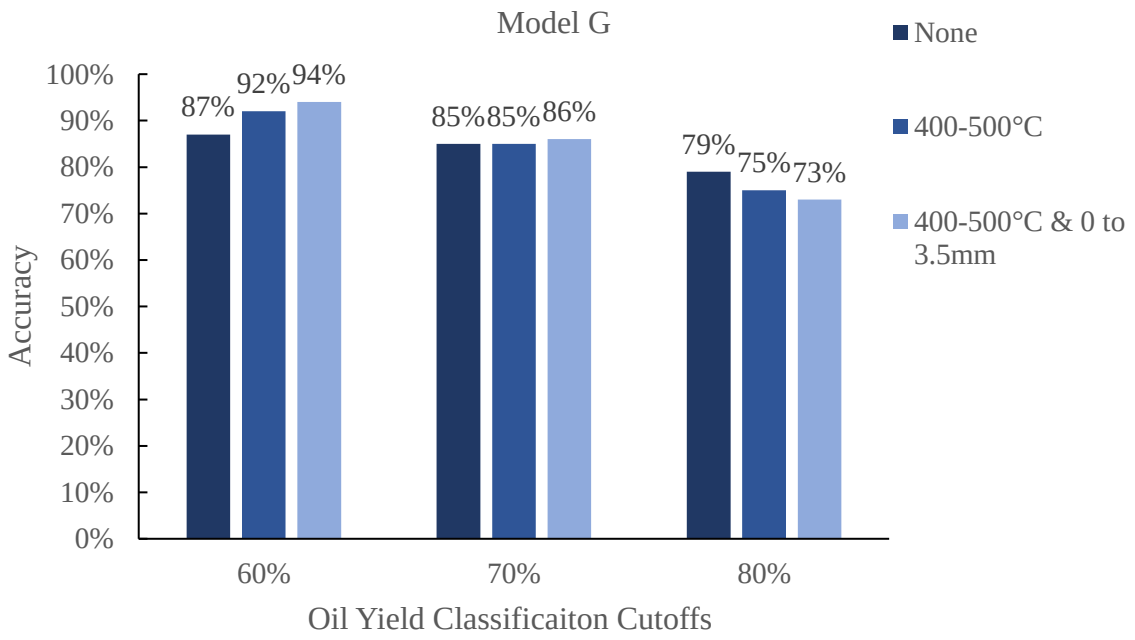

Figure 22: Feature cutoffs accuracy result at each oil yield cutoff for the imputed Model G.

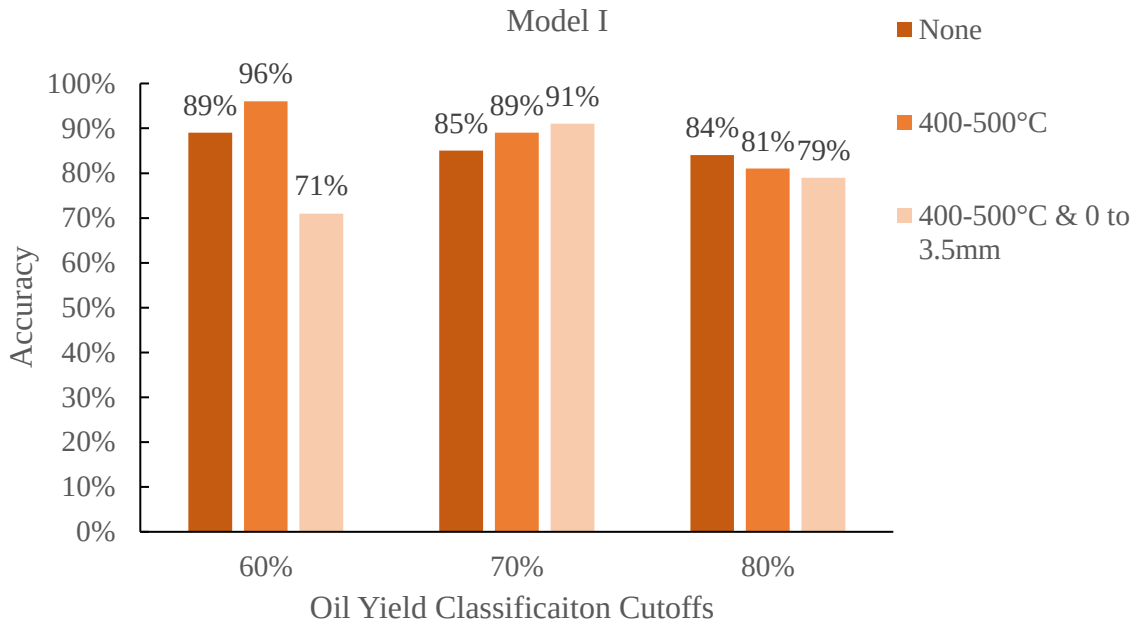

Figure 23: Feature cutoffs accuracy result at each oil yield cutoff for the unimputed Model I.

The imputed Model I provided different trends from the feature engineering. For the 60% yield cutoff, the initial 400 to 500℃ range of data provided a 7% increase in accuracy up to 96%. However, the next cutoff removing data outside of 0 to 3.5 mm particle size decreased the accuracy dramatically to 71%. The exact reason for this was unknown. While the unimputed model performed better initially, critical information might have been lost when the unimputed model was reduced to 99 data points as opposed to the imputed 152 data points at the 400 to 500℃ and 0 to 3.5mm particle range. Another difference from the imputed model was that the unimputed Model I improved as feature cutoffs were added at the 70% oil yield cutoff level. The 80% yield cutoff provided a similar trend of decreasing accuracy percentage as more feature refining was increased. Overall, the unimputed Model I outperformed Model G with the feature cutoffs except for the low accuracy seen for increasing cutoffs at 60% oil yield. In addition, this data showed that models would need to be run independently to determine if feature removal could provide an increase in accuracy. Another important consideration was the tradeoff between a model that was broader with slightly lower accuracy than one that could only predict within a certain range of the data. Altogether, classification proved to be a successful and interesting viewpoint for machine learning on the pyrolysis dataset. The unimputed model slightly outperformed the imputed model at almost all oil yield cutoffs. Accuracies in the range of 80% to 90% were considered impressive especially for the small dataset size. Ideally, these classification results could be compared to literature, however, the pyrolysis machine learning literature space has solely focused on regression problems at this point. This data showed that a combination of the two, both regression and classification, could be a powerful tool in determining how well an algorithm could predict oil yields. First, the model could predict which data was above or below a specific oil yield, and then a regression model could be used to pinpoint the oil yield percentage.

Feature Importance

In addition to running the models and observing the machine learning metrics, the relative importance of each input variable was also determined for both the regression and classification problems. This was also done alongside the 100 simulations where each feature's importance would be recorded for each simulation run and then averaged after completion. The importance scores all add up to 100%. To compute the importance, the importance method within the Python sci-kit learn library was used.

Regression Variable Importance

Importance's were recorded for several of the regression models. Initially, the importance was tested on the piecewise model with compositions, temperature, catalyst, and reactor type. Importance's were also run on the base Model B.1 and E.1 to compare the difference between the imputed and unimputed model importance's. Furthermore, Model D.3 was tested because D.3 reported the best regression metrics after feature engineering. From testing the importance's of the piecewise model, the temperature was by far the most important input into the model at 40%. This was followed by the compositions of PET, HDPE, and PolyS from 9% to 13%. Figure 24 below includes the results of these tests and the importance's for running solely the compositions only was also included. With compositions alone, PET, HDPE, and PolyS remain the most important compositions and PP gained more importance than when included with the other variables. Interestingly, LDPE did not appear to have significant importance in either model. The lowest importance of PVC was likely due to there being only 7 datapoints out of the 310 that had PVC in them. Another interesting takeaway was the lack of importance the catalyst and reactor type had on the model. Added together the reactor type only contributed 10% importance total with the catalyst only being 4% of importance.

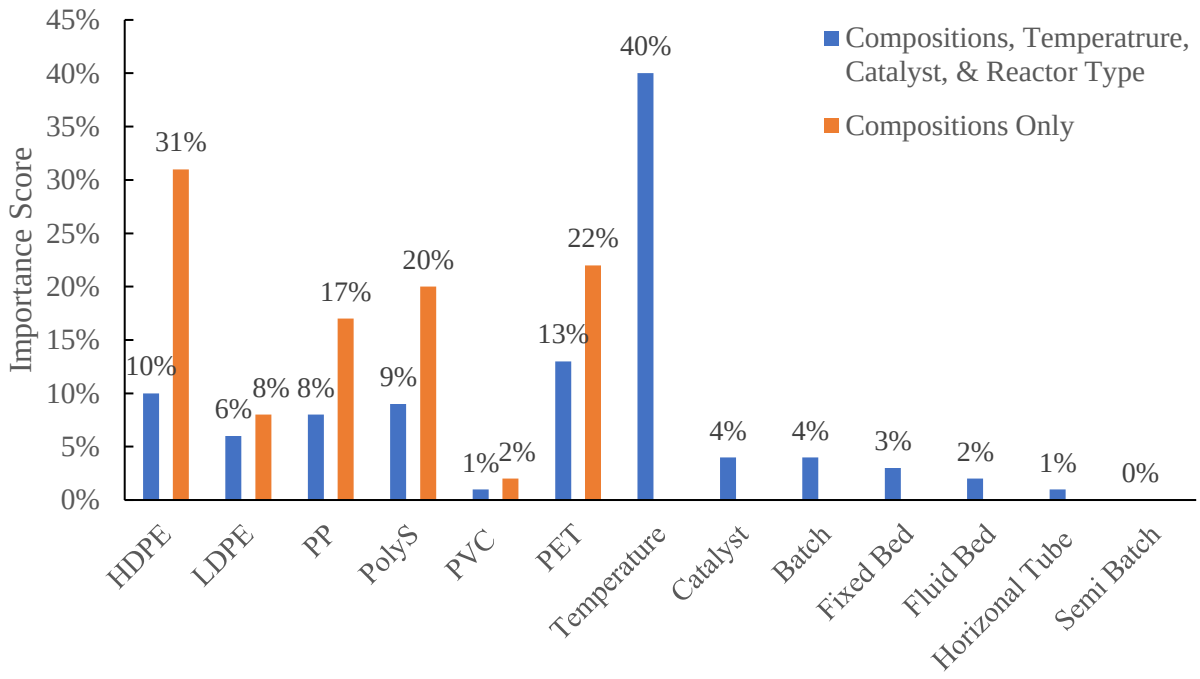

Figure 24: Piecewise model importance's for regression.

For comparing the imputed base Model B.1 and unimputed base Model F.1 input importance's, Figure 25 was included below. Model F.1 did not include feed size nor reaction time and for that reason, there was no importance score for those variables. Like the piecewise model, the importance's of the catalyst and reactor type were low and almost were not important to the model at all. For Model F.1, the compositions remained roughly like the piecewise model, except for PET, which rose to 21%. Furthermore, the effect of temperature decreased but remained similar for both B.1 and F.1. The addition of particle size shifted the importance towards it for both models and it was the most important input as well at 24%. For the imputed model, the feed size and reaction time also appeared to play important roles and it appeared that the importance for PET and other compositions decreased due to the addition of those variables.

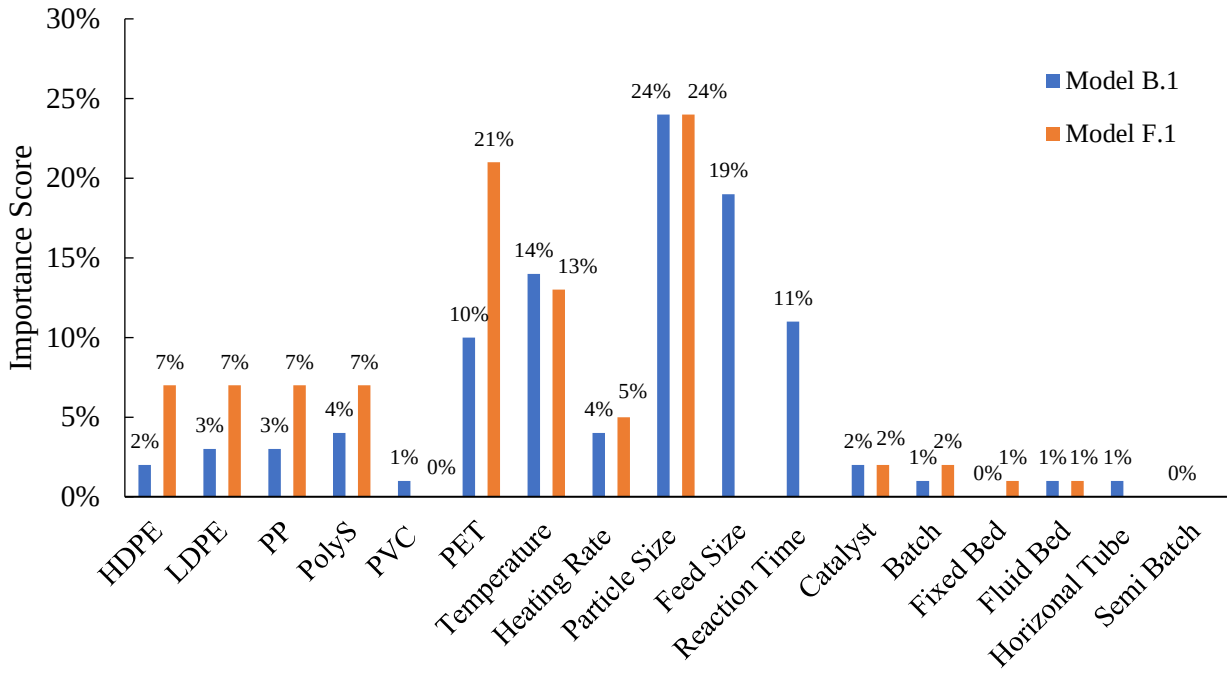

Figure 25: Feature importance's for Model's B.1 and F.1 to observe the differences in importance's for the imputed and unimputed models.

The importance of PET specifically was striking and was further shown by finding the importance's for Model D.3 which provided the best MAE of around 6.8. These importance's were included in Figure 26. For Model D.3, the importance of PET increased dramatically to 51% and the reaction time was the second most important variable at 22%. The importance of the rest of the compositions was strikingly low around 6% altogether. Particle size, feed size, and temperature's importance all decreased as well to around 5%. The only variables that remained consistent with importance's found for Model's B.1 and F.1 were heating rate and catalyst. Due to the high importance of PET, Model D.3 was run over 100 simulations with no PET input variable. The result of this was an MAE of 9.3 and for this reason, the PET composition must have positively influenced the model and played a role in its MAE of 6.8.

