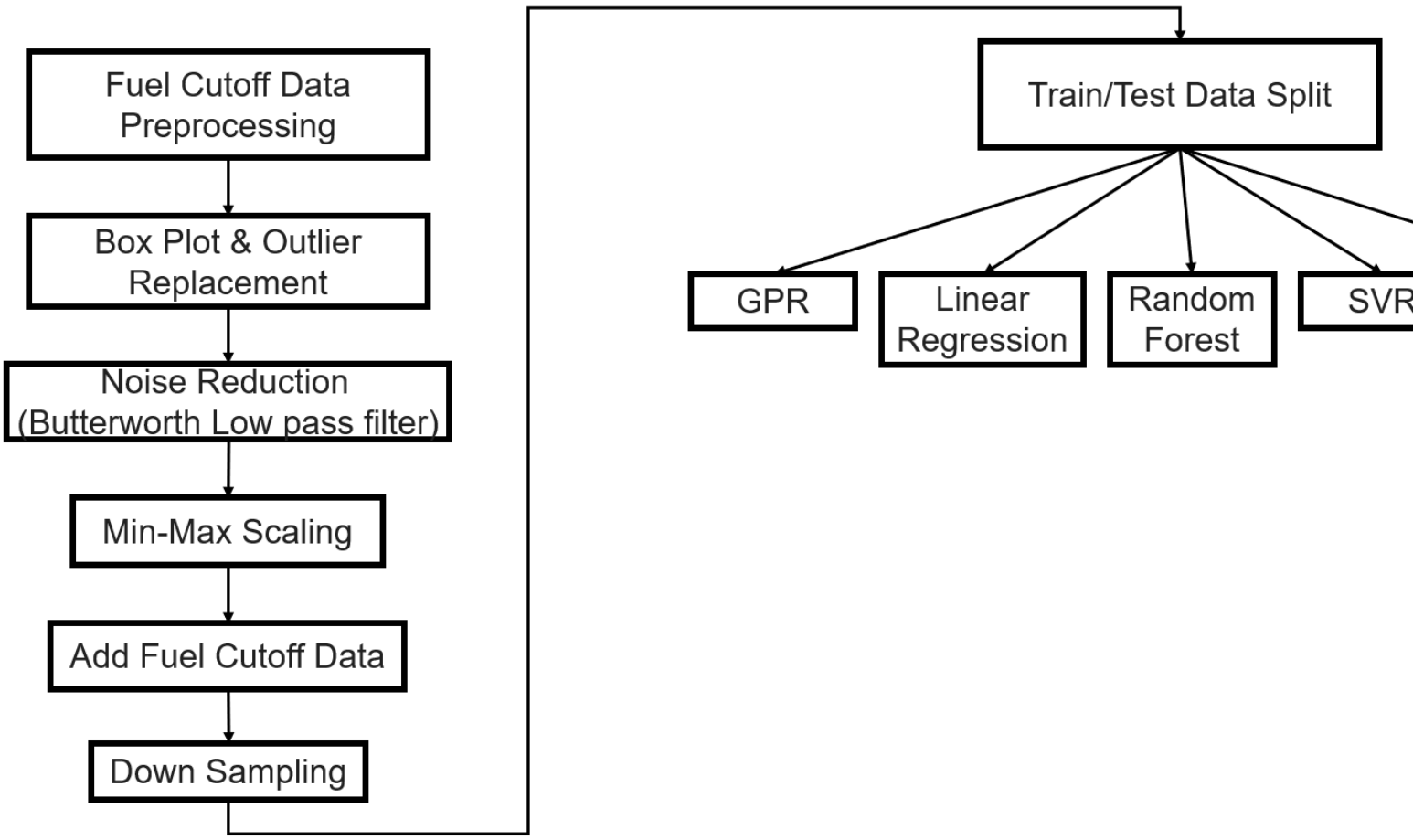
Project#1-2: BOSCH Datasets Multi-Output Gaussian Regression (MOGP)

Class: Industrial AI & Automation

Author: GYEONHEAL AN(21900416), GARMA JIN(21900727)

Date: 2024-10-29



Background Introduction

- Necessity

- Emissions and performance indicators from automotive engines are crucial for meeting environmental regulations and ensuring vehicle efficiency.
- Pollutants like CO, HC, and CO_2 contribute to environmental degradation, and engine performance decline leads to increased maintenance costs and reduced safety.
- Predicting emissions and performance indicators based on engine data is essential to address these challenges.

- Technical Challenges

- Traditionally, analyzing emissions and performance indicators requires costly equipment such as sensors and Gas Analyzers.
- Such equipment is expensive and requires accessible environments for routine inspections.
- To detect performance decline early and conduct preventive maintenance, precise predictions across various driving conditions are essential.

- Solution

- The BOSCH engine dataset enables us to analyze and predict engine input and output variables.
- The MOGP (Multi-Output Gaussian Process) model is well-suited for accurate predictions, as it captures the correlations among multiple output variables.
- Through the MOGP model, real-time engine condition monitoring and fault prediction can be achieved without emissions analysis equipment.
- This model allows for effective engine performance management and proactive emissions issue prevention under diverse conditions.
- Combining the BOSCH engine dataset with the MOGP model provides an efficient and cost-effective solution for engine management systems.

Objective

- Evaluate the predictive performance of the MOGP model in comparison with other regression models.
- Demonstrate the predictive accuracy of the MOGP model by achieving a lower RMSE value compared to alternative models.
- Enhance the reliability of emissions and temperature predictions through reduced RMSE, thereby improving engine condition forecasting accuracy.

Baseline Survey & Datasets

- **Datasets:** https://github.com/boschresearch/Bosch-Engine-Datasets
- **Baseline Journal:** https://proceedings.mlr.press/v151/li22d.html

[1] Li, Z., Vijitbenjaronk, W., Zhang, S., & Calandra, R. (2022). *Learning how to safely explore dynamic systems*. In *Proceedings of the 5th Conference on Learning for Dynamics and Control* (Vol. 151, pp. 1223-1234).

[2] Bosch Research. (n.d.). *Bosch Engine Datasets*. GitHub repository. Retrieved from https://github.com/boschresearch/Bosch-Engine-Datasets

Contents

1. Raw BOSCH Datasets Analysis

- Raw Data Plot
- Correlation Check

2. Data Preprocessing

- Fuel Cutoff Data Preprocessing
- Box Plot & Outlier Replacement
- Noise Reduction - Low pass filter
- Min-Max Scaling
- Add Fuel Cutoff Data
- Down Sampling

3. Classification

- Train/Test Data Split
- GPR
- Linear Regression / Random Forest / SVM

4. Result & Discussion

0. Import Datasets

```
clc; clear; close all;

num_files = 33;
bosch_datasets = [];

for i = num_files
    filename = sprintf('../BOSCH_Datasets/gengine1/BOSCH_raw_%02d.csv', i);
    data = readtable(filename);
    bosch_datasets = [bosch_datasets, data];
end
bosch_datasets.Properties.VariableNames = {'Speed', 'Load', 'Lambda',
'Ignition_angle', 'Fuel_cutoff', 'Particle_no', 'CO', 'CO2', 'HC', 'NOx', 'O2',
'Temp_manifold', 'Temp_catalyst'};
bosch_datasets.Properties
```

```
ans =
  TableProperties - 속성 있음:

            Description: ''
               UserData: []
         DimensionNames: {'Row'  'Variables'}
          VariableNames: {'Speed'  'Load'  'Lambda'  'Ignition_angle'  'Fuel_cutoff'  'Particle_no'  'CO'  'CO2'
          VariableTypes: ["double"    "double"    "double"    "double"    "double"    "double"    "double"    "doub
   VariableDescriptions: {}
         VariableUnits: {}
     VariableContinuity: []
               RowNames: {}
       CustomProperties: 사용자 지정 속성을 설정하지 않았습니다.
 addprop 및 rmprop을(를) 사용하여 CustomProperties를 수정하십시오.
```

```
% Data Length Configuration
L = height(bosch_datasets);
% Data Copy
```

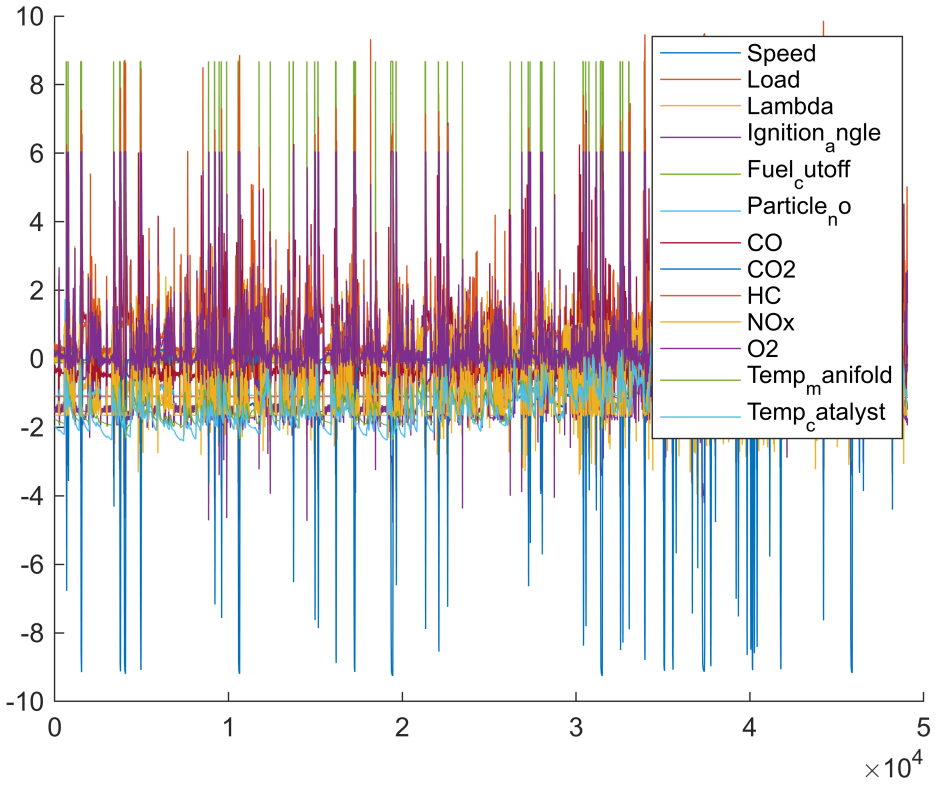
```
datasets = bosch_datasets;
```

1. Raw BOSCH Datasets Analysis

1.1. Raw Data Plot

Examined the overall graph structure of the BOSCH Dataset.

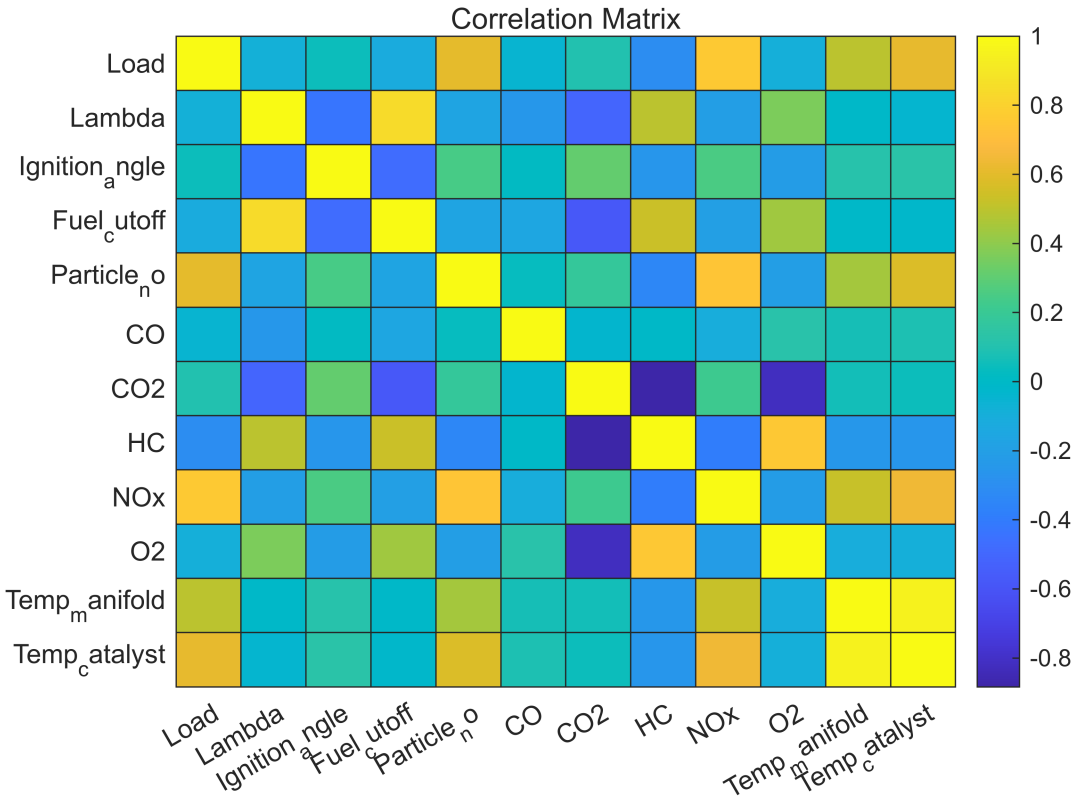
```
figure;
hold on;
for i = 1:length(datasets.Properties.VariableNames)
    plot(table2array(datasets(:, i)))
end
hold off;
legend(datasets.Properties.VariableNames)
```