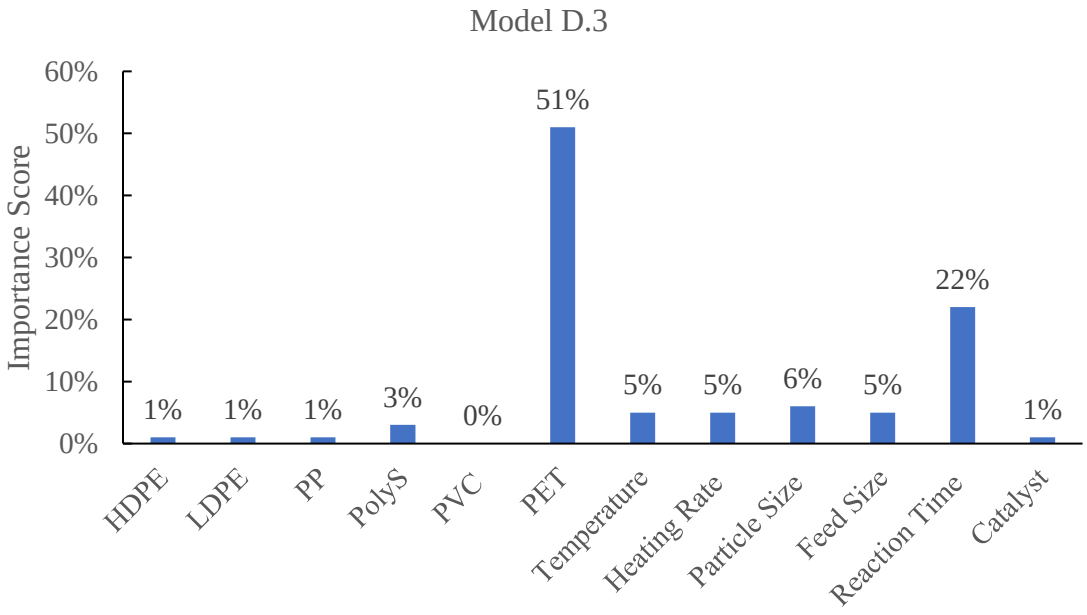

Figure 26: Model D.3 feature importance's. This was performed because Model D.3 provided the best MAE from the regression models after feature engineering.

Classification Variable Importance

The importance of classifiers was tested to compare to the regression importance's to observe if the difference in method had an effect. For the classifiers, the piecewise model with compositions, temperature, catalyst, and reactor type was tested along with the base Models G.1 and I.1. Model H importance's were not run due to the low resulting accuracy score discussed in the last section. The results of the piecewise model are seen in Figure 27 and the results of G.1 and H.1 are in Figure 28. The piecewise results were like the regression piecewise importance's. Temperature was 6% lower in importance but remained the most important variable.

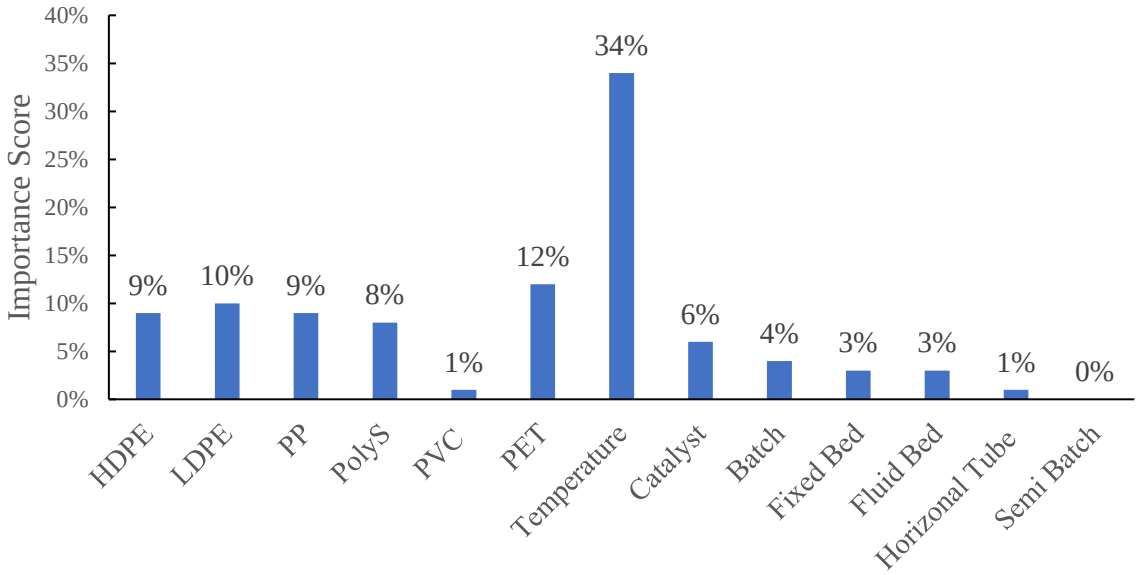

Figure 27: Chart displaying variable importance for the piecewise classification model.

For Models G.1 and I.1, a similar result to the regression models were seen where the unimputed model had more importance on the compositions than the imputed model. Furthermore, in both, the catalyst and reactor types still appeared to not play an important role in the classifiers either. The relative importance of PET, temperature, particle size, feed size, and reaction time were also consistent. This indicated that the RF regressor and classifier treated similar models with similar importance with respect to the input variables.

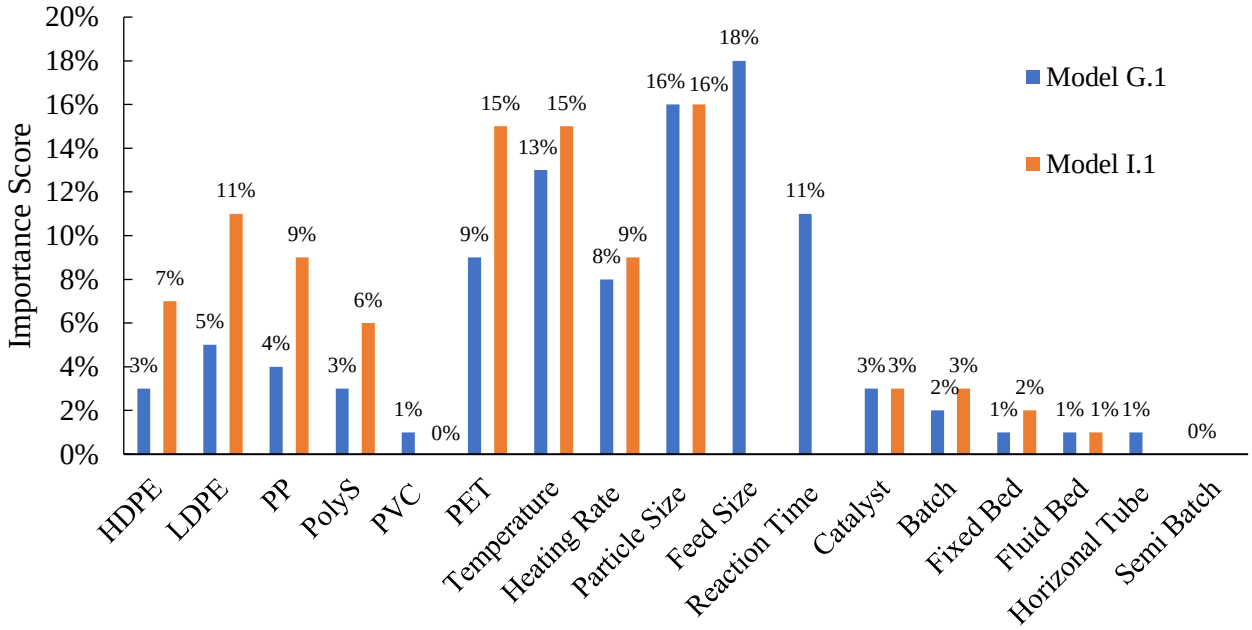

Figure 28: Imputed Model G.1 and unimputed Model H.1 importance's for comparison to the regression importance's.

In comparison to literature importance's, there was a lack in pyrolysis of plastics machine learning overall. However, two papers reported RF importance's in the biomass pyrolysis literature. For bio-oil yield, one paper found that the compositions were 60% of the importance. For biomass, compositions were fixed carbon, ash, and volatiles. The most important variables for process conditions were around 16% for heating rate and around 10% for particle size. Temperature had an importance around 5%. This was lower than expected considering the researchers expected the temperature to play the most important role of all pyrolysis conditions.[8] While this study focused on biomass, the results were compared to the results for pyrolysis of plastics. For the unimputed Model F.1, the compositions were around 50% in importance which was slightly lower than the biomass composition's importance. However, for the imputed model B.1, the importance of the compositions dropped to 23%. Interestingly, the models in this study found a similar trend with the importance of particle size as a pyrolysis condition of around 10%. A paper investigating RF regression to predict bio-char yield found that the temperature was the most important.[12] While this met the expectations of the other paper, the comparison was difficult due to the end products of bio-oil and bio-char being different. Hence, a difference in what machine learning models would consider important for predicting each output seemed plausible. Altogether, further pyrolysis machine learning would need to be performed to determine whether the importance's in this study match expectations. Nevertheless, these importance's still provided critical insight into how the model performed and why it may have performed in a certain way.

Cross-Validation

The primary method for gathering data for regression metrics of MAE and RMSE and classification accuracy for testing models was 100 simulations of the algorithm. However, it was also necessary to assess the robustness of the various models created. This was done by performing cross-validation on the models. This was performed as described in the methods using two approaches: Python's 'cross_val_score' with 5 cross folds and a stratified sampling 5 cross-fold validation.

Python's Sklearn 'Cross_val_score' on Regression Models

Python's cross-validation method allowed for the determination of regression and classification metrics utilizing a standard 5 cross-fold validation. Validation using Python's cross-validation initially revealed large discrepancies for regression models determined from 100 simulations and the 'cross_val_score' function. These differences were often twice as large as the metrics from the simulation tests. For example, Table 9 below, displays the difference between the two for the unimputed models, Models E and F. The base 100 simulation runs of Model E, which had 132 datapoints, resulted in average MAE and RMSE of 8.99 and 12.8, respectively. However, 'cross_val_score' resulted in an MAE and RMSE of 14.3 and 18.2, respectively. Model F, which had 171 datapoints, had even larger differences from an MAE of 8.75% from simulations and 20.0% for 'cross_val_score'.

Table 9: Differences between 100 simulations and 'cross_val_score' for MAE and RMSE in unimputed Models (E&F).

Model	100 Simulations		'Cross_Val_Score'	
	MAE	RMSE	MAE	RMSE
E	8.99	12.8	14.3	18.2
F	8.75	13.0	20.0	25.6

Similar discrepancies were seen in KNN imputed models as well. For example, the base run of Model B.1 had an MAE of 8.66 for simulations and an MAE of 14.9 from 'cross_val_score'. In addition, the other regressions base models suffered similar differences between the testing and validation which is highlighted in Table 10. While the differences vary between the models, generally the 'cross_val_score' result was much higher than the simulation result. An exception to this would be for Model C. However, the Model C.1 simulation MAE was already high at 11.5. Discrepancy for RMSE was similar and this can be seen in the results in Appendix B.

Table 10: Difference in MAE between base model runs of imputed models for 100 simulations vs 'cross_val_score'.

Model	100 Simulations MAE	Python 'Cross_val_score' MAE	Absolute Difference in MAE
A.1	8.3	13.3	5.0
B.1	8.7	14.9	6.2
C.1	11.5	14.3	2.8
D.1	8.4	15.7	7.3

Like improvements or deterioration in simulation performance, whether or not feature engineering improved the validation score largely depended upon the specific model. For example, the cutoffs for Model B.6 from 400 to 500 °C and 0 to 3.5 mm particle size with no 1000g feed size datapoints improved the 'cross_val_score' on MAE from 14.9 to 12.6 when compared to the base B.1 model. The difference between the simulation and the validation also decreased from a MAE of 6.2 to 4.5. While this was an improvement, the difference of 4.5 was still not ideal. Alternatively, Model D showed lesser difference in the validation score once the exact same constraints were added (Model D.3). From D.1 to D.3, the validation score from 'cross_val_score' improved from an MAE of 15.7 to 8.78. RMSE also improved from 18.3 to 12.6. In addition, the difference between the simulation and 'cross_val_score' was only 2.1 for MAE and 1.4 for RMSE. Table 11 displays the disparities between MAE and RMSE for each model with the associated cutoffs performed. While in general there was associated improvement, the larger improvement was seen from the two imputed models B and D. Furthermore, this indicated that whether the model's validation score improved or not based on feature engineering was model specific and not inherently tied to the constraints themselves.

Table 11: Absolute differences between simulations and 'cross_val_score' with MAE and RMSE. Note: models where cutoff from 400-500C, 0 to 3.5 mm, and no 1000g FS were used.

Model	Simulations MAE	'Cross_val_score' MAE	Absolute Difference in MAE	Simulations RMSE	'Cross_val_score' RMSE	Absolute Difference in RMSE
B.6	8.1	12.6	4.5	12.3	14.5	2.2
D.3	6.7	8.8	2.1	11.2	12.6	1.4
E.6	8.8	12.9	4.1	12.2	16.7	4.5
F.6	8.0	17.1	9.1	12	21	9.0

Python's 'Cross_val_score' on Classification Models

In addition to the regression models, the Python 'cross_val_score' was also tested for the classification models. In general, the difference between the simulations and cross validation score were far less in the imputed Model G than in the unimputed models H and I. The differences between the simulated accuracy and the 'cross_val_score' can be seen below for all models in Table 12.

Table 12: Difference between 100 Simulations and 'cross_val_score' for classification models in accuracy.

Model	Simulations	'Cross_val_score'	Difference in Accuracy %
G.1	87	82	5
G.2	85	78	7
G.3	79	72	7
H.1	46	29	17
H.2	60	38	22
I.1	89	69	20
I.2	85	51	34
I.3	84	67	17

As shown in Table 12, the difference between cross validation accuracy and simulation accuracy was better for Model G. Whereas, Model's H and I have similar large discrepancies that were seen in the regression models for the unimputed models. Model H performed poorly from the initial simulations and therefore it was not a surprise to see poor robustness from the model. From this, Model G appeared to be one of the more robust base models as compared to the other classification attempts. This showed that having more data and specifically imputed data, was more robust than not. From the simulations, however, Model I performed slightly better than Model G despite lower robustness from Python's 'cross_val_score'.

Stratified Cross Validation Sampling Approach

In response to high differences between 100 simulation runs and 'cross_val_score' validation, it was hypothesized that the imbalance in oil yields within the data set could be causing problems. Due to this, 'cross_val_score' could have been separating data into five folds where some contained representative samples and others contained imbalanced ones. For

example, if a fold did not contain any low yield points, then the testing fold could have resulted in a poor validation for that cross fold. The solution for this was development of Python code that performed validation with randomly shuffled folds from user divided data. This code can be found in Appendix G.

After running this method on several different regression models, it was clear that this validation method more accurately matched the data gathered by running 100 simulations. Table 13 below includes a comparison between the average MAE and RMSE for the models that were also cross validated with this method.

Table 13: Comparison between 100 simulations and the stratified cross validation method.

	100 Simulations		Stratified Cross Validation	
Model	MAE	RMSE	MAE	RMSE
A.1	8.3	12.5	8.8	12.9
B.1	8.7	13.2	9.0	13.7
D.3	6.7	11.2	7.2	11.6
E.1	9.0	12.8	7.1	11.1
F.6	8.0	11.8	7.3	11.1

Base models were run with the stratified method for a means of comparison to other base models. A comparison between the base models for the differences in simulation, 'cross_val_score', and stratified validation MAE and RMSE can be seen in Figures 29 and 30, respectively. All results for the stratified method can be seen in Appendix B.

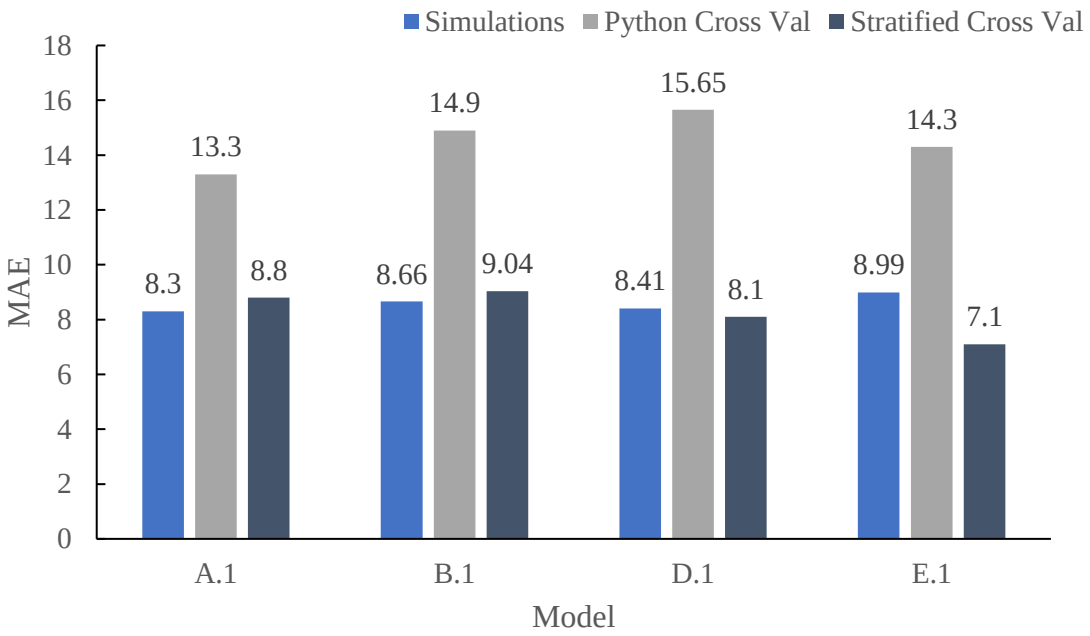

Figure 29: MAE for base models A.1, B,1, D.1, and E.1 for simulations, python 'cross_val_score', and the stratified cross validation method.

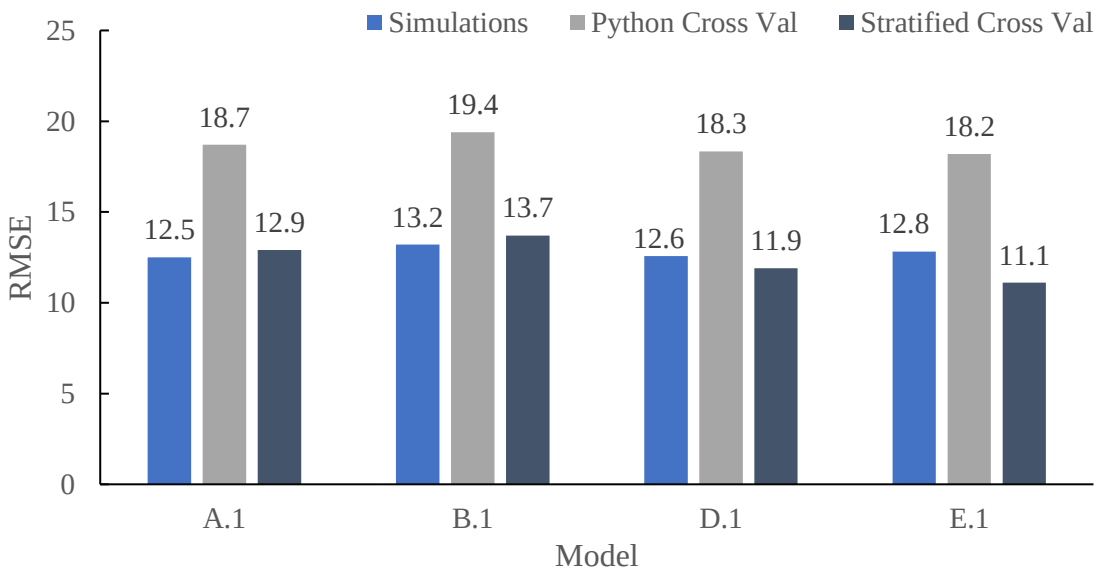

Figure 30: RMSE for base models A.1, B.1, D.1, and E.1 for simulations, python 'cross_val_score', and the stratified cross validation method.

Figures 29 and 30 display the clear difference in validation comparison to simulations between the 'cross_val_score' and the stratified method. In every case, the stratified method was closer and only deviated about 1% MAE difference for Model E.1. Altogether, this indicated that the stratified method was a better validation for all models and problems of this type where there are vast imbalances in data distribution.

In addition to running the regression models with the stratified cross validation approach, the classifier Models G and I were run as well using the exact same method as the regression problems. These were performed at the various yield cutoffs to determine if the stratified method carried through at each cutoff. The results of this can be see below in Figures 31 and 32 for Model G and I, respectively. Similarly, to the regression models, the stratified method more closely resembled the simulation results for Model G and I. Model G's Python 'cross_val_score' validations were not vastly different from the simulations, but the stratified method still resulted in closer accuracy. Model I showed the performance of the stratified cross validation method more significantly. While 'cross_val_score' predicted the validation accuracy of Model I nearly and over 20% in accuracy off from the simulations, the stratified method was closer and, in some cases, higher in accuracy than the simulation accuracy results. In addition, for both the imputed Model G and unimputed Model I, the stratified method of cross validation-maintained success over each classification cutoff. Overall, this showed that the stratified method was also applicable to the classification models and that the classification models were more robust than initially considered using only the Python 'cross_val_score'.

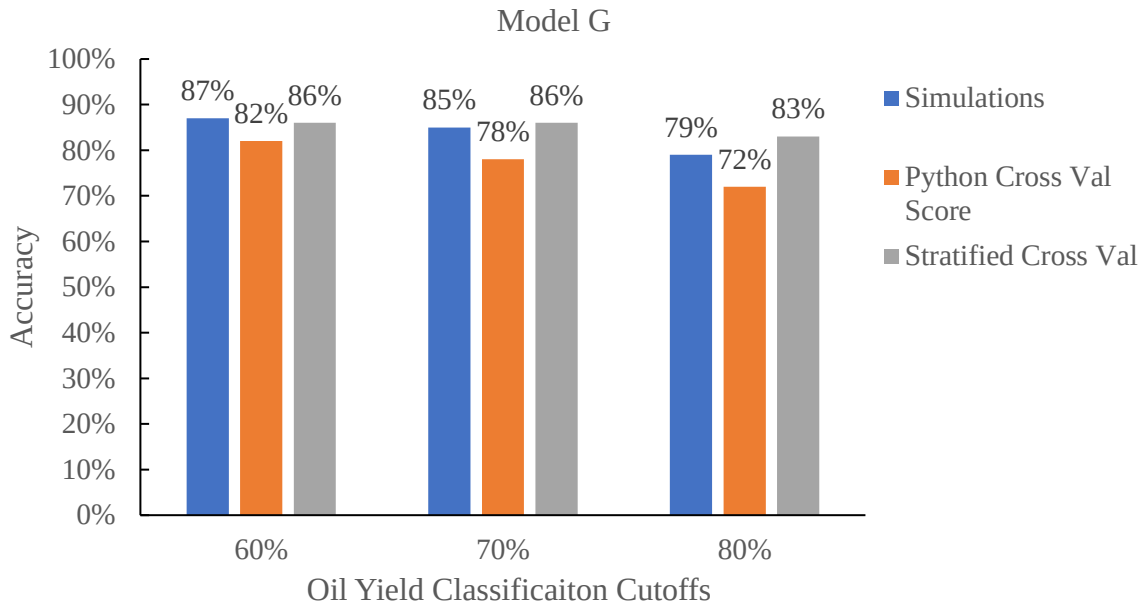

Figure 31: Model G classification model results for simulations, 'cross_val_score', and the stratified validation method.

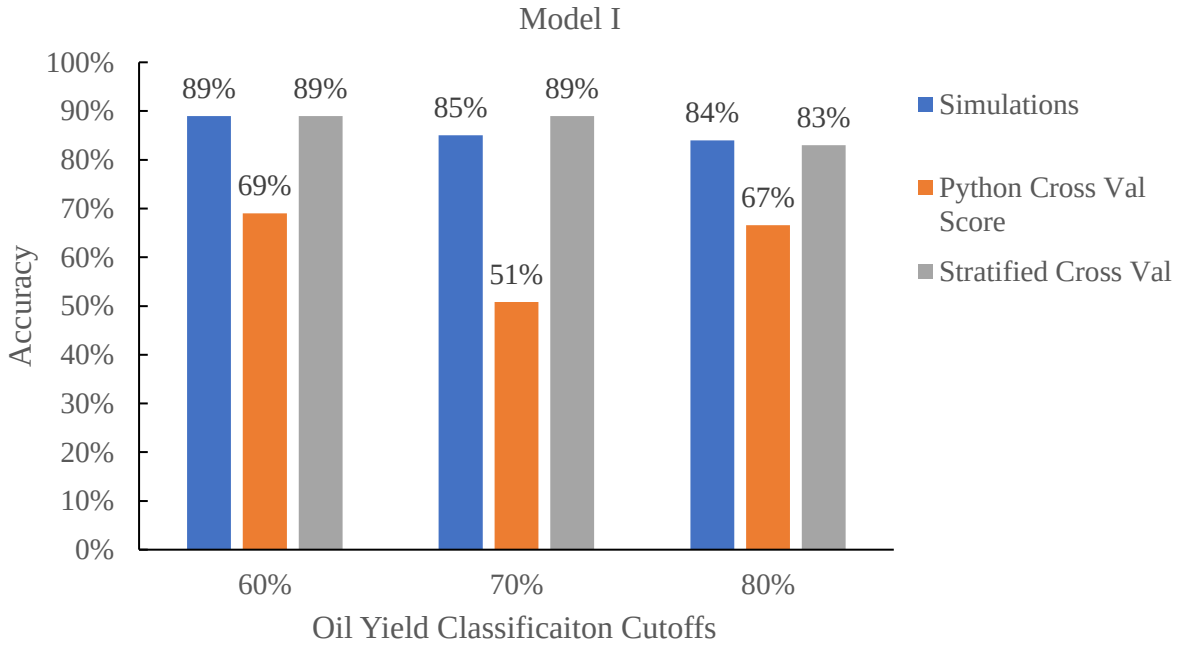

Figure 32: Model I classification model results for simulations, 'cross_val_score', and the stratified validation method.

Error Outlier Comparison

After running simulations, it was clear that certain datapoints were being predicted well whereas others were more temperamental. This was visualized by a parity plot shown in Figure 33 which displays the predicted vs. actual oil yields of the model run. In Figure 33, Model B.1 was simulated once to display such a plot. As can be seen there was an outlier point around 30 predicted yield and 80 actual yield.

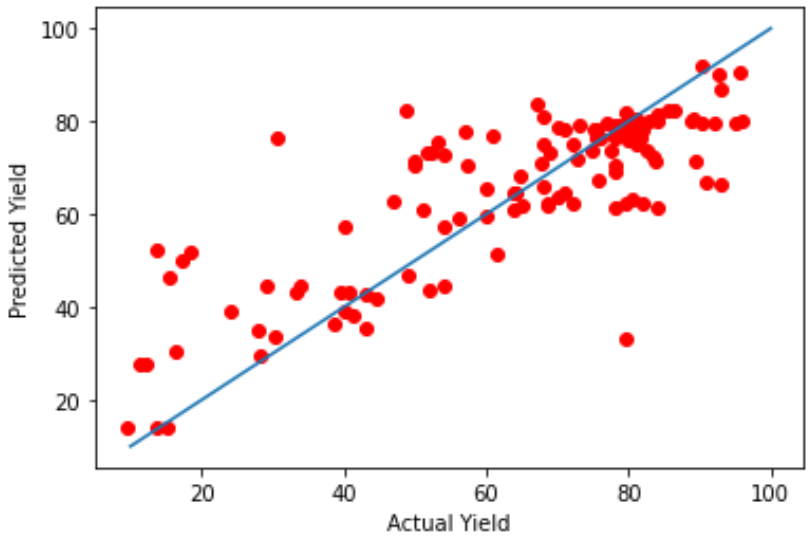

Figure 33: Model B.1 simulation to display outliers in model predictions.

These occurrences led to the development of code that would highlight datapoints that were predicted beyond a certain set range or error for example greater than 40% oil yield between the actual yield and predicted yield. This code can be found in Appendix H. In addition to this, Model B.1 simulation was also run to determine outlier errors associated with particular features. This can be seen in Figures 34 and 35 that display the outlier error with points at associated temperatures and reaction times. In Figure 34, high errors were seen where there were few data around 700 °C. However, there was also high error seen in the bulk of the data from 350 to 500 °C. In addition, high error spans from short to long reaction times as seen in Figure 35. From this, it can be stated that high errors were not simply due to a lack of coverage in data alone. Although, spread out data might be one reason for errors, there were several other issues that could be at play. Pyrolysis is complex, and the use of only hundreds of data points, spanning nearly a dozen of different variables, was unlikely to truly capture every facet of the process.

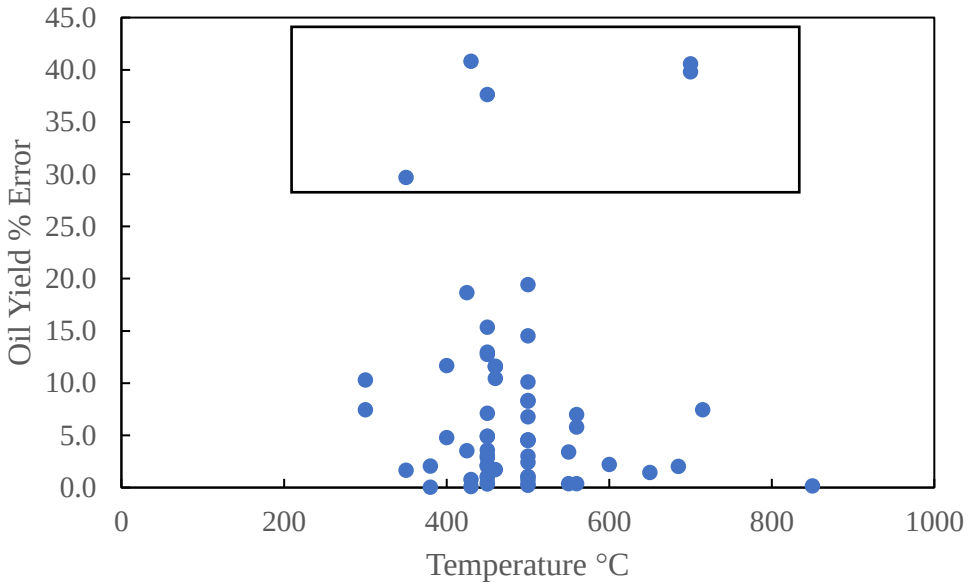

Figure 34: Model B.1 errors between predicted and actual oil yield for datapoints at associated temperatures.

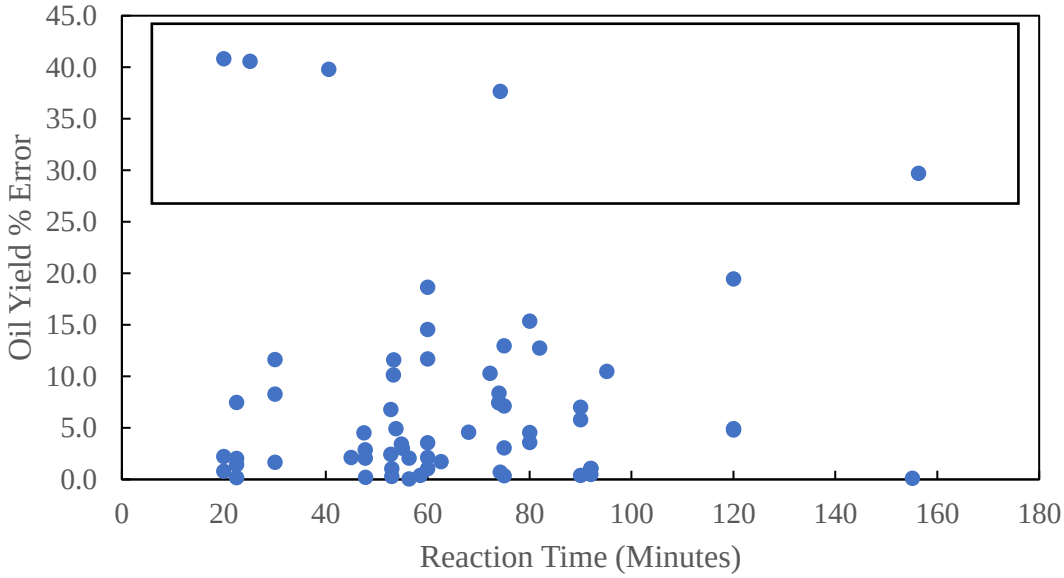

Figure 35: Model B.1 errors between predicted and actual oil yield for datapoints at associated reaction times.