1.2. Correlation Check

To identify relationships between variables, a **Correlation Heatmap** was generated.

```
figure;
variable_names = datasets.Properties.VariableNames(2:end);
corr_matrix = corrcoef(table2array(datasets(:, 2:end)));
heatmap(corr_matrix, 'XData', variable_names, 'YData', variable_names, ...
    'Title', 'Correlation Matrix', 'Colormap', parula);
```




```
disp(corr_matrix);
```

```
    1.0000   -0.0824    0.0473   -0.1190    0.6096   -0.0496    0.1017   -0.3072    0.7608   -0.0917    0.4973    0
   -0.0824    1.0000   -0.4339    0.8452   -0.1681   -0.2488   -0.5138    0.4989   -0.2045    0.3630   -0.0138   -0
    0.0473   -0.4339    1.0000   -0.4784    0.2425    0.0173    0.3141   -0.2522    0.2550   -0.2196    0.1134    0
   -0.1190    0.8452   -0.4784    1.0000   -0.1665   -0.1515   -0.5796    0.5352   -0.1953    0.4393   -0.0100   -0
    0.6096   -0.1681    0.2425   -0.1665    1.0000    0.0319    0.1774   -0.3397    0.7395   -0.1999    0.4472    0
   -0.0496   -0.2488    0.0173   -0.1515    0.0319    1.0000   -0.0325   -0.0036   -0.1073    0.1216    0.0720    0
    0.1017   -0.5138    0.3141   -0.5796    0.1774   -0.0325    1.0000   -0.8833    0.2130   -0.8245    0.0649    0
   -0.3072    0.4989   -0.2522    0.5352   -0.3397   -0.0036   -0.8833    1.0000   -0.3975    0.7505   -0.2462   -0
    0.7608   -0.2045    0.2550   -0.1953    0.7395   -0.1073    0.2130   -0.3975    1.0000   -0.2142    0.5263    0
   -0.0917    0.3630   -0.2196    0.4393   -0.1999    0.1216   -0.8245    0.7505   -0.2142    1.0000   -0.1053   -0
    0.4973   -0.0138    0.1134   -0.0100    0.4472    0.0720    0.0649   -0.2462    0.5263   -0.1053    1.0000    0
    0.6117   -0.0394    0.1312   -0.0220    0.5786    0.0850    0.0484   -0.2563    0.6398   -0.0919    0.9534    1
```

```
% Find Variables that Correlation is higher than threshold
corr_threshold = 0.8;
high_corr = abs(corr_matrix) > corr_threshold;
high_corr = high_corr - eye(size(high_corr));
[rows, cols] = find(high_corr);

cor_idx = unique(max(rows, cols));
variable_names = datasets.Properties.VariableNames(2:end);
cor_vars = variable_names(cor_idx);

disp(['Highly Correlated Variables: ', strjoin(cor_vars, ', ')]);
```

```
Highly Correlated Variables: Fuel_cutoff, HC, O2, Temp_catalyst
```

2. Data Preprocessing

2.1. Fuel Cutoff Data Preprocessing

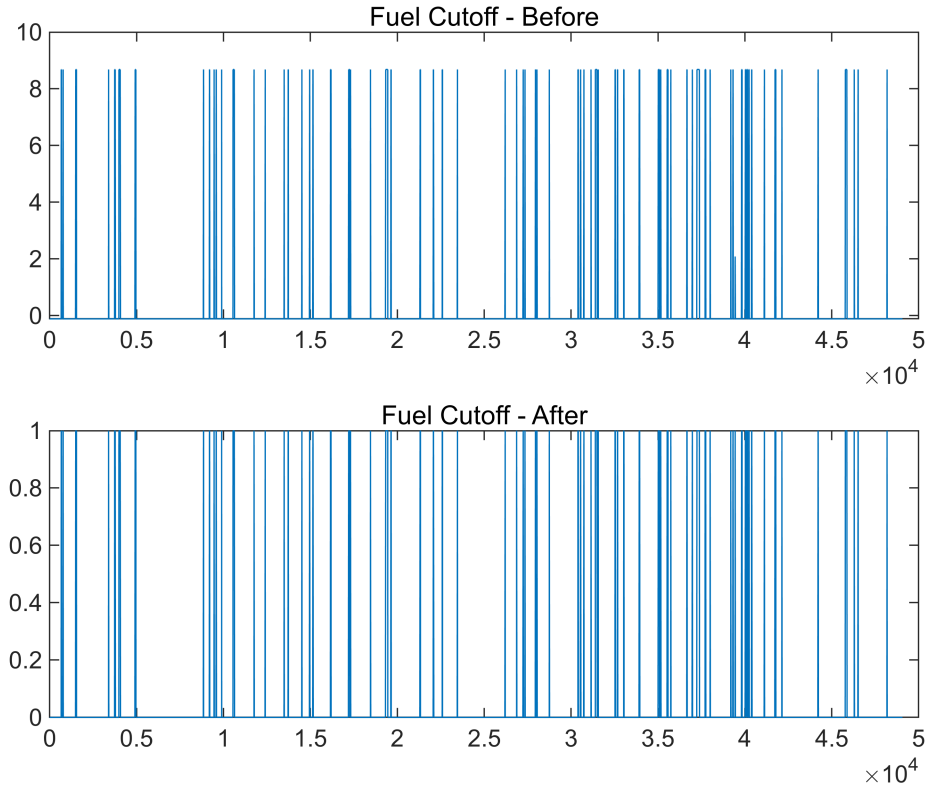
The *Fuel Cutoff* data represents the state or timing when the fuel supply to the engine is cut off. This function is used to reduce fuel consumption or ensure safety under certain engine conditions. For instance, fuel supply is halted when the driver does not press the accelerator pedal or when the vehicle is braking.

Thus, *Fuel Cutoff* data is binary, with values of 0 or 1. If the *Fuel Cutoff* data is not separated, potential transformations or distortions might occur in the upcoming data preprocessing steps; therefore, it was excluded from preprocessing.

Additionally, raw *Fuel Cutoff* values below 0 were set to 0, and values equal to or above 0 were set to 1.

```
fc_index_0 = find(bosch_datasets.Fuel_cutoff < 0);
fc_index_1 = find(bosch_datasets.Fuel_cutoff > 0);
% fc_index_1 = find(bosch_datasets.Fuel_cutoff > 8);
datasets.Fuel_cutoff(fc_index_0) = 0;
datasets.Fuel_cutoff(fc_index_1) = 1;

% Exclude the Fuel Cutoff data along preprocessing
fuel_cutoff = datasets.Fuel_cutoff;
datasets = datasets(:, ~strcmp(datasets.Properties.VariableNames, 'Fuel_cutoff'));
```

```
figure;
subplot(2,1,1);
plot(bosch_datasets.Fuel_cutoff)
title('Fuel Cutoff - Before')
subplot(2,1,2);
plot(fuel_cutoff)
title('Fuel Cutoff - After')
```

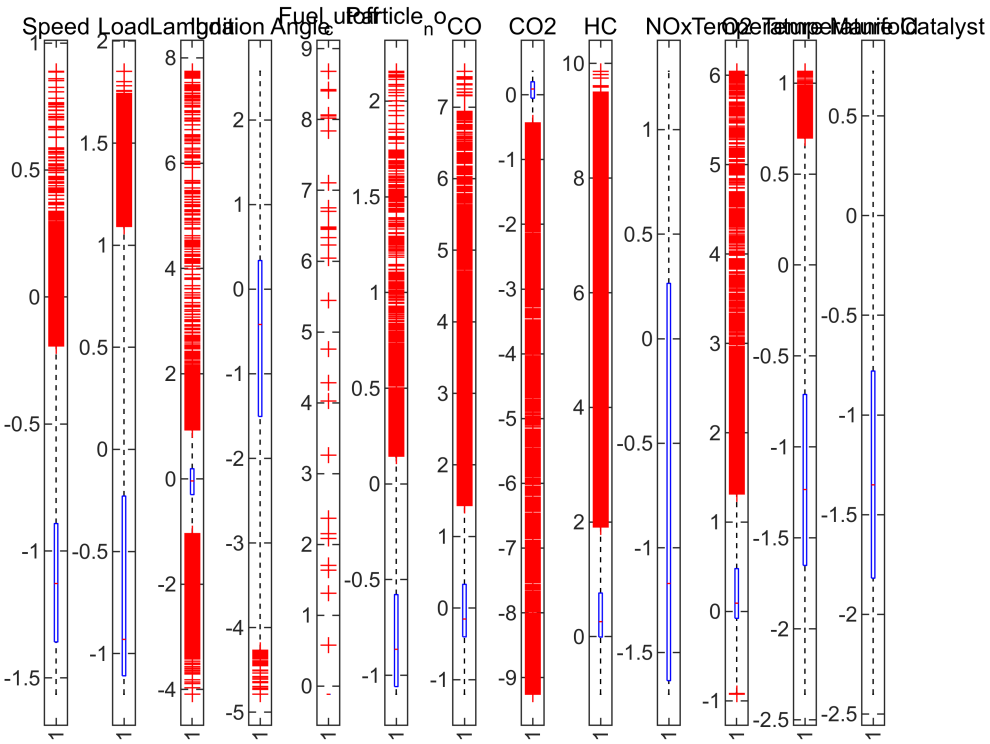


2.2. Box Plot & Outlier Replacement

Box plot is a visual tool that displays the distribution of data, showing the **minimum, first quartile, median, third quartile, and maximum**, which helps visualize the spread and detect outliers in the dataset.

```
figure;
subplot(1,13,1);
boxplot(bosch_datasets.Speed)
title("Speed")
subplot(1,13,2);
boxplot(bosch_datasets.Load)
title("Load")
subplot(1,13,3);
boxplot(bosch_datasets.Lambda)
title("Lambda")
subplot(1,13,4);
boxplot(bosch_datasets.Ignition_angle)
title("Ignition Angle")
subplot(1,13,5);
boxplot(bosch_datasets.Fuel_cutoff)
title("Fuel_cutoff")
subplot(1,13,6);
boxplot(bosch_datasets.Particle_no)
title("Particle_no")
```

```
subplot(1,13,7);
boxplot(bosch_datasets.CO)
title("CO")
subplot(1,13,8);
boxplot(bosch_datasets.CO2)
title("CO2")
subplot(1,13,9);
boxplot(bosch_datasets.HC)
title("HC")
subplot(1,13,10);
boxplot(bosch_datasets.NOx)
title("NOx")
subplot(1,13,11);
boxplot(bosch_datasets.O2)
title("O2")
subplot(1,13,12);
boxplot(bosch_datasets.Temp_manifold)
title("Temperature Manifold")
subplot(1,13,13);
boxplot(bosch_datasets.Temp_catalyst)
title("Temperature Catalyst")
```



Outlier Replacement Using Linear Interpolation

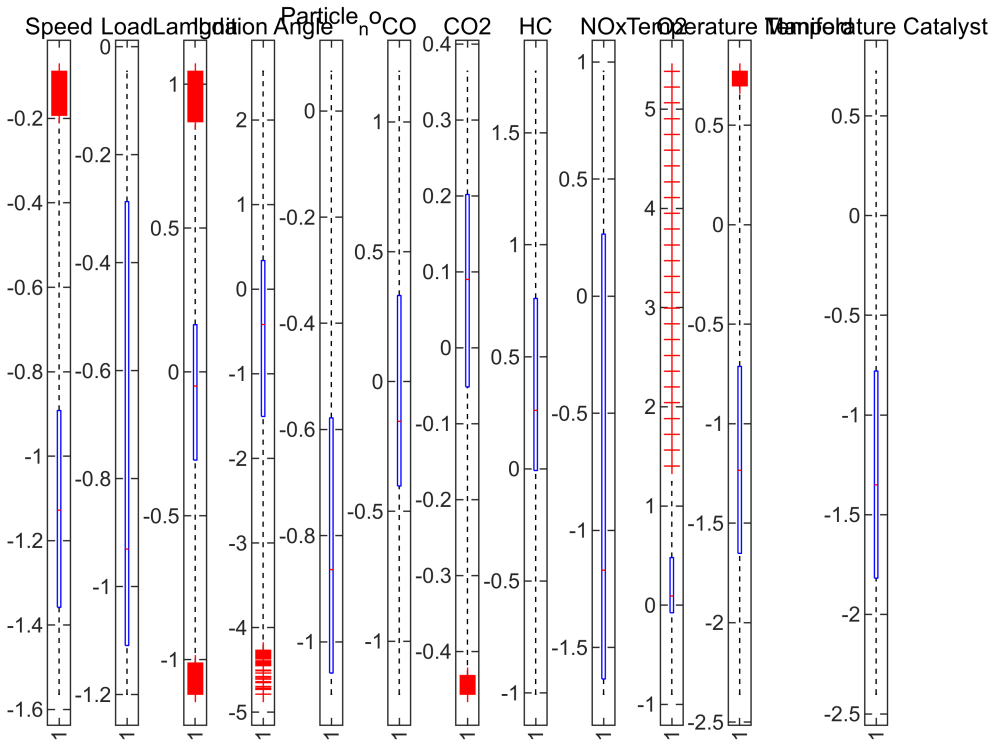
The data revealed a significant presence of outliers during the sensor data acquisition process. To replace these outliers, the **Linear Interpolation** method was used.

```
% Outlier Replacement - Linear interpolation
outlier_eli = [];
for i = 1:length(datasets.Properties.VariableNames)
    temp = filloutliers(table2array(datasets(:, i)), 'linear');
    outlier_eli = [outlier_eli, temp];
end
outlier_eli = array2table(outlier_eli);
outlier_eli.Properties.VariableNames = datasets.Properties.VariableNames;
```