To continue to investigate error reported by the model, many simulations were run on both Models A.1 and B.1 to compare the datapoints that resulted in high reoccurring predictive outliers for those models. Due to the method, data points were discerned via a concatenation of associated feature values and ending with an oil yield to create a unique label for each datapoint within the code. In this section, these have been simplified to a single number based on their location in the master dataset. Table 14 contains the 10 highest outlier error producing points for Model B.1 after

running 500 simulations. The count indicated the number of times that each point deviated greater than or equal to 40% oil yield from its actual yield. Initially, it was hypothesized that potentially that catalyst points could have caused the issues. However, Model B.1 showed that catalyst points did not significantly make up the datapoints causing the highest amount of error.

Table 14: Model B.1 500 simulations with a count of each time an associated datapoint resulted in greater than a 40% oil yield error.

Associated Cell	Oil Yield (%)	Sum of count	Catalyst	% of runs
27	24.0	109	no	21.8%
16	93.1	82	no	16.4%
46	79.2	73	no	14.6%
26	11.2	58	no	11.6%
8	28.5	44	no	8.8%
219	28.6	32	no	6.4%
235	18.3	28	yes	5.6%
17	84.7	23	no	4.6%
50	31.7	20	no	4.0%
167	37.5	19	no	3.8%

An interesting takeaway from these findings was that four data points made up around 64% of the errors above 40% oil yield for this model. Many of the datapoints in Table 14 were associated with lower oil yield ranging from 11.2 to 37.5% and two data points were roughly from the bulk of the data, 79.2% and 84.7%. Hence, there was a numerical indication on top of a graphical one that lower yields could be causing problems for the models. By further looking into the datapoints causing higher errors both datapoints associated with number 26 and 27 were from the same paper by Kumar & Singh 2011.[17] The paper reported oil yields of 11.2 and 24.0% at pyrolysis reaction temperatures of 400 and 450 °C on pure HDPE. It was possible that the long reaction times of 760 and 290 minutes respectively could be causing issues with model since there was not many other data within that range of reaction times. Aside from an interesting occurrence, there was no indication that these datapoints were incorrect. Furthermore, a lack of existing literature within certain feature ranges was not solely unique to this occurrence. Model A.1 was run as a comparison to see if datapoint outlier error was replicable across models. Model A.1 ran

1500 simulations and Model B.1 ran 2500 simulations and accounting for errors larger than 40 oil yield. Table 15 visualized the 10 most common error points for each model by the percentage of number of runs each point appeared in. Cells were highlighted in blue if both appeared in the top ten of each model. The cells that were not highlighted were not present in the Model A.1 dataset and therefore could have never been repeated. From this, it can be stated that issues in error points were not completely model specific and that certain data points were causing issues for all RF simulations.

Table 15: Model A.1 and B.1 comparison of errors of over 40% oil yield by % appearance and associated cell number. Blue cells indicate that associated data point was a common source of error for both models A.1 and B.1.

Model A.1		Model B.1	
Cell Associated	% Run Appearance	Cell Associated	% Run Appearance
26	33.1%	27	20%
27	30.8%	16	15%
128	18.8%	46	13%
167	6.8%	26	11%
271	5.7%	8	8%
132	1.4%	219	6%
109	0.7%	235	5%
139	0.4%	17	4%
140	0.4%	50	4%
99	0.3%	167	3%

Numerous new models were tested by removing data points that were the most inaccurate most often. Model D.3, one of the best performing regression models, was run with the error code with a threshold of greater than or equal to 20% oil yield in two iterations. The first time the top seven errors were removed and the next the top nine errors were removed. Model E.1 was also run with a similar method to visualize this on a unimputed model. For Model E.1, 18 points were removed over the course of two iterations for errors surpassing 30% oil yield. The classification Model I.2 was also run with 19 points removed over two iterations for the datapoints that were predicted wrong the most. Since Model I.2 was classification no direct oil yield percentage cutoff could be placed. The results of these models can be seen in Table 16 below for the classification model and Table 17 below for the regression models. Unsurprisingly, the classification improves from an accuracy on 85% to 93%, Model E.1 improved from 8.8 to 5.6 MAE, and Model D.3 improved from 6.7 to 4.2 MAE.

Due to the dataset being small, it would be nearly impossible to make good predictions for every value for the included dependent variables. The act of removing erroneous data, while perhaps losing out on some of the chemistry occurring, helps to hone and improve the model for values of the dependent variables that have more datapoints. Nevertheless, error visualization and removal showcased the large effect that certain data points had on model accuracy. Additionally,

it was important to note that the large effect of outlier removal was likely due to the small size and spread of the data set. Furthermore, being able to locate and observe the datapoints was helpful in learning more about how the algorithms interacted with the data. Further studies of plastic pyrolysis machine learning should be done, not to remove more erroneous data points, but to add more data points to the areas of the data that were consistently predicted incorrectly. The increase in accuracy of the error removal models justifies this point.

Table 16: Classification outlier removal on Model I.2.

Model	Accuracy %	Data Points
I.2	85	171
I.2 B	87	162
I.2 C	93	152

Table 17: Regression outlier removal on Models E.1 and D.3.

Model	MAE	RMSE	Data Points
D.3	6.7	11.2	114
D.3 B	5.3	7.4	107
D.3 C	4.3	5.9	98
E.1	8.8	13.0	132
E.1B	7.3	11.2	123
E.1C	5.6	7.7	114

Future Directions and Improvements

Altogether, the information and understanding gained by performing Random Forest algorithms on this pyrolysis dataset was a novel concept that revealed both successful, insightful, and promising results. However, many of the metrics reported in literature were often higher than those gained by this study. In the future, there are several ways that this work can be expanded upon and added to improve results and explore machine learning's application further. One of these is by testing other machine learning methods. At the suggestion and interpretation of past literature, this study focused primarily on Random Forest. However, there are many more options for machine learning problems. These include: Neural networks, XGBoost, Support Vector Machines, Naive Bayes, and K-means algorithms. While Random Forest showed promise, other algorithms could be attempted to observe and or recontextualize the results from the dataset. In addition, further hyperparameter tuning could also be employed. In this study, focus was placed on developing models using feature engineering such as the additions, cutoffs, imputation, and development of models such as the batch only Model D to observe changes in performance metrics. Further hyperparameter tuning of each model individually could also improve results.

Another element of future improvement could be addition and future manipulation of the base 310 dataset. Before this study, pyrolysis of plastics data had not been aggregated from literature data in such a way. Several ways the dataset could be added to could be by further

literature search as well as performing experiments directly. Performing experiments would allow for gaps in the dataset to be filled and provide more information to the model that was not previously known. Controlling factors in a lab space would also eliminate the occurrence of missing data not being included from literature sources. Furthermore, one hot encoded variables such as reactor type and catalyst could be expanded upon to include more information such as reactor geometry and size and catalyst type and acidity. Overall, the possibilities for expanding upon this work is vast and the promise machine learning shows for both pyrolysis and chemical engineering are exciting.

Conclusion

The machine learning methods instituted in this study of plastic pyrolysis are not solely limited to this chemical process. The piecewise addition of variables and variable constraint analyses may be used on any process. The same is true for the methods of cross validation. In this study, when it was found that a basic 5-fold cross validation failed to achieve similar results compared to the models, a stratified method was chosen instead which indicated the robustness of the random forest regressor and classifier models. The best regression model was found to be Model D.3 which achieved an MAE of 6.7. The best classifier models were found to have accuracies of 80-90%, with accuracy generally decreasing with increasing oil yield cutoffs. It was also found that outlier data points were a source of high error for many models, indicating the need to expand the data set for completeness of process variables. Additionally, a study of imputed models versus unimputed models was performed and it was found that there were only small differences in performance between the two, indicating the validity of using KNN on this pyrolysis dataset. The only models with significant differences between models with imputed versus unimputed data was for those testing different variable cutoffs. Also, within the random forest models, the importance of each variable was determined, and in the models, it was generally found that the reactor type and presence of a catalyst were not as important as the plastic feed composition. Further, plastic feed composition was found to be less important than variables such as temperature and feed size. While many of the models created were successful and exhibited MAE values of 8 or lower, the process of analyzing the dataset is of most use going forward. These methods may be extrapolated to different processes, or in the case of this pyrolysis data set, expanded to include more data in the future with more refined models.

References

1 Rashid Miandad, M. R., Mohammad A. Barakat, Asad S. Aburiazaiza, Hizbullah Khan, Iqbal M. I. Ismail, Jeya Dhavamani, Jabbar Gardy, Ali Hassanpour and Abdul-Sattar Nizami. Catalytic Pyrolysis of Plastic Waste: Moving Toward Pyrolysis Based Biorefineries. *Frontiers in Energy Research*, doi:https://doi.org/10.3389/fenrg.2019.00027 (2019).

2 R.Miandad, M. A. B., Asad S.Aburiazaiza, M.Rehan, A.S.Nizami. Catalytic pyrolysis of plastic waste: A review. *Process Safety and Environmental Protection* **102**, 822-838, doi:https://doi.org/10.1016/j.psep.2016.06.022 (2016).

3 Muhammad Saad Qureshi, A. O., Hanna Pihkola, Ivan Deviatkin, Anna Tenhunen, Juha Mannila, Hannu Minkkinen, Maija Pohjakallio, Jutta Laine-Ylijoki. Pyrolysis of plastic waste: Opportunities and challenges. *Journal of Analytical and Applied Pyrolysis* **152**, doi:https://doi.org/10.1016/j.jaap.2020.104804 (2020).

4 T. Faravelli, G. B., C. Scassa, M. Perego, S. Fabini, E. Ranzi, M. Dente. Gas product distribution from polyethylene pyrolysis. *Journal of Analytical and Applied Pyrolysis* **52**, doi:https://doi.org/10.1016/S0165-2370(99)00032-7 (1999).

5 Tiziano Faravelli, G. B., Mauro Colombo, Eliseo Ranzi, Mario Dente. Kinetic modeling of the thermal degradation of polyethylene and polystyrene mixtures. *Journal of Analytical and Applied Pyrolysis* **70**, 761-777, doi:https://doi.org/10.1016/S0165-2370(03)00058-5 (2003).

6 Elham Khaghanikavkani, M. M. F. Thermal Pyrolysis of Polyethylene: Kinetic Study. *CSCanada Energy Science and Technolog* **2**, 1-10, doi:10.3968/j.est.1923847920110201.597 (2011).

7 Blake R. Hough, D. A. C. B., Daniel T. Schwartz, Jim Pfaendtner. Application of machine learning to pyrolysis reaction networks: Reducing model solution time to enable process optimization. *Computers and Chemical Engineering* **104**, 56-63, doi:https://doi.org/10.1016/j.compchemeng.2017.04.012 (2017).

8 Qinghui Tang, Y. C., Haiping Yang, Ming Liu, Haoyu Xiao, Ziyue Wu, Hanping Chen, and Salman Raza Naqvi. Prediction of Bio-oil Yield and Hydrogen Contents Based on Machine Learning Method: Effect of Biomass Compositions and Pyrolysis Conditions. *Energy Fuels* **34**, 11050-11060, doi:https://doi.org/10.1021/acs.energyfuels.0c01893 (2020).

9 ShichaoZhang. Nearest neighbor selection for iteratively kNN imputation. *Journal of Systems and Software* **85**, 2541-2552, doi:https://doi.org/10.1016/j.jss.2012.05.073 (2012).

10 Rajat M.Thomas, W., PaulZhutovsky, Guidovan Wingen. Chapter 14 - Dealing with missing data, small sample sizes, and heterogeneity in machine learning studies of brain disorders. *Machine Learning Methods and Applications to Brain Disorders*, 249-266, doi:https://doi.org/10.1016/B978-0-12-815739-8.00014-6 (2020).

11 Cort J. Willmott, K. M. Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research* **30**, 79-82, doi:doi:10.3354/cr030079 (2005).

12 Xinzhe Zhu, Y. L., Xiaonan Wang. Machine learning prediction of biochar yield and carbon contents in biochar based on biomass characteristics and pyrolysis conditions. *Biosource Technology* **288**, doi:https://doi.org/10.1016/j.biortech.2019.121527 (2019).

13 Faisal Abnisa, S. D. A. S., Mohd Fauzi bin Zanil, Wan Mohd Ashri Wan Daud, Teuku Meurah Indra Mahlia. The Yield Prediction of Synthetic Fuel Production from Pyrolysis of Plastic Waste by Levenberg–Marquardt Approach in Feedforward Neural Networks Model. *Polymers*, doi:https://dx.doi.org/10.3390%2Fpolym11111853 (2019).

14 Carlos Ludlow-Palafox, H. A. C. Microwave-Induced Pyrolysis of Plastic Wastes. *Industrial & Engineering Chemistry Research* **40**, 4749-4756, doi:https://doi.org/10.1021/ie010202j (2001).

15 Kedar Potdar, T. P., Chinmay Pai. A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers. *International Journal of Computer Applications* **175**, doi:http://dx.doi.org/10.5120/ijca2017915495 (2017).

16 L. Quesada, A. P., V.Godoy, F.J.Peula, M.Calero, G. Blázquez. Optimization of the pyrolysis process of a plastic waste to obtain a liquid fuel using different mathematical models. *Energy Conversion and Management* **188**, 19-26, doi:https://doi.org/10.1016/j.enconman.2019.03.054 (2019).

17 Sachin Kumar, R. K. S. Recovery of hydrocarbon liquid from waste high density polyethylene by thermal pyrolysis. *Brazilian Journal of Chemical Engineering* **28**, doi:https://doi.org/10.1590/S0104-66322011000400011 (2011).

18 Pedregosa et al., *Scikit-learn: Machine Learning in Python*, <https://scikit-learn.org/stable/about.html#citing-scikit-learn> (2011).

19 Nadkarni, P. Chapter 10 - Core Technologies: Data Mining and "Big Data". *Clinical Research Computing A Practitioner's Handbook* 187-204, doi:https://doi.org/10.1016/B978-0-12-803130-8.00010-5 (2016).

Appendices

Appendix A: Literature Papers used for Data

Table 18: Citations for literature papers used for data within this study.