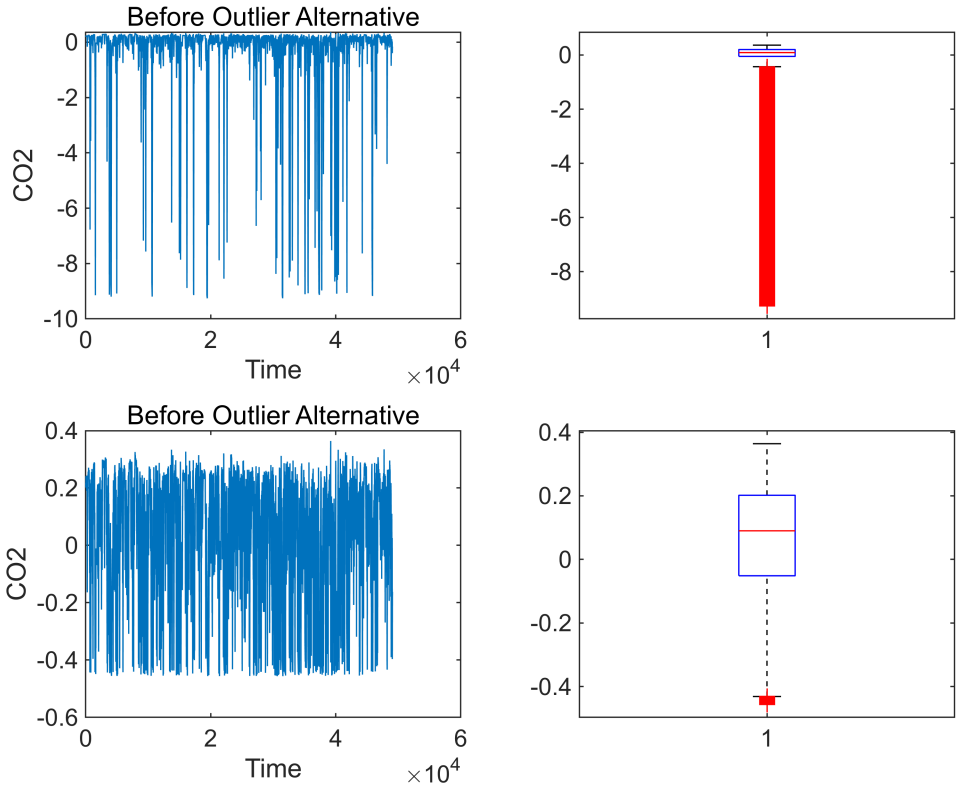
```
% Update
datasets = outlier_eli;
```

```
% After Replacing Outliers
figure;
subplot(1,13,1);
boxplot(datasets.Speed)
title("Speed")
subplot(1,13,2);
boxplot(datasets.Load)
title("Load")
subplot(1,13,3);
boxplot(datasets.Lambda)
title("Lambda")
subplot(1,13,4);
boxplot(datasets.Ignition_angle)
title("Ignition Angle")
subplot(1,13,5);
boxplot(datasets.Particle_no)
title("Particle_no")
subplot(1,13,6);
boxplot(datasets.CO)
title("CO")
subplot(1,13,7);
boxplot(datasets.CO2)
title("CO2")
subplot(1,13,8);
boxplot(datasets.HC)
title("HC")
subplot(1,13,9);
boxplot(datasets.NOx)
title("NOx")
subplot(1,13,10);
boxplot(datasets.O2)
title("O2")
subplot(1,13,11);
```

```
boxplot(datasets.Temp_manifold)
title("Temperature Manifold")
subplot(1,13,13);
boxplot(datasets.Temp_catalyst)
title("Temperature Catalyst")
```




```
% Comparison of Before & After Outlier Alternative
figure;
subplot(2,2,1);
plot(bosch_datasets.CO2)
xlabel("Time");
ylabel("CO2");
title("Before Outlier Alternative");
subplot(2,2,2);
boxplot(bosch_datasets.CO2)
subplot(2,2,3);
plot(datasets.CO2)
xlabel("Time");
ylabel("CO2");
title("Before Outlier Alternative");
subplot(2,2,4);
boxplot(datasets.CO2)
```



Compared to the previous plot, it is evident that the number of outliers has significantly decreased.

2.3. Noise Reduction - Low pass filter

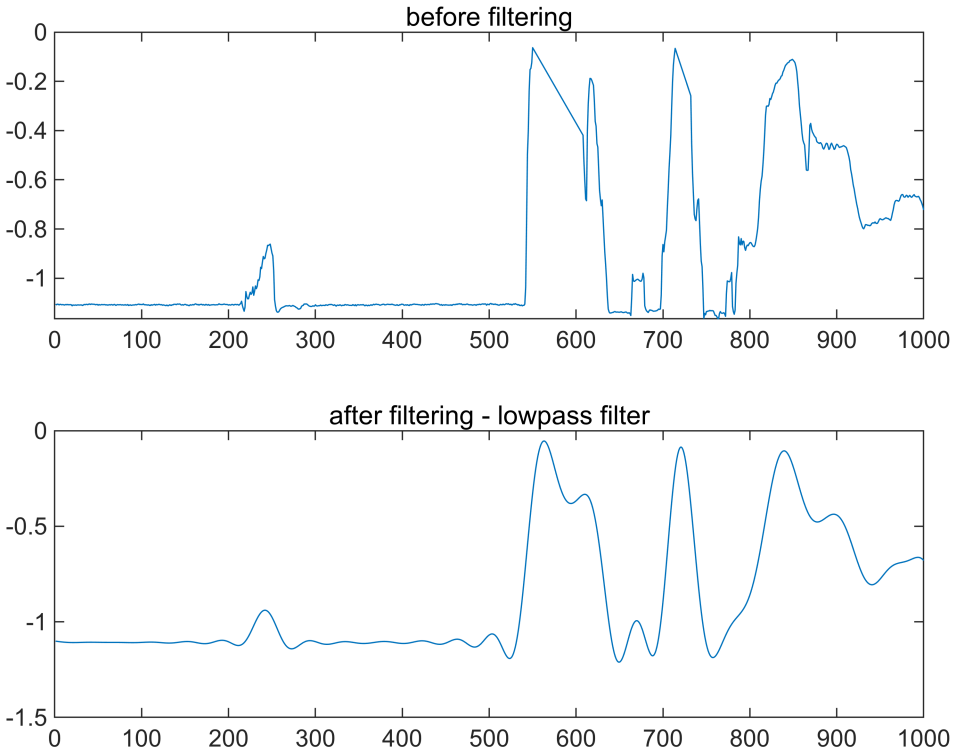
This data had significant noise introduced during acquisition through sensors.

To address this issue, a **Butterworth Lowpass Filter** was applied, which helps reduce the standard deviation of the data by smoothing out high-frequency noise. This filtering process allows the underlying distribution characteristics of the data to become clearer, enhancing the model's ability to learn from the data.

```
% Filter Structure
filtered_datasets = datasets;
variables = datasets.Properties.VariableNames;

d1 = designfilt("lowpassiir",FilterOrder=12, ...
    HalfPowerFrequency=0.05,DesignMethod="butter");

% Apply Butterworth Lowpass Filter

for i = 1:width(datasets)
    filtered_datasets.(variables{i}) = filtfilt(d1, datasets.(variables{i}));
end
```

```
figure;
```

```
subplot(2,1,1);
plot(datasets.Load(1:1000))
title('before filtering')

subplot(2,1,2);
plot(filtered_datasets.Load(1:1000))
title('after filtering - lowpass filter')
```



Compared to the previous plot, after applying the **Butterworth Lowpass Filter**, the standard deviation of the data was significantly reduced, allowing the distribution characteristics of the data to be more clearly defined. This smoothing process helped eliminate high-frequency noise, making the patterns in the variables easier to interpret and thus improving the data's suitability for training.

```
%update
datasets = filtered_datasets;
```

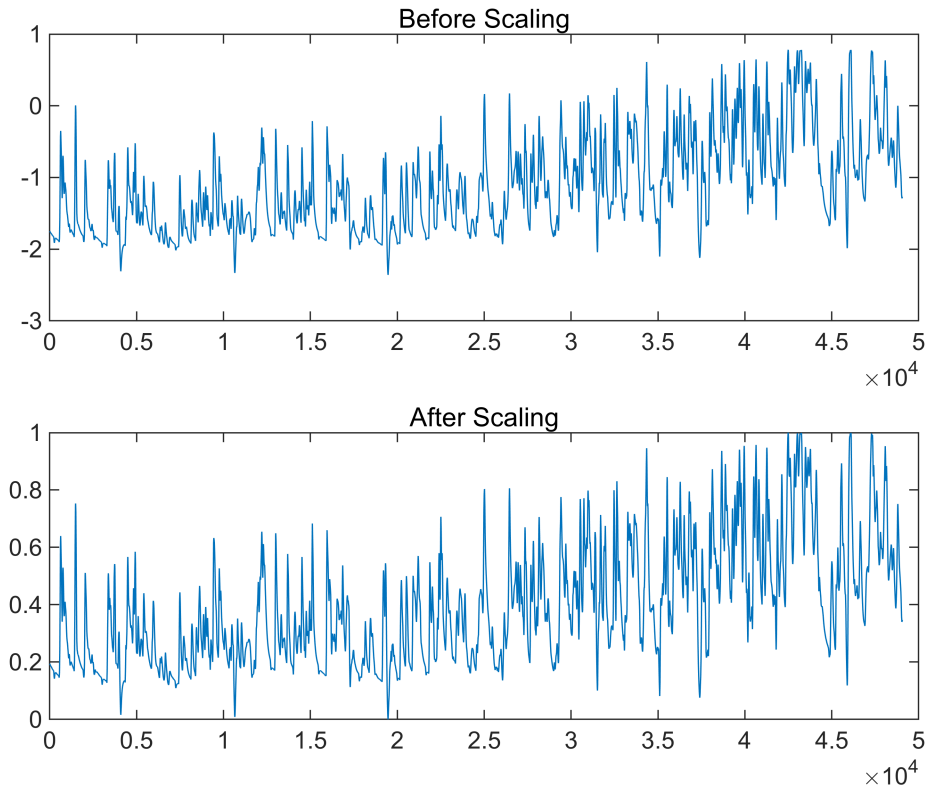
2.4. Min-Max Scaling

As in the previous Baseline Journal, **Min-Max Scaling** was applied. This method normalizes the data by scaling the values of each variable to a range between 0 and 1, ensuring consistency across different units for better comparison and analysis during the modeling process.

```
% Min-Max Scaling
scaled_datasets = datasets;
for i = 1:width(datasets)
```

```
    scaled_datasets.(variables{i}) = normalize(datasets.(variables{i}), 'range');
end
```

```
figure;

subplot(2,1,1);
plot(datasets.Temp_manifold);
title('Before Scaling')

subplot(2,1,2);
plot(scaled_datasets.Temp_manifold);
title('After Scaling')
```




```
%update
datasets = scaled_datasets;
```

2.5. Add Fuel Cutoff Data

```
% Add saved fuel cutoff data
datasets = datasets(:, ~strcmp(datasets.Properties.VariableNames, 'fuel_cutoff'));
added_datasets = [datasets, array2table(fuel_cutoff)];

%update
datasets = added_datasets;
```

```
clear added_datasets;
```