Dataset References
Bagri, R., & Williams, P. (2002). Catalytic pyrolysis of polyethylene. *Journal of Analytical and Applied Pyrolysis, 63*(1), 29-41. https://doi.org/10.1016/S0165-2370(01)00139-5
Zhao, D., Wang, X., Miller, J., & Huber, G. (2020). The Chemistry and Kinetics of Polyethylene Pyrolysis: A Process to Produce Fuels and Chemicals. *ChemSusChem, 13*(7), 1764-1774. https://doi.org/10.1002/cssc.201903434
Al-Salem, S. (2019). Thermal pyrolysis of high density polyethylene (HDPE) in a novel fixed bed reactor system for the production of high value gasoline range hydrocarbons (HC). *Process Saftey and Enviromental Protection, 127*, 171-179. https://doi.org/10.1016/j.psep.2019.05.008
Marcilla, A., Beltrán, M., & Navarro, R. (2009). Thermal and catalytic pyrolysis of polyethylene over HZSM5 and HUSY zeolites in a batch reactor under dynamic conditions. *Applied Catalysis B: Environmental*, *86*(1-2), 78-86. doi: 10.1016/j.apcatb.2008.07.026
F. Pinto., P. Costa., I. Gulyurtlu., I. Cabrita. (1999). Pyrolysis of plastic wastes. Effect of plastic waste composition on product yield, *Journal of Analytical and Applied Pyrolysis*, *51*(1-2), 39-55, ISSN 0165-2370,https://doi.org/10.1016/S0165-2370(99)00007-8.
Seo, Y., Lee, K., & Shin, D. (2003). Investigation of catalytic degradation of high-density polyethylene by hydrocarbon group type analysis. *Journal of Analytical and Applied Pyrolysis*, *70*(2), 383-398. https://doi.org/10.1016/S0165-2370(02)00186-9
Kumar S. & Singh, R. (2011). Recovery of Hydrocarbon Liquid from Waste High Density Polyethylene by Thermal Pyrolysis. *Brazilian Journal of Chemical Engineering*, *28*(4), 659- 667. https://www.scielo.br/pdf/bjce/v28n4/a11v28n4.pdf
J.M. Encinar & J.F. González. (2008). Pyrolysis of synthetic polymers and plastic wastes. *Fuel Processing Technology*, *89*(7), 678-686, ISSN 0378-3820, https://doi.org/10.1016/j.fuproc.2007.12.011.
Williams, E. A., & Williams, P. T. (1997). The pyrolysis of individual plastics and a plastic mixture in a fixed bed reactor. *Journal of Chemical Technology & Biotechnology: International Research in Process, Environmental and Clean Technology*, *70*(1), 9-20.
Ludlow-Palafox, C., & Chase, H. (2001). Microwave-Induced Pyrolysis of Plastic Wastes. *Industrial & Engineering Chemistry Research, 40*, 4749-4756.
Quesada, L., Pérez, A., Godoy, V., Peula, F., Calero, M., & Blázquez, G. (2019). Optimization of the pyrolysis process of a plastic waste to obtain a liquid fuel using different mathematical models. *Energy Conversion And Management, 188,* 19-26. doi: 10.1016/j.enconman.2019.03.054
Singh, R., Ruj, B., Sadhukhan, A., & Gupta, P. (2019). Impact of fast and slow pyrolysis on the degradation of mixed plastic waste: Product yield analysis and their characterization. *Journal Of The Energy Institute*, 92(6), 1647-1657. doi: 10.1016/j.joei.2019.01.009
Kumar, S., & Singh, R. (2011). Recovery of hydrocarbon liquid from waste high density polyethylene by thermal pyrolysis. *Brazilian Journal Of Chemical Engineering*, *28*(4), 659-667. doi: 10.1590/s0104-66322011000400011
Ahmad, I., Khan, M., Khan, H., Ishaq, M., Tariq, R., Gul, K., & Ahmad, W. (2014). Pyrolysis Study of Polypropylene and Polyethylene Into Premium Oil Products. *International Journal Of Green Energy*, *12*(7), 663-671. doi: 10.1080/15435075.2014.880146
Anene, A., Fredriksen, S., Sætre, K., & Tokheim, L. (2018). Experimental Study of Thermal and Catalytic Pyrolysis of Plastic Waste Components. Dept of Process, Energy and Enviormnetla Technology, University of South-Eastern Norway. DOI:10.3390/su10113979
FakhrHoseini, S. & Dastanian. (2013). Predicting Pyrolysis Products of PE, PP, and PET Using NRTL Activity Coefficient Model. *Journal of Chemistry, Article ID 487878.* https://doi.org/10.1155/2013/487676
Absina, F., Sharuddin, S., Zanil, M., Daud, W., & Mahlia, T. (2019). The Yield Prediction of Synthetic Fuel Production from Pyrolysis of Plastic Waste by Levenberg–Marquardt Approach in Feedforward Neural Networks Model. *Polymers (Basel)*, *11*(11), 1853. doi: 10.3390/polym11111853
Uddin, A., Koizumi, K., Murata, K., & Sakata, Y. (1996). Thermal and catalytic degradation of structurally different types of polyethylene into fuel oil. Polymer Degredation and Stabliltiy, 56(1), 37-44

Miandad, R., Barakat, M, Aburiazaiza, A., Rehan, M., Ismail, I., Nizami, A. (2017). Effect of plastic waste types on pyrolysis liquid oil. *International Biodeterioration & Biodegradation*, *119*, 239-252. https://doi.org/10.1016/j.ibiod.2016.09.017

Bajus, M. & Hájeková, E. (2010). Thermal Cracking of the Model Seven Components Mixed Plastics into Oils/Waxes. Slovak University of Technology.Retrieved from https://www.researchgate.net/profile/Elena_Hajekova/publication/47394502_THERMAL_CRACKING_OF_THE_MODEL_SEVEN_COMPONENTS_MIXED_PLASTICS_INTO_OILSWAXES/links/00b7d52e

Bajus, M. & Hájeková, E. (2008). Copyrolysis of oils/waxes of individual and mixed polyalkenes cracking products with petroleum fraction. *Fuel Processing Technology*, *89*(11), 1047-1055. https://doi.org/10.1016/j.fuproc.2008.04.007

Wiiliams, P. & Slaney, E. (2007). Analysis of products from the pyrolysis and liquefaction of single plastics and waste plastic mixtures. Resources, *Conservation and Recycling*, *51*(4), 754, 769. https://doi.org/10.1016/j.resconrec.2006.12.002

Muhammad, C., Onwudili, J., Williams, P. (2015). Thermal Degradation of Real-World Waste Plastics and Simulated Mixed Plastics in a Two-Stage Pyrolysis–Catalysis Reactor for Fuel Production. *Energy Fuels*, *29*(4), 2601-2609. https://doi.org/10.1021/ef502749h

Singh, R. & Ruj, B. (2016). Time and temperature depended fuel gas generation from pyrolysis of real world municipal plastic waste. *Fuel*, *174*(15), 163-171. https://doi.org/10.1016/j.fuel.2016.01.049

Mastral, F., Esperanza, E., García, P., & Juste, M. (2002). Pyrolysis of high-density polyethylene in a fluidised bed reactor. Influence of the temperature and residence time. J*ournal of Analytical and Applies Pyrolysis*, *63*(1), 1-15. https://doi.org/10.1016/S0165-2370(01)00137-1

Liu, Y., Qian, Y., Wang, Y. (2000). Pyrolysis of polystyrene waste in a fluidized-bed reactor to obtain styrene monomer and gasoline fraction. *Fuel Processing Technology*, *63*(1), 45-55.https://doi.org/10.1016/S0378-3820(99)00066-1

Onwudili, J., Insura, N., & Williams, P. (2009). Composition of products from the pyrolysis of polyethylene and polystyrene in a closed batch reactor: Effects of temperature and residence time. *Journal of Analytical and Applies Pyrolysis*, *86(*2), 293-303. https://www.sciencedirect.com/science/article/pii/S0165237009001119

Sakata, Y., Uddin, A., & Muto, I. (1999). Degradation of polyethylene and polypropylene into fuel oil by using solid acid and non-acid catalysts. J*ournal of Analytical and Applies Pyrolysis*, *51*(1-2), 135-55. https://doi.org/10.1016/S0165-2370(99)00013-3

Martynis, M., Mulyazmi, M., Winada, E., & Harahap, A. (2019). Thermal Pyrolysis of Polypropylene Plastic Waste into Liquid Fuel: Reactor Performance Evaluation. IOP Confrence: *Materials Science and Engineering*, *543*. doi:10.1088/1757-899X/543/1/012047

Papuga, S., Gvero, P., & Vukić, L. (2015). Temperature and time influence on the waste plastics pyrolysis in the fixed bed reactor. *Thermal Science*, *20*, 154. DOI: 10.2298/TSCI141113154P

López, A. Marco, I., Caballero, B., Laresgoiti, M., & Adrados, A. (2011). Influence of time and temperature on pyrolysis of plastic wastes in a semi-batch reactor. *Chemical Engineering Journa*l, *173*(1), 62-71. https://doi.org/10.1016/j.cej.2011.07.037

Williams, P., & Williams, E. (1999). Fluidised bed pyrolysis of low density polyethylene to produce petrochemical feedstock. *Journal Of Analytical And Applied Pyrolysis*, *51*(1-2), 107-126. doi: 10.1016/s0165-2370(99)00011-x

Mastral, F., Esperanza, E., Berrueco, C., Juste, M., & Ceamanos, J. (2003). Fluidized bed thermal degradation products of HDPE in an inert atmosphere and in air–nitrogen mixtures. *Journal Of Analytical And Applied Pyrolysis*, *70*(1), 1-17. doi: 10.1016/s0165-2370(02)00068-2

Ajibola, A., Omoleye, J., & Efeovbokhan, V. (2018). Catalytic cracking of polyethylene plastic waste using synthesised zeolite Y from Nigerian kaolin deposit. *Applied Petrochemical Research*, *8*, 211-217. https://doi.org/10.1007/s13203-018-0216-7

Jadhao, S. & Seethamraju, S. (2020). Pyrolysis Study of mixed plastics waste. IOP Conf. Ser.: *Mater. Sci. Eng.*, *736*. Retrived from https://iopscience.iop.org/article/10.1088/1757-899X/736/4/042036/pdf

Panda, A., Alotaibi, A., Kozhevnikov, I., & Shiju, N. (2020). Pyrolysis of Plastics to Liquid Fuel Using Sulphated Zirconium Hydroxide Catalyst. *Waste and Biomass Valorization*, *11*, 6337-6345. Retrived from https://link.springer.com/article/10.1007/s12649-019-00841-4

Sembiring et al. (2018). Catalytic Pyrolysis of Waste Plastic Mixture. IOP Conf. Ser.: *Mater. Sci. Eng.*, *316*. Retrived from https://iopscience.iop.org/article/10.1088/1757-899X/316/1/012020/pdf

Appendix B: All Model Results

For the model results, these have been split up between regression results and classification results. In addition, the piecewise results were separated from other model results due to the different naming system. For trials were the imputed regression results, two included for each model run. The stratified cross validation method results were separated as well independently from the simulations and python 'cross_val_score' since only a few models were run with the stratified method. This was due to time constraints and the use of the stratified method as a proof of concept where not all models needed to be run to prove efficacy.

Piecewise Model Regression Results

Table 19: Piecewise regression results.

	100 Simulations				Python 'Cross_val_score'				
Model	**MAE**	**Std Dev**	**RMSE**	**Std Dev**	**MAE Avg**	**Std Dev**	**RMSE Avg**	**Std Dev**	**Data Points**
Compositions	13.9	1.3	18	1.6	21.5	10.4	24.5	10.3	310
Compositions with Temp	11.3	1.3	16.3	1.7	19.82	9.76	23.22	9.5	310
Only Temp	16.4	1.2	20.5	1.4	22.7	12.1	24.4	11.7	310
Comps, Temp, & Rxn Time	9.7	1.4	13.8	2.4	21.4	7.6	25.9	8.8	182
Comps, Temp, & Catalyst	11.4	1.4	16.3	1.9	19.6	9.9	23	9.6	310
Comps, Temp, & R type	10.5	1.3	15.1	1.9	18.3	8.8	22	8.2	310
Comps, Temp, & Catalyst, & R type	10.3	1.2	14.8	1.9	18.3	8.7	21.3	8.4	310

Regression Results for Models with Imputation

Table 20: Imputed regression model results.

	100 Simulations				Python 'Cross_val_score'				
Model	MAE	Std Dev	RMSE	Std Dev	MAE	Std Dev	RMSE	Std Dev	Data Points
A.1	8.3	1.6	12.5	2.8	13.3	4.2	18.7	5.4	182
	8.11	1.6	12.1	2.7					
A.2	7.5	1.4	11	2.2	15.2	8.1	19.6	7.4	176
	7.3	1.1	10.7	1.9					
A.3	8.4	1.6	12.1	2.4	12.9	4.3	17.4	4.8	160
	8.4	1.5	11.9	2.3					
A.4	6.9	1.3	9.9	1.7	10.6	3.3	13.6	4	139
	6.9	1.4	9.7	2					
A.5	4.8	0.9	6.4	1.1	7.3	2.3	8.8	2.1	110
	4.6	1	6.1	1.2					
B.1	8.66	1.1	13.2	1.8	14.9	3.6	19.4	4.5	310
	8.65	1.3	12.9	1.9					
B.2	8.41	0.93	13.06	1.6	17.3	4.2	22.6	5	310
	8.38	1.27	12.8	2.17					
B.3	8.43	1.2	12.9	1.9	17.1	4.3	22.3	5.1	265
	8.69	1.3	13.42	2					
B.4	4.8	0.7	6.7	1	7.3	1.8	8.8	1.8	201
	4.6	0.6	6.3	0.8					
B.5	8.45	1.46	12.32	2.3	15.8	4.8	20.09	5.93	198
	8.51	1.19	12.5	1.97					
B.6	8.14	1.73	12.32	2.9	12.64	5.26	14.47	5.17	151
	7.83	1.78	11.8	2.9					
C.1	11.5	1.4	15.6	2	14.25	1.23	19	1.8	199
	11.4	1.5	15.4	2					
D.1	8.41	1.44	12.58	2.42	15.65	9.8	18.34	10.54	200
	8.54	1.41	12.7	2.14					
D.2	8.5	1.5	12.65	2.47	11.1	4.34	14.24	5.18	152
	7.99	1.65	11.9	2.7					
D.3	6.72	1.76	11.16	3.59	8.78	1.82	12.63	3.77	114
	6.84	1.75	10.96	3.24					
D.4	6.86	1.6	10.72	2.93	8.83	2.65	13.15	4.88	114
	6.65	1.56	10.77	2.95					
D.5	7.55	1.84	11.85	3.34	9.26	2.71	13.17	5.12	114
	7.13	1.8	11.37	3.48					
D.6	7.08	1.81	11.31	3.74	8.86	2.06	12.68	3.82	114
	7.42	2.02	11.86	3.91					
D.7	7.36	1.83	11.65	3.23	9.11	1.7	13.1	3.45	114

	7	1.6	11.02	3.04					

Regression Results for Models without Imputation

Table 21: Regression results from unimputed Models F and E.

	100 Simulations				Python 'Cross_val_score'				
Model	MAE	Std Dev	RMSE	Std Dev	MAE	Std Dev	RMSE	Std Dev	Data Points
F.1	8.75	1.49	12.97	2.11	20.04	5.19	25.62	4.87	310
F.2	8.94	1.4	13.42	2.16	20.62	5.57	25.78	5.37	310
F.3	9	1.39	12.85	2	17.38	5.23	21.29	5.06	310
F.4	9.49	1.44	13.65	2.03	17.52	5.54	21.44	5.16	310
F.5	8.51	1.68	12.03	2.3	21.1	10	25	11	124
F.6	7.95	1.86	11.8	2.6	17.1	4.6	21.1	5.4	99
E.1	8.99	1.62	12.82	2.46	14.3	3.21	18.2	4.2	132
E.2	8.82	1.64	12.77	2.44	14.3	3.6	18.2	4.6	132
E.3	9.36	1.57	13.14	2.2	17.44	5.2	21	5.7	132
E.4	9.83	1.65	13.74	2.2	18.8	5.4	21.8	5.9	132
E.5	8.82	1.94	12.16	2.7	13.9	7	17.3	7.7	94
E.6	8.77	2.37	12.21	3.2	12.9	7.2	16.7	9.1	89
E.7	12.42	1.8	16.4	2.6	16.9	6.6	20.6	7.6	132

Regression Stratified Cross Validation Results

Table 22: Stratified cross validation results for regression models.