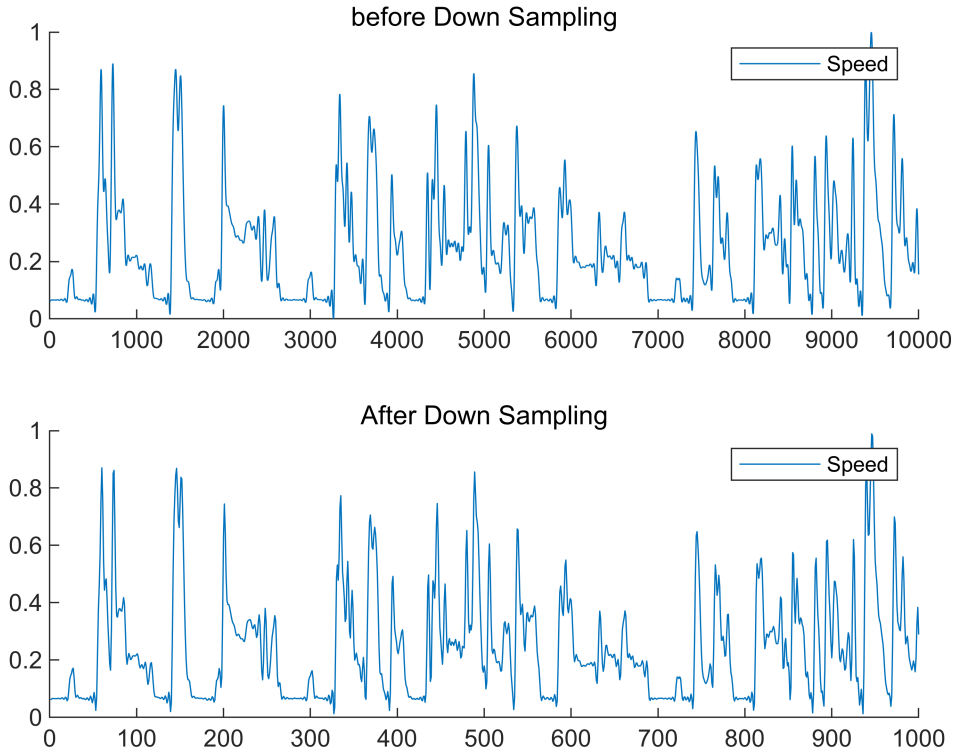
2.6. Down Sampling

The computational complexity of GPR training increases exponentially as the input data grows.

To address this, a down-sampling process was applied, reducing data size while preserving its trends.

```
% Down Sampling
t = 10;
downsampled_datasets = datasets(1:t:end, :);
```

```
% Comparison
figure;

subplot(2,1,1);
hold on;
plot(table2array(datasets((1:10000), 1)))
hold off;
legend(datasets.Properties.VariableNames(1))
title('before Down Sampling')

subplot(2,1,2);
hold on;
plot(table2array(downsampled_datasets((1:1000), 1)))
hold off;
legend(downsampled_datasets.Properties.VariableNames(1))
title('After Down Sampling')
```




```
%update
datasets = downsampled_datasets;
```

3. Classification

3.1. Train/Test Data Split

The **train** and **test datasets** were randomly split in an 80% : 20% ratio. This ensures that 80% of the data is used for model training and 20% is reserved for testing the model's performance on unseen data.

```
cv = cvpartition(height(datasets), 'HoldOut', 0.2);
Trainset = datasets(training(cv), :);
Testset = datasets(test(cv), :);

X_train = [Trainset.Speed, Trainset.Load, Trainset.Lambda, Trainset.Ignition_angle,
Trainset.fuel_cutoff];
Y_train = [Trainset.Particle_no, Trainset.CO, Trainset.CO2, Trainset.HC,
Trainset.NOx, Trainset.O2, Trainset.Temp_manifold, Trainset.Temp_catalyst];

X_test = [Testset.Speed, Testset.Load, Testset.Lambda, Testset.Ignition_angle,
Testset.fuel_cutoff];
Y_test = [Testset.Particle_no, Testset.CO, Testset.CO2, Testset.HC, Testset.NOx,
Testset.O2, Testset.Temp_manifold, Testset.Temp_catalyst];
```

```
% For Python MOGP
% writematrix(X_train, 'X_train.csv');
% writematrix(Y_train, 'Y_train.csv');
% writematrix(X_test, 'X_test.csv');
% writematrix(Y_test, 'Y_test.csv');
```

3.2. Gaussian Process Regression (GPR)

While Matlab supports GPR, it lacks functions for multi-output regression; therefore, separate models were trained for each output variable.

```
% GPR (Basic Option)
% tic
% gprMdl_Particle_no = fitrgp(X_train,Y_train(:,1));
% gprMdl_CO = fitrgp(X_train,Y_train(:,2));
% gprMdl_CO2 = fitrgp(X_train,Y_train(:,3));
% gprMdl_HC = fitrgp(X_train,Y_train(:,4));
% gprMdl_NOx = fitrgp(X_train,Y_train(:,5));
% gprMdl_O2 = fitrgp(X_train,Y_train(:,6));
% gprMdl_Temp_manifold = fitrgp(X_train,Y_train(:,7));
% gprMdl_Temp_catalyst = fitrgp(X_train,Y_train(:,8));
% toc

% GPR (Add Accuracy)
tic
gprMdl_Particle_no = fitrgp(X_train,Y_train(:,1),'Basis','linear',...
      'FitMethod','exact','PredictMethod','exact');
gprMdl_CO = fitrgp(X_train,Y_train(:,2),'Basis','linear',...
      'FitMethod','exact','PredictMethod','exact');
gprMdl_CO2 = fitrgp(X_train,Y_train(:,3),'Basis','linear',...
      'FitMethod','exact','PredictMethod','exact');
gprMdl_HC = fitrgp(X_train,Y_train(:,4),'Basis','linear',...
      'FitMethod','exact','PredictMethod','exact');
gprMdl_NOx = fitrgp(X_train,Y_train(:,5),'Basis','linear',...
      'FitMethod','exact','PredictMethod','exact');
gprMdl_O2 = fitrgp(X_train,Y_train(:,6),'Basis','linear',...
      'FitMethod','exact','PredictMethod','exact');
gprMdl_Temp_manifold = fitrgp(X_train,Y_train(:,7),'Basis','linear',...
      'FitMethod','exact','PredictMethod','exact');
gprMdl_Temp_catalyst = fitrgp(X_train,Y_train(:,8),'Basis','linear',...
      'FitMethod','exact','PredictMethod','exact');
toc
```

```
경과 시간은 132.359309초입니다.
```

GPR is a regression technique that models multiple output variables simultaneously, reflecting correlations and dependencies among them. As an extension of multivariate Gaussian Process, GPR enables each output variable to be learned through interrelationships with other outputs, enhancing prediction accuracy.