	100 Simulations				**Stratified Cross Validation**				
Model	MAE	Std Dev	RMSE	Std Dev	MAE	Std Dev	RMSE	Std Dev	Data Points
B.1	8.66	1.1	13.2	1.8	9.04	0.7	13.7	1.8	310
D.1	8.41	1.44	12.58	2.42	8.1	0.7	11.9	2.4	200
D.3	6.72	1.76	11.16	3.59	7.2	1.9	11.6	3.8	114
E.1	8.99	1.62	12.82	2.46	7.1	0.8	11.1	0.9	132
E.5	8.82	1.94	12.16	2.7	9.3	2	12.5	3	94
F.5	8.51	1.68	12.03	2.3	9.3	1.1	12.9	2	124
F.6	7.95	1.86	11.8	2.6	7.33	1.3	11.1	2.1	99

Piecewise Classification Results

Table 23: Piecewise classification results. Note: python cross validation was not run on most these models because they were solely created to compare simulation results.

	100 Simulations			Python 'Cross_val_score'		
Model	**Classification Cutoff (%)**	**Accuracy**	**Std Dev**	**Accuracy**	**Std Dev**	**Datapoints**
Compositions	60	77%	5	73%	11	310
Temperature	60	69%	5			310
Compositions & Temperature	60	83%	5	74%	11	310
Comps, Temp, & Catalyst	60	85%	5			310
Comps, Temp, & R Type	60	85%	5			310
Comps, Temperature, Catalyst, & R Type	60	88%	5			310
Comps, Temperature, Rxn Time	60	87%	5			182
Comps, Temperature, Catalyst, & R Type	70	77%	5			310
Comps, Temperature, Catalyst, & R Type	80	76%	5			310
Compositions	80	71	5	69	3	310
Compositions & Temperature	80	73	6	65	5	310

Classification Model Results

Table 24: Classification Model Results for models I and G.

		100 Simulations		Python 'Cross_val_score'		
Model	**Classification Cutoff**	**Accuracy**	**St Dev**	**Accuracy**	**Std Dev**	**Data Points**
I.1	60%	89%	5%	69%	11%	171
I.2	70%	85%	5%	51%	17%	171
I.3	80%	84%	6%	67%	14%	171
G.1	60%	87%	3%	82%	10%	310
G.2	70%	85%	5%	78%	14%	310
G.3	80%	79%	4%	72%	12%	310

Classification Feature Cutoff Model Results

Table 25: Feature cutoff results for classification models.

Model	Classification Cutoff	Feature Cutoffs	100 Simulations Accuracy	Std Dev	Data Points
I.4	60%	None	89	5	171
I.5	60%	400-500°C	96	5	124
I.6	60%	400-500°C & 0 to 3.5mm	71	10	99
I.7	70%	None	85	5	171
I.8	70%	400-500°C	89	6	124
I.9	70%	400-500°C & 0 to 3.5mm	91	5	99
I.10	80%	None	84	6	171
I.11	80%	400-500°C	81	8	124
I.12	80%	400-500°C & 0 to 3.5mm	79	8	99
G.4	60%	None	87	3	310
G.5	60%	400-500°C	92	4	199
G.6	60%	400-500°C & 0 to 3.5mm	94	4	152
G.7	70%	None	85	5	310
G.8	70%	400-500°C	85	6	199
G.9	70%	400-500°C & 0 to 3.5mm	86	6	152
G.10	80%	None	79	4	310
G.11	80%	400-500°C	75	6	199
G.12	80%	400-500°C & 0 to 3.5mm	73	8	152

Classification Stratified Cross Validation Results

Table 26: Classification results for stratified cross validation

		100 Simulations		Stratified Cross Validation		
Model	**Classification Cutoff**	**Accuracy**	**St Dev**	**Accuracy**	**Std Dev**	**Data Points**
I.1	60%	89%	5%	89%	3%	171
I.2	70%	85%	5%	89%	3%	171
I.3	80%	84%	6%	83%	8%	171
G.1	60%	87%	3%	86%	3%	310
G.2	70%	85%	5%	86%	6%	310
G.3	80%	79%	4%	83%	4%	310

Appendix C: Stratified Cross Validation Method Example

Below is an example of the dataset splits for preprocessing before being input in the stratified validation code for Model D.1. Model D.1 contained a total of 200 datapoints which were split up into 5 prep sheets with 40 datapoints each. Each prep sheet therefore represented 20% of the oil yields within the model. For example, sheet 1 contained 20% of the highest oil yields which ranged from 83% to 99% oil yield. Each sheet would be input into the python code found in Appendix G and then shuffled and sorted into the five cross validation bins. The splits of this data convey the oil yield spread disparity with excellent clarity. The first three splits were all over 70% oil yield.

Table 27: Example stratified cross val example from Model D.1

Split 1: 40 Datapoints 83 to 99% yield												
HDPE	LDPE	PP	PS	PVC	PET	Temperature	Heating Rate	Particle Size	Feed Size	Reaction Time	Catalyst	Oil Yield
0	0	0	100	0	0	350	10	2	10	60	0	99
0	70	0	30	0	0	400	10	2	10	60	0	96
0	0	0	100	0	0	450	10	2	10	58	0	96
0	0	0	100	0	0	450	5	2	10	53	0	96
0	0	0	100	0	0	430	10	3	474	20	0	95
0	100	0	0	0	0	460	10	2	10	101	0	95
0	0	100	0	0	0	500	5	4	35	60	0	95
0	100	0	0	0	0	400	5	1	1	117	0	93
0	68	16	16	0	0	430	11	3	22	20	0	93
100	0	0	0	0	0	500	5	4	35	60	0	93
0	0	0	100	0	0	450	15	2	10	58	0	93
0	0	0	100	0	0	450	20	2	10	53	0	93
0	16	16	68	0	0	430	10	3	538	20	0	92
0	0	100	0	0	0	430	6	3	9	20	0	92
0	33	33	33	0	0	430	11	3	419	20	0	91
0	70	0	30	0	0	425	10	2	10	60	0	90

0	16	68	16	0	0	430	6	3	75	20	0	90
0	100	0	0	0	0	430	10	3	10	20	0	90
0	100	0	0	0	0	425	10	2	10	60	0	90
0	0	100	0	0	0	400	10	50	1000	60	0	89
0	0	100	0	0	0	380	3	2	10	61	1	86
0	0	100	0	0	0	460	14	2	10	56	0	86
0	0	100	0	0	0	380	3	2	10	61	1	85
0	0	100	0	0	0	350	10	50	1000	60	0	85
0	0	100	0	0	0	450	8	2	17	92	0	85
0	0	100	0	0	0	500	20	3	20	52	1	85
100	0	0	0	0	0	400	5	1	1	169	0	85
100	0	0	0	0	0	430	3	1	10	350	0	85
0	100	0	0	0	0	430	3	1	10	300	0	85
0	100	0	0	0	0	430	3	1	10	300	0	84
100	0	0	0	0	0	450	7	7	16	30	0	84
0	34	66	0	0	0	460	13	2	10	62	0	84
29	29	27	9	0	6	600	20	20	200	60	0	84
0	0	100	0	0	0	500	20	3	20	55	1	84
50	50	0	0	0	0	430	3	2	10	375	0	84
0	70	0	30	0	0	450	10	2	10	60	0	84
0	0	100	0	0	0	380	3	2	10	61	1	84
0	0	100	0	0	0	450	5	2	10	56	0	83
0	0	100	0	0	0	380	3	2	10	61	1	83
Split 2: 40 Datapoints 80 to 83% yield												
0	0	100	0	0	0	500	20	3	20	56	0	83
0	0	100	0	0	0	500	20	3	20	48	1	83
0	0	100	0	0	0	450	15	2	10	56	0	83
0	0	100	0	0	0	450	10	2	10	56	0	83
33	33	33	0	0	0	450	8	2	17	92	0	82
0	0	100	0	0	0	500	6	2	147	80	0	82
29	29	27	9	0	6	500	20	6	54	30	0	82
29	29	27	9	0	6	550	20	20	200	60	0	82
0	100	0	0	0	0	500	20	3	20	74	1	82
100	0	0	0	0	0	430	3	3	10	168	1	82
0	100	0	0	0	0	450	5	2	10	97	0	82
100	0	0	0	0	0	430	3	3	10	168	1	81
0	100	0	0	0	0	450	10	2	10	97	0	81
0	0	100	0	0	0	500	8	2	147	80	0	81
100	0	0	0	0	0	500	14	3	50	69	0	81
10	0	0	90	0	0	560	10	4	23	90	1	81
90	0	0	10	0	0	560	10	4	23	90	1	81
0	0	100	0	0	0	500	10	25	1000	120	0	81
100	0	0	0	0	0	350	5	2	2	169	0	81
0	0	0	100	0	0	450	10	17	1000	75	0	81
0	0	100	0	0	0	500	10	2	147	80	0	81
0	0	100	0	0	0	380	3	2	10	61	1	81
29	29	27	9	0	6	500	20	20	200	60	0	81
0	100	0	0	0	0	500	6	2	14	106	0	80
0	100	0	0	0	0	450	14	2	17	80	1	80
0	0	100	0	0	0	400	10	50	1000	30	0	80
0	0	100	0	0	0	380	3	2	10	61	0	80
100	0	0	0	0	0	430	3	3	10	168	1	80
0	10	90	0	0	0	560	10	4	23	90	1	80
70	0	0	30	0	0	560	10	4	23	90	1	80
100	0	0	0	0	0	500	20	3	20	64	1	80
33	33	33	0	0	0	500	20	3	20	66	1	80
0	100	0	0	0	0	450	8	2	17	92	0	80
100	0	0	0	0	0	450	8	3	17	92	0	80
25	50	25	0	0	0	450	8	2	17	92	0	80
25	50	25	0	0	0	450	12	2	17	80	0	80
0	0	100	0	0	0	350	10	50	1000	30	0	80
0	100	0	0	0	0	500	8	2	14	92	0	80
0	0	100	0	0	0	450	10	25	1000	120	0	80
Split 3: 40 Datapoints 72 to 79% yield												
0	50	50	0	0	0	450	8	2	17	92	0	79
0	0	0	100	0	0	450	10	2	10	60	0	79
100	0	0	0	0	0	600	13	3	50	75	0	79
0	100	0	0	0	0	450	8	2	17	92	0	79
0	100	0	0	0	0	450	14	2	17	80	0	79
100	0	0	0	0	0	550	20	3	20	54	0	79
100	0	0	0	0	0	500	20	3	20	69	1	79
100	0	0	0	0	0	500	20	3	20	60	1	79
33	33	33	0	0	0	500	20	3	20	73	1	79
0	0	100	0	0	0	400	10	25	1000	120	1	78
0	0	100	0	0	0	380	3	2	10	61	1	78
0	0	100	0	0	0	500	14	2	147	80	0	78
29	29	27	9	0	6	450	20	20	200	60	0	78
0	30	70	0	0	0	560	10	4	23	90	1	78
50	50	0	0	0	0	560	10	4	23	90	1	78
0	100	0	0	0	0	500	20	3	20	80	1	78
100	0	0	0	0	0	500	20	3	20	73	0	78

33	33	33	0	0	0	500	20	3	20	58	1	78
0	0	100	0	0	0	400	10	25	1000	120	0	78
100	0	0	0	0	0	450	8	3	17	80	1	77
0	0	100	0	0	0	500	20	3	20	44	1	77
33	33	33	0	0	0	500	20	3	20	74	0	77
0	100	0	0	0	0	500	10	2	14	78	0	76
0	100	0	0	0	0	500	20	3	20	83	0	76
33	33	33	0	0	0	500	20	3	20	48	1	76
29	29	27	9	0	6	500	10	7	54	30	0	76
0	0	100	0	0	0	450	10	25	1000	120	1	76
35	35	10	10	11	1	450	8	2	17	92	0	75
0	0	100	0	0	0	380	3	2	10	61	1	75
100	0	0	0	0	0	430	3	3	10	168	1	75
30	70	0	0	0	0	560	10	4	23	90	1	75
70	30	0	0	0	0	560	10	4	23	90	1	75
0	100	0	0	0	0	500	20	3	20	70	1	75
0	100	0	0	0	0	500	12	2	13	78	0	74
100	0	0	0	0	0	430	3	3	10	168	1	74
100	0	0	0	0	0	500	20	3	20	52	1	73
0	0	90	0	0	10	400	10	25	1000	120	1	73
0	100	0	0	0	0	450	15	2	10	101	0	73
100	0	0	0	0	0	500	20	3	20	68	0	72
Split 4: 40 Datapoints 52 to 71% yield												
0	100	0	0	0	0	500	14	2	13	78	0	71
100	0	0	0	0	0	430	3	3	10	168	1	71
0	0	0	100	0	0	500	5	4	35	60	0	71
0	50	50	0	0	0	560	10	4	23	90	1	70
50	0	0	50	0	0	560	10	4	23	90	1	70
0	0	0	100	0	0	450	5	20	1000	75	1	70
0	0	100	0	0	0	300	5	2	2	80	0	70
0	0	100	0	0	0	300	10	50	1000	60	0	69
100	0	0	0	0	0	430	3	3	10	168	0	69
0	0	100	0	0	0	300	10	50	1000	30	0	69
0	0	100	0	0	0	380	3	2	10	61	1	69
0	0	100	0	0	0	450	20	2	10	55	0	68
0	70	30	0	0	0	560	10	4	23	90	1	68
30	0	0	70	0	0	560	10	4	23	90	1	68
0	100	0	0	0	0	500	20	3	20	66	1	68
100	0	0	0	0	0	430	3	3	10	168	1	68
0	0	100	0	0	0	350	5	2	2	80	0	68
0	0	0	100	0	0	500	10	2	10	60	0	67
0	66	34	0	0	0	460	10	2	10	106	1	67
0	0	100	0	0	0	380	3	2	10	61	0	65
0	0	100	0	0	0	400	5	2	2	61	0	63
0	66	34	0	0	0	460	10	2	10	106	0	63
0	34	66	0	0	0	460	10	2	10	66	1	62
0	100	0	0	0	0	550	5	1	1	77	1	62
0	100	0	0	0	0	450	20	2	10	101	0	61
90	10	0	0	0	0	560	10	4	23	90	1	61
0	0	0	100	0	0	450	10	20	1000	75	1	60
0	0	100	0	0	0	250	4	2	2	124	0	57
0	0	100	0	0	0	460	10	2	10	59	1	57
10	90	0	0	0	0	560	10	4	23	90	1	55
0	0	75	0	0	25	450	10	25	1000	120	1	55
0	0	100	0	0	0	380	3	2	10	61	1	55
100	0	0	0	0	0	400	6	3	2	170	0	54
25	25	0	50	0	0	450	10	19	1000	75	0	54
0	0	100	0	0	0	450	10	20	1000	75	1	54
0	0	50	50	0	0	450	10	20	1000	75	1	54
0	0	100	0	0	0	500	10	25	1000	120	1	53
0	0	100	0	0	0	250	10	50	1000	60	0	52
0	50	0	50	0	0	450	10	20	1000	75	1	52
Split 5: 40 Datapoints 52 to 71% yield												
0	100	0	0	0	0	460	10	2	10	104	1	50
100	0	0	0	0	0	430	3	3	10	168	1	50
13	13	25	50	0	0	450	10	19	1000	75	0	49
44	0	21	13	12	9	500	5	8	35	60	0	49
0	0	100	0	0	0	380	3	2	10	61	1	47
0	0	65	0	0	35	500	10	25	1000	120	1	45
0	50	0	50	0	0	450	10	20	1000	75	1	44
0	50	50	0	0	0	450	10	20	1000	75	1	44
0	25	25	50	0	0	450	10	20	1000	75	1	44
0	0	100	0	0	0	450	10	15	1000	75	0	42
50	50	0	0	0	0	450	10	20	1000	75	1	42
100	0	0	0	0	0	550	5	1	1	73	1	41
10	10	20	40	0	20	450	10	20	1000	75	0	40
50	50	0	0	0	0	450	10	20	1000	75	1	40
0	0	100	0	0	0	450	10	20	1000	75	1	40
0	50	50	0	0	0	450	10	20	1000	75	1	40
0	25	25	50	0	0	450	10	20	1000	75	1	40
0	0	0	0	0	100	450	5	2	10	90	0	39

0	0	0	0	0	100	500	6	2	293	68	0	39
0	100	0	0	0	0	500	10	2	10	60	0	38
0	0	0	0	0	100	450	10	2	10	90	0	35
0	0	0	0	0	100	500	8	2	293	68	0	34
0	0	50	50	0	0	450	10	20	1000	75	1	34
0	0	0	0	0	100	500	10	2	293	68	0	32
100	0	0	0	0	0	300	6	2	2	169	0	31
0	0	0	0	0	100	500	12	2	293	68	0	30
0	20	20	20	0	20	450	10	20	1000	75	1	30
0	0	0	0	0	100	450	15	2	10	67	0	30
0	0	0	0	0	100	450	20	2	10	67	0	29
0	0	0	0	0	100	500	14	2	293	68	0	29
0	90	10	0	0	0	560	10	4	23	90	1	29
0	20	20	20	0	20	450	10	20	1000	75	1	28
50	50	0	0	0	0	450	10	16	1000	75	0	25
0	0	50	50	0	0	450	10	20	1000	75	0	25
25	25	50	0	0	0	450	10	20	1000	75	0	24
100	0	0	0	0	0	450	20	1	20	290	0	24
0	100	0	0	0	0	550	5	1	1	77	1	18
100	0	0	0	0	0	550	5	1	1	73	1	17
0	0	0	0	0	100	500	5	4	35	60	0	15

Appendix D: Random Forest Regressor 100 Simulations Code

```
import pandas as pd
import numpy as np
import io
import statistics
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from google.colab import files
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.neural_network import MLPRegressor
from statistics import mean
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
%matplotlib inline

uploaded = files.upload()
features = pd.read_csv('some data set')
features.head(5)

# Labels are the values we want to predict
labels = np.array(features['Oil Yield '])

#Needed to do this loop because one of the csv's kept outputting strings. Not sure why
#for i in range(len(labels)):
 # labels[i]= float(labels[i])

# Remove the labels from the features
# axis 1 refers to the columns
features= features.drop('Oil Yield ', axis = 1)

# Saving feature names for later use

feature_list = list(features.columns)

# Convert to numpy array
features = np.array(features)
#Regression
#Create many simulations at once

#In compare_vis first number is how many desired simulations, second is the feature index eg Temp is 6
HDPE is 0
#MAE, RMSE ,and R^2 will also be highlighted
#could run this with any ML just need to adjust metrics and code for model prediction
#Creating open lists to put train test splits in
train_features_lists = []
test_features_lists = []
train_labels_lists = []
```

```
test_labels_lists = []

R2_collect = []
MAE_collect = []
RMSE_collect = []
# Put in number of tests desired and the number associated with each feature
def compare_vis(tests,Feature_Name):

  for i in range(tests):
    if i <=tests:
      train_features, test_features, train_labels, test_labels = train_test_split(features, labels, test_size = 0.20
)
      train_features_lists.append(train_features.tolist())
      test_features_lists.append(test_features.tolist())
      train_labels_lists.append(train_labels.tolist())
      test_labels_lists.append(test_labels.tolist())

    #Number of tests results in arrays for train test split being put into the holder lists created outside of the
 function
      #Each train and test set is a 3 layer list. These are created until number of desired lists are created

  else:
    #Once all are created graphing and models can run

    for i in range(tests):
      train_f_list_1 =[]
      test_f_list_1 =[]
      train_l_list_1 =[]
      test_l_list_1 =[]

      test = train_features_lists[i]
      test_2 = test_features_lists[i]
      test_3 = train_labels_lists[i]
      test_4 = test_labels_lists[i]
      #Code puts each simulations splits into variables called test-
test_4 each correspond to a split. These are overridden for each simulation
      '''
      for y in range(len(test)):
        train_f_list_1.append(test[y][Feature_Name]) #looks at created list and puts each desired feature fro
m each list of lists within each simulation
        train_l_list_1.append(test_3[y]) #puts oil yeild associate with that simulation into a list to be graphed



      for c in range(len(test_2)):
        test_f_list_1.append(test_2[c][Feature_Name]) #separate for loop for the test set since the test set is s
maller than the train but does the same function
        test_l_list_1.append(test_4[c]) #as the foor loop above

      plt.scatter(train_f_list_1, train_l_list_1, c='r') #plots training data on scatter plot in red
      plt.scatter(test_f_list_1, test_l_list_1, c='b') #plots testing data on scatter plot in blue
```

```
    plt.ylabel('Oil Yeild')
    plt.xlabel(feature_list[Feature_Name]) #Plots the feature name based on given input
    plt.show()
    '''
    #where code goes for desired model runs
    rf = RandomForestRegressor(n_estimators = 1000)
    rf.fit(test, test_3)

    predictions = rf.predict(test_2)
    errors = abs(predictions - test_4)

    #where the metrics go
    #MAE
    mae=round(np.mean(errors), 2)
    #print('Mean Absolute Error:', mae)
    MAE_collect.append(mae)

    #RMSE
    rms = mean_squared_error(test_4, predictions, squared=False)
    rms = round(rms, 2)
    #print('The Root Mean Square Error (RMSE) is', rms)
    RMSE_collect.append(rms)

    #R2 Score
    score=r2_score(test_4, predictions)
    #print('The R Squared Value is', score)
    R2_collect.append(score)

compare_vis(100,0)

#Getting the average and standard deviations of all MAE and RMSE for simulations

R2_avg = mean(R2_collect)
R2_std = statistics.pstdev(R2_collect)
MAE_avg = mean(MAE_collect)
RMSE_avg = mean(RMSE_collect)
MAE_std = statistics.pstdev(MAE_collect)
RMSE_std = statistics.pstdev(RMSE_collect)

print("R2 Mean ", round(R2_avg,2))
print("R2 Standard Deviation ", round(R2_std, 2))

print("MAE Mean ", round(MAE_avg,2))
print("MAE Standard Deviation ", round(MAE_std, 2))

print("RMSE Mean ", round(RMSE_avg,2))
print("RMSE Standard Deviation ", round(RMSE_std,2))
```

Appendix E: Random Forest Classifier 100 Simulation Code

```
import pandas as pd
import numpy as np
import io
import statistics

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from google.colab import files
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.neural_network import MLPRegressor
from statistics import mean
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score

%matplotlib inline


features = pd.read_csv('some data set')
features.head(5)


# Labels are the values we want to predict
labels = np.array(features['Oil Yield '])

#Needed to do this loop because one of the csv's kept outputting strings. Not sure why
#for i in range(len(labels)):
 # labels[i]= float(labels[i])

# Remove the labels from the features
# axis 1 refers to the columns
features= features.drop('Oil Yield ', axis = 1)

# Saving feature names for later use

feature_list = list(features.columns)

# Convert to numpy array
features = np.array(features)


#Classifier
#Create many simulations at once

#In compare_vis first number is how many desired simulations, second is the feature index eg Temp is 6
HDPE is 0
#MAE, RMSE ,and R^2 will also be highlighted
#could run this with any ML just need to adjust metrics and code for model prediction
```

```
#Creating open lists to put train test splits in
train_features_lists = []
test_features_lists = []
train_labels_lists = []
test_labels_lists = []

Acc_collect = []
# Put in number of tests desired and the number associated with each feature
def compare_vis(tests,Feature_Name):

  for i in range(tests):
    if i <=tests:
      train_features, test_features, train_labels, test_labels = train_test_split(features, labels, test_size = 0.20
)

      train_features_lists.append(train_features.tolist())

      test_features_lists.append(test_features.tolist())

      train_labels_lists.append(train_labels.tolist())

      test_labels_lists.append(test_labels.tolist())

      #Number of tests results in arrays for train test split being put into the holder lists created outside of th
e function
      #Each train and test set is a 3 layer list. These are created until number of desired lists are created

  else:
    #Once all are created graphing and models can run

    for i in range(tests):
      train_f_list_1 =[]
      test_f_list_1 =[]
      train_l_list_1 =[]
      test_l_list_1 =[]

      test = train_features_lists[i]
      test_2 = test_features_lists[i]
      test_3 = train_labels_lists[i]
      test_4 = test_labels_lists[i]



      #Code puts each simulations splits into variables called test-
test_4 each correspond to a split. These are overridden for each simulation

      for y in range(len(test)):
        train_f_list_1.append(test[y][Feature_Name]) #looks at created list and puts each desired feature fro
m each list of lists within each simulation
```

```
        train_l_list_1.append(test_3[y]) #puts oil yeild associate with that simulation into a list to be graphed



    for c in range(len(test_2)):
        test_f_list_1.append(test_2[c][Feature_Name]) #separate for loop for the test set since the test set is s
maller than the train but does the same function
        test_l_list_1.append(test_4[c]) #as the foor loop above

    #plt.scatter(train_f_list_1, train_l_list_1, c='r') #plots training data on scatter plot in red
    #plt.scatter(test_f_list_1, test_l_list_1, c='b') #plots testing data on scatter plot in blue
    #plt.ylabel('Oil Yeild')
    #plt.xlabel(feature_list[Feature_Name]) #Plots the feature name based on given input
    #plt.show()

    #where code goes for desired model runs

    #Classifier Code
    clf = RandomForestClassifier(n_estimators=1000)
    clf.fit(test, test_3)
    predictions = clf.predict(test_2)
    accuracy = accuracy_score(test_4,predictions)
    Acc_collect.append(accuracy)

compare_vis(100,0)
Acc_avg = mean(Acc_collect)
Acc_std = statistics.pstdev(Acc_collect)
print("Accuracy Mean ", round(Acc_avg,2))
print("Accuracy Standard Deviation ", round(Acc_std,2))
```

Appendix F: Python Cross_val_score Cross Validation for Regression and Classification Models

```
#Regressor Cross Validation
scores_2 = cross_val_score(rf,features,labels,cv=5,scoring='neg_mean_absolute_error')
scores_3 = cross_val_score(rf,features,labels,cv=5,scoring='neg_root_mean_squared_error')
#'neg_root_mean_squared_error'
#'neg_mean_absolute_error'
print(scores_2)
print("MAE Mean ", round(np.mean(scores_2), 2))
print("MAE Std Dev ", round(np.std(scores_2), 2))
print(scores_3)
print("RMSE Mean ", round(np.mean(scores_3), 2))
print("RMSE Std Dev ", round(np.std(scores_3), 2))

#Classification Cross Validation
classification_score = cross_val_score(clf, features, labels, cv =5, scoring = 'accuracy')
print(classification_score)
print("Accuracy Mean ", round(np.mean(classification_score), 2))
print("Accuracy Std Dev ", round(np.std(classification_score), 2))
```

Appendix G: Stratified Cross Validation Method

```
#This code performs the Stratified Cross Validation Method
import pandas as pd
import numpy as np
import io
import statistics

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from google.colab import files
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from statistics import mean
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier

%matplotlib inline

uploaded = files.upload()

#Where each dataset split is added to variable names (each CSV should have a different 'distrubution of the data')
sample_1 = pd.read_csv('a')
sample_2 = pd.read_csv('b')
sample_3 = pd.read_csv('c')
sample_4 = pd.read_csv('d')
sample_5 = pd.read_csv('e')
#Shuffling All Sets

#Takes each added csv which became a dataframe and splits it randomly into 5 sets which are also dataframes
shuffled_1 = sample_1.sample(frac=0.8)
result_1 = np.array_split(shuffled_1, 5)

shuffled_2 = sample_2.sample(frac=0.8)
result_2 = np.array_split(shuffled_2, 5)

shuffled_3 = sample_3.sample(frac=0.8)
result_3 = np.array_split(shuffled_3, 5)

shuffled_4 = sample_4.sample(frac=0.8)
result_4= np.array_split(shuffled_4, 5)

shuffled_5 = sample_5.sample(frac=0.8)
result_5 = np.array_split(shuffled_5, 5)


#Spliting All Sets

#This takes each dataframe from the shuffles and assocaites it with a variable that can be added to each "k-fold"
#Each dataframe becomes and array which then becomes a list (necesary because second part of the code was create
d dealing with list' first)
split_1a =np.array(result_1[0]).tolist()
split_2a =np.array(result_1[1]).tolist()
split_3a =np.array(result_1[2]).tolist()
split_4a =np.array(result_1[3]).tolist()
```

```
split_5a =np.array(result_1[4]).tolist()

split_1b =np.array(result_2[0]).tolist()
split_2b =np.array(result_2[1]).tolist()
split_3b =np.array(result_2[2]).tolist()
split_4b =np.array(result_2[3]).tolist()
split_5b =np.array(result_2[4]).tolist()

split_1c =np.array(result_3[0]).tolist()
split_2c =np.array(result_3[1]).tolist()
split_3c =np.array(result_3[2]).tolist()
split_4c =np.array(result_3[3]).tolist()
split_5c =np.array(result_3[4]).tolist()

split_1d =np.array(result_4[0]).tolist()
split_2d =np.array(result_4[1]).tolist()
split_3d =np.array(result_4[2]).tolist()
split_4d =np.array(result_4[3]).tolist()
split_5d =np.array(result_4[4]).tolist()

split_1e =np.array(result_5[0]).tolist()
split_2e =np.array(result_5[1]).tolist()
split_3e =np.array(result_5[2]).tolist()
split_4e =np.array(result_5[3]).tolist()
split_5e =np.array(result_5[4]).tolist()


#Regression: Adding all splits to associated k fold sets

#Each k fold should have the necesary splits of data from the csvs entered
t1 = split_1a + split_1b + split_1c + split_1d + split_1e
t2 = split_2a + split_2b + split_2c + split_2d + split_2e
t3 = split_3a + split_3b + split_3c + split_3d + split_3e
t4 = split_4a + split_4b + split_4c + split_4d + split_4e
t5 = split_5a + split_5b + split_5c + split_5d + split_5e


#Creat lists to put MAE in and RMSE in
MAE_Holder=[]
RMSE_Holder =[]

#Brute Force Method

#Cross Fold 1 : 1-4 Train 5 is test

training_1 = t1+t2+t3+t4
testing_1 = t5
training_1_labels_1=[]
training_1_features_1=[]
testing_1_labels_1 =[]
testing_1_features_1 = []

for i in range(len(training_1)):
  training_1_features_1.append(training_1[i][0:(len(training_1[i])-1)])
  training_1_labels_1.append(training_1[i][-1])
```

```
for i in range(len(t5)):
  testing_1_features_1.append(t5[i][0:-1])
  testing_1_labels_1.append(t5[i][-1])

training_1_features_1 = np.array(training_1_features_1)
training_1_labels_1= np.array(training_1_labels_1)
testing_1_features_1 = np.array(testing_1_features_1)
testing_1_labels_1 = np.array(testing_1_labels_1)

rf = RandomForestRegressor(n_estimators = 1000)
rf.fit(training_1_features_1, training_1_labels_1)

predictions_1 = rf.predict(testing_1_features_1)
errors_1 = abs(predictions_1 - testing_1_labels_1)
mae_1 = round(np.mean(errors_1), 2)
rmse_1 = (mean_squared_error(testing_1_labels_1, predictions_1, squared=False))
MAE_Holder.append(mae_1)
RMSE_Holder.append(rmse_1)

#Cross Fold 2 Train 1,2,3,5 Test 4

training_1 = t1+t2+t3+t5
testing_1 = t4
training_1_labels_1=[]
training_1_features_1=[]
testing_1_labels_1 =[]
testing_1_features_1 = []

for i in range(len(training_1)):
  training_1_features_1.append(training_1[i][0:(len(training_1[i])-1)])
  training_1_labels_1.append(training_1[i][-1])

for i in range(len(t4)):
  testing_1_features_1.append(t4[i][0:-1])
  testing_1_labels_1.append(t4[i][-1])

training_1_features_1 = np.array(training_1_features_1)
training_1_labels_1= np.array(training_1_labels_1)
testing_1_features_1 = np.array(testing_1_features_1)
testing_1_labels_1 = np.array(testing_1_labels_1)

rf = RandomForestRegressor(n_estimators = 1000)
rf.fit(training_1_features_1, training_1_labels_1)

predictions_1 = rf.predict(testing_1_features_1)
errors_1 = abs(predictions_1 - testing_1_labels_1)
mae_1 = round(np.mean(errors_1), 2)
rmse_1 = (mean_squared_error(testing_1_labels_1, predictions_1, squared=False))
MAE_Holder.append(mae_1)
RMSE_Holder.append(rmse_1)

#Cross Fold 3 Train 1,2,4,5 Test 3

training_1 = t1+t2+t4+t5
testing_1 = t3
training_1_labels_1=[]
```

```
training_1_features_1=[]
testing_1_labels_1 =[]
testing_1_features_1 = []

for i in range(len(training_1)):
  training_1_features_1.append(training_1[i][0:(len(training_1[i])-1)])
  training_1_labels_1.append(training_1[i][-1])

for i in range(len(t3)):
  testing_1_features_1.append(t3[i][0:-1])
  testing_1_labels_1.append(t3[i][-1])

training_1_features_1 = np.array(training_1_features_1)
training_1_labels_1= np.array(training_1_labels_1)
testing_1_features_1 = np.array(testing_1_features_1)
testing_1_labels_1 = np.array(testing_1_labels_1)

rf = RandomForestRegressor(n_estimators = 1000)
rf.fit(training_1_features_1, training_1_labels_1)

predictions_1 = rf.predict(testing_1_features_1)
errors_1 = abs(predictions_1 - testing_1_labels_1)
mae_1 = round(np.mean(errors_1), 2)
rmse_1 = (mean_squared_error(testing_1_labels_1, predictions_1, squared=False))
MAE_Holder.append(mae_1)
RMSE_Holder.append(rmse_1)

#Cross Fold 4 Train 1,3,4,5 Test 2

training_1 = t1+t3+t4+t5
testing_1 = t2
training_1_labels_1=[]
training_1_features_1=[]
testing_1_labels_1 =[]
testing_1_features_1 = []

for i in range(len(training_1)):
  training_1_features_1.append(training_1[i][0:(len(training_1[i])-1)])
  training_1_labels_1.append(training_1[i][-1])

for i in range(len(t2)):
  testing_1_features_1.append(t2[i][0:-1])
  testing_1_labels_1.append(t2[i][-1])

training_1_features_1 = np.array(training_1_features_1)
training_1_labels_1= np.array(training_1_labels_1)
testing_1_features_1 = np.array(testing_1_features_1)
testing_1_labels_1 = np.array(testing_1_labels_1)

rf = RandomForestRegressor(n_estimators = 1000)
rf.fit(training_1_features_1, training_1_labels_1)

predictions_1 = rf.predict(testing_1_features_1)
errors_1 = abs(predictions_1 - testing_1_labels_1)
mae_1 = round(np.mean(errors_1), 2)
rmse_1 = (mean_squared_error(testing_1_labels_1, predictions_1, squared=False))
```

```
MAE_Holder.append(mae_1)
RMSE_Holder.append(rmse_1)

#Cross Fold 5 Train 2,3,4,5 Test 1
training_1 = t2+t3+t4+t5
testing_1 = t1
training_1_labels_1=[]
training_1_features_1=[]
testing_1_labels_1 =[]
testing_1_features_1 = []

for i in range(len(training_1)):
  training_1_features_1.append(training_1[i][0:(len(training_1[i])-1)])
  training_1_labels_1.append(training_1[i][-1])

for i in range(len(t1)):
  testing_1_features_1.append(t1[i][0:-1])
  testing_1_labels_1.append(t1[i][-1])

training_1_features_1 = np.array(training_1_features_1)
training_1_labels_1= np.array(training_1_labels_1)
testing_1_features_1 = np.array(testing_1_features_1)
testing_1_labels_1 = np.array(testing_1_labels_1)

rf = RandomForestRegressor(n_estimators = 1000)
rf.fit(training_1_features_1, training_1_labels_1)

predictions_1 = rf.predict(testing_1_features_1)
errors_1 = abs(predictions_1 - testing_1_labels_1)
mae_1 = round(np.mean(errors_1), 2)
rmse_1 = (mean_squared_error(testing_1_labels_1, predictions_1, squared=False))
MAE_Holder.append(mae_1)
RMSE_Holder.append(rmse_1)

#Note: Did not need to rename variabes since python overwrites the new variables onece redefined

print("MAE:", MAE_Holder)
print("MAE Std Dev:", statistics.pstdev(MAE_Holder))
print("MAE Mean:", mean(MAE_Holder))

print("RMSE:", RMSE_Holder)
print("RMSE Std Dev:", statistics.pstdev(RMSE_Holder))

print("RMSE Mean:", mean(RMSE_Holder))


#Classifier Test
#Adding all splits to associated k fold sets

#Each k fold should have the necesary splits of data from the csvs entered
t1 = split_1a + split_1b + split_1c + split_1d + split_1e
t2 = split_2a + split_2b + split_2c + split_2d + split_2e
t3 = split_3a + split_3b + split_3c + split_3d + split_3e
t4 = split_4a + split_4b + split_4c + split_4d + split_4e
t5 = split_5a + split_5b + split_5c + split_5d + split_5e
```

```
#Creat lists to put Accuracies in
Accuracy_Holder = []

#Brute Force Method

#Cross Fold 1 : 1-4 Train 5 is test

training_1 = t1+t2+t3+t4
testing_1 = t5
training_1_labels_1=[]
training_1_features_1=[]
testing_1_labels_1 =[]
testing_1_features_1 = []

for i in range(len(training_1)):
  training_1_features_1.append(training_1[i][0:(len(training_1[i])-1)])
  training_1_labels_1.append(training_1[i][-1])

for i in range(len(t5)):
  testing_1_features_1.append(t5[i][0:-1])
  testing_1_labels_1.append(t5[i][-1])

training_1_features_1 = np.array(training_1_features_1)
training_1_labels_1= np.array(training_1_labels_1)
testing_1_features_1 = np.array(testing_1_features_1)
testing_1_labels_1 = np.array(testing_1_labels_1)

clf = RandomForestClassifier(n_estimators=1000)
clf.fit(training_1_features_1, training_1_labels_1)
predictions_1 = clf.predict(testing_1_features_1)
accuracy = accuracy_score(testing_1_labels_1,predictions_1)
Accuracy_Holder.append(accuracy)

#Cross Fold 2 Train 1,2,3,5 Test 4

training_1 = t1+t2+t3+t5
testing_1 = t4
training_1_labels_1=[]
training_1_features_1=[]
testing_1_labels_1 =[]
testing_1_features_1 = []

for i in range(len(training_1)):
  training_1_features_1.append(training_1[i][0:(len(training_1[i])-1)])
  training_1_labels_1.append(training_1[i][-1])

for i in range(len(t4)):
  testing_1_features_1.append(t4[i][0:-1])
  testing_1_labels_1.append(t4[i][-1])

training_1_features_1 = np.array(training_1_features_1)
training_1_labels_1= np.array(training_1_labels_1)
testing_1_features_1 = np.array(testing_1_features_1)
testing_1_labels_1 = np.array(testing_1_labels_1)

clf = RandomForestClassifier(n_estimators=1000)
```

```
clf.fit(training_1_features_1, training_1_labels_1)
predictions_1 = clf.predict(testing_1_features_1)
accuracy = accuracy_score(testing_1_labels_1,predictions_1)
Accuracy_Holder.append(accuracy)

#Cross Fold 3 Train 1,2,4,5 Test 3

training_1 = t1+t2+t4+t5
testing_1 = t3
training_1_labels_1=[]
training_1_features_1=[]
testing_1_labels_1 =[]
testing_1_features_1 = []

for i in range(len(training_1)):
  training_1_features_1.append(training_1[i][0:(len(training_1[i])-1)])
  training_1_labels_1.append(training_1[i][-1])

for i in range(len(t3)):
  testing_1_features_1.append(t3[i][0:-1])
  testing_1_labels_1.append(t3[i][-1])

training_1_features_1 = np.array(training_1_features_1)
training_1_labels_1= np.array(training_1_labels_1)
testing_1_features_1 = np.array(testing_1_features_1)
testing_1_labels_1 = np.array(testing_1_labels_1)

clf = RandomForestClassifier(n_estimators=1000)
clf.fit(training_1_features_1, training_1_labels_1)
predictions_1 = clf.predict(testing_1_features_1)
accuracy = accuracy_score(testing_1_labels_1,predictions_1)
Accuracy_Holder.append(accuracy)


#Cross Fold 4 Train 1,3,4,5 Test 2

training_1 = t1+t3+t4+t5
testing_1 = t2
training_1_labels_1=[]
training_1_features_1=[]
testing_1_labels_1 =[]
testing_1_features_1 = []

for i in range(len(training_1)):
  training_1_features_1.append(training_1[i][0:(len(training_1[i])-1)])
  training_1_labels_1.append(training_1[i][-1])

for i in range(len(t2)):
  testing_1_features_1.append(t2[i][0:-1])
  testing_1_labels_1.append(t2[i][-1])

training_1_features_1 = np.array(training_1_features_1)
training_1_labels_1= np.array(training_1_labels_1)
testing_1_features_1 = np.array(testing_1_features_1)
testing_1_labels_1 = np.array(testing_1_labels_1)
```

```
clf = RandomForestClassifier(n_estimators=1000)
clf.fit(training_1_features_1, training_1_labels_1)
predictions_1 = clf.predict(testing_1_features_1)
accuracy = accuracy_score(testing_1_labels_1,predictions_1)
Accuracy_Holder.append(accuracy)

#Cross Fold 5 Train 2,3,4,5 Test 1
training_1 = t2+t3+t4+t5
testing_1 = t1
training_1_labels_1=[]
training_1_features_1=[]
testing_1_labels_1 =[]
testing_1_features_1 = []

for i in range(len(training_1)):
  training_1_features_1.append(training_1[i][0:(len(training_1[i])-1)])
  training_1_labels_1.append(training_1[i][-1])

for i in range(len(t1)):
  testing_1_features_1.append(t1[i][0:-1])
  testing_1_labels_1.append(t1[i][-1])

training_1_features_1 = np.array(training_1_features_1)
training_1_labels_1= np.array(training_1_labels_1)
testing_1_features_1 = np.array(testing_1_features_1)
testing_1_labels_1 = np.array(testing_1_labels_1)

clf = RandomForestClassifier(n_estimators=1000)
clf.fit(training_1_features_1, training_1_labels_1)
predictions_1 = clf.predict(testing_1_features_1)
accuracy = accuracy_score(testing_1_labels_1,predictions_1)
Accuracy_Holder.append(accuracy)


print("Accuracy:", Accuracy_Holder)
print("Accuracy Std Dev:", statistics.pstdev(Accuracy_Holder))
print("Accuracy", mean(Accuracy_Holder)
```

Appendix H: Error Comparison Code

```
import pandas as pd
import numpy as np
import io

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from google.colab import files
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
%matplotlib inline
uploaded = files.upload()
features = pd.read_csv('Some data set')
features.head(5)


# Labels are the values we want to predict
labels = np.array(features['Oil Yield '])

# Remove the labels from the features
# axis 1 refers to the columns
features= features.drop('Oil Yield ', axis = 1)

# Saving feature names for later use

feature_list = list(features.columns)

# Convert to numpy array
features = np.array(features)
Code that runs simulations and puts errors for cell ids into a dataframe
from collections import Counter
from itertools import chain
def error_finder(tests):

  simulations_all = [] #holds keys, predictions, and errors
  simulations_keys = [] #holds only the keys

  #Gets ids of all codes
  feature_strings = [[str(x) for x in inner_int] for inner_int in features]
  code_ids = [["".join(inner_comps[0:len(feature_list)])] for inner_comps in feature_strings]
  #Might want to add ids if that is helpful
  for i in range(tests):
    train_features, test_features, train_labels, test_labels = train_test_split(features, labels, test_size = 0.20)

    test_features_variable = test_features.tolist()
    test_labels_variable= test_labels.tolist()



    rf = RandomForestRegressor(n_estimators = 1000)
    rf.fit(train_features, train_labels);

    predictions = rf.predict(test_features)
```

```
    errors = abs(predictions - test_labels)


    #Converting the test features to strings so they can be concatenated
    test_feature_strings = [[str(x) for x in inner_int] for inner_int in test_features_variable]
    #Convert features to one cell by concatenating for the length of the feature list
    new_strings_conc =  [[''.join(inner_comps[0:len(feature_list)])] for inner_comps in test_feature_strings]

    #Convert test labels as strings
    test_label_strings = [str(oil) for oil in test_labels_variable]

    #Putting Yields, predictions, and errors with feature ID. Yields get appended
    for i in range(len(new_strings_conc)):
      new_strings_conc[i].append(test_label_strings[i])


    new_strings_conc_2 = [[''.join(inner_comps[0:2])] for inner_comps in new_strings_conc]

    for i in range(len(new_strings_conc)):
      new_strings_conc_2[i].append(predictions[i])
      new_strings_conc_2[i].append(errors[i])

    #Separate entry if error is more than x (could change this)

    large_error_list = [] #contains keys, predictions, and errors
    large_error_keys = []
    for i in range(len(new_strings_conc_2)):
      if new_strings_conc_2[i][-1] >=10:
        large_error_list.append(new_strings_conc_2[i])

        for keys in range(len(large_error_list)):
          large_error_keys.append(large_error_list[keys][0])

    simulations_all.append(large_error_list)
    simulations_keys.append(large_error_keys)

    #Counts number of times a certain cell appears
    no_of_lists_per_name = Counter(chain.from_iterable(map(set, simulations_keys)))

    for name, no_of_lists in no_of_lists_per_name.most_common():
        if no_of_lists == 1:
            break # since it is ordered by count, once we get this low we are done
        #print(f"'{name}' is in {no_of_lists} lists")

  #Create a dataframe with IDs and number of errors
  error_df = pd.DataFrame.from_dict(no_of_lists_per_name, orient='index').reset_index()
  error_df = error_df.rename(columns={'index':'ID', 0:'count'})
  print(error_df)

  error_df.to_excel(excel_writer ="error_compare.xlsx")
  from google.colab import files
  files.download('error_compare.xlsx')

error_finder(100)
```