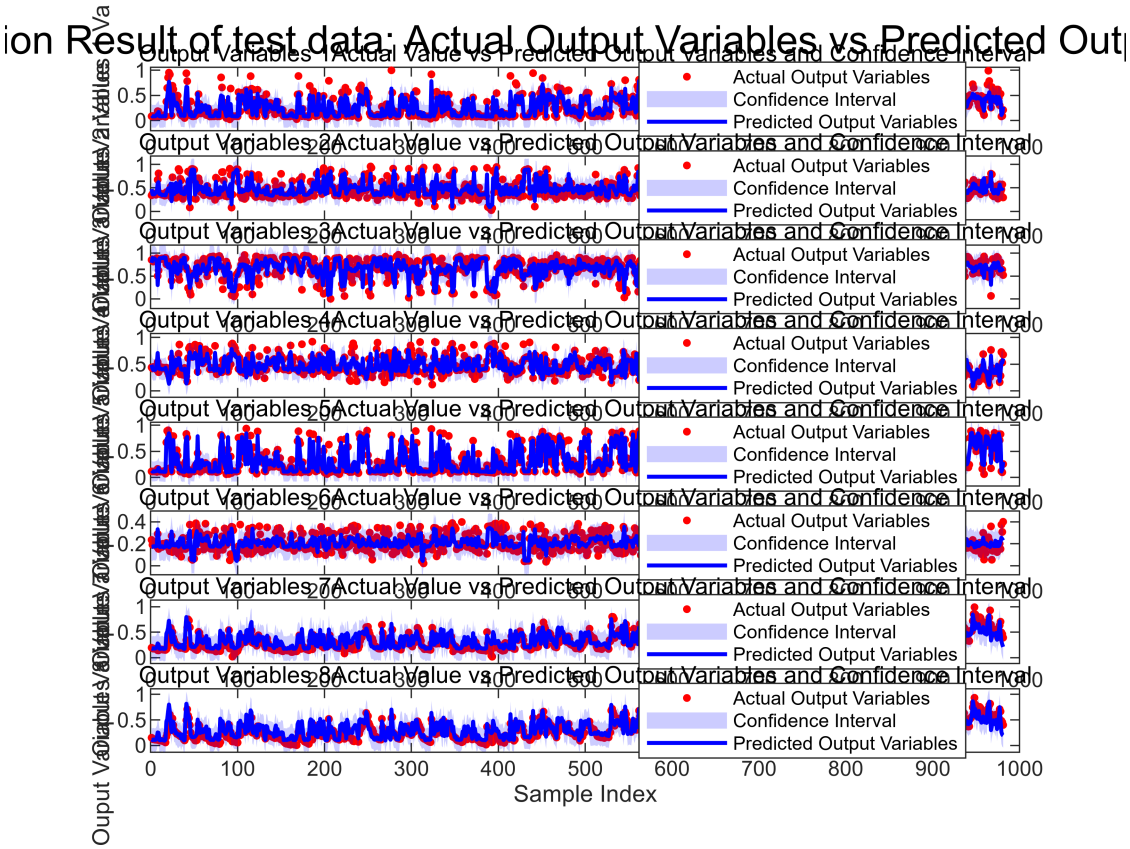
GPR effectively leverages these relationships to optimize prediction performance, particularly when strong correlations or shared patterns exist among output variables.

In contrast, Linear Regression focuses solely on the linear relationship between input variables and each individual output variable. It models each output independently, disregarding correlations or interactions between outputs. As a result, Linear Regression can underperform relative to GPR in situations with strong inter-output dependencies. Furthermore, Linear Regression struggles to account for complex non-linear relationships, limiting its predictive capabilities.

Therefore, when dealing with complex patterns, such as those in engine datasets, GPR is expected to achieve superior performance.

```
% Total Size of Output Variables
num_outputs = size(Y_test, 2);
test_size = size(Y_test, 1);

% Pre-learned GPR models
gprModels = {gprMdl_Particle_no, gprMdl_CO, gprMdl_CO2, gprMdl_HC, gprMdl_NOx,
gprMdl_O2, gprMdl_Temp_manifold, gprMdl_Temp_catalyst};

% Initializing the Prediction Matrix
ypred_test = zeros(size(Y_test));
ystd_test = zeros(size(Y_test));

% Predict the Ouput Variables using Gaussian Process Regression Model with X_test
for i = 1:num_outputs
    [ypred_test(:, i), ystd_test(:, i)] = predict(gprModels{i}, X_test);
end

% Confidence Interval Setting (95% Confidence Interval)
conf_interval = 1.96 * ystd_test;

% Plot about all of the Output Variables
figure;
for i = 1:num_outputs
    subplot(num_outputs, 1, i);

    % Plot the Actual Value
    plot(1:test_size, Y_test(:, i), 'r.', 'MarkerSize', 10);
    hold on;

    % Plot the Confidence Area
    fill([1:test_size, fliplr(1:test_size)], ...
         [ypred_test(:, i) - conf_interval(:, i); flipud(ypred_test(:, i) +
conf_interval(:, i))], ...
         'b', 'FaceAlpha', 0.2, 'EdgeColor', 'none');

    % Plot the Y_prediction
    plot(1:test_size, ypred_test(:, i), 'b-', 'LineWidth', 1.5);
```

```
    xlabel('Sample Index');
    ylabel(['Ouput Variables ' num2str(i) ' Value']);
    legend({'Actual Output Variables', 'Confidence Interval', 'Predicted Output
Variables'}, 'Location', 'Best');
    title(['Output Variables ' num2str(i) 'Actual Value vs Predicted Output
Variables and Confidence Interval']);
    hold off;
end

sgtitle('GPR Prediction Result of test data: Actual Output Variables vs Predicted
Output Variables');
```




```
% RMSE
rmse_values = zeros(1, num_outputs);

for i = 1:num_outputs
    rmse_values(i) = sqrt(mean((Y_test(:, i) - ypred_test(:, i)).^2));
    fprintf('Output Variables %d - RMSE: %.4f\n', i, rmse_values(i));
end
```

```
Output Variables 1 - RMSE: 0.1135
Output Variables 2 - RMSE: 0.1263
Output Variables 3 - RMSE: 0.1529
Output Variables 4 - RMSE: 0.1168
Output Variables 5 - RMSE: 0.1005
Output Variables 6 - RMSE: 0.0642
Output Variables 7 - RMSE: 0.1386
Output Variables 8 - RMSE: 0.1262
```

```
% Average RMSE
average_rmse = mean(rmse_values);
fprintf('\nAverage RMSE: %.4f\n', average_rmse);
```

```
Average RMSE: 0.1174
```

Linear Regression, Random Forest, SVM (Support Vetor Machine)

To benchmark GPR's performance, three other models were selected as reference points.

```
% Compare with other models
num_outputs = size(Y_train, 2);

models = struct();
models.LinearRegression = @(X, Y) fitlm(X, Y);
models.RandomForest = @(X, Y) TreeBagger(100, X, Y, 'Method', 'regression');
models.SVM = @(X, Y) fitrsvm(X, Y, 'KernelFunction', 'linear');

rmse_values = struct();

for model_name = fieldnames(models)'
    model_name = model_name{1};
    rmse_values.(model_name) = zeros(1, num_outputs);

    for i = 1:num_outputs
        model = models.(model_name)(X_train, Y_train(:, i));
        y_pred = predict(model, X_test);

        rmse_values.(model_name)(i) = sqrt(mean((Y_test(:, i) - y_pred).^2));
        fprintf('%s Model - Output Variables %d - RMSE: %.4f\n', model_name, i,
rmse_values.(model_name)(i));
    end
end
```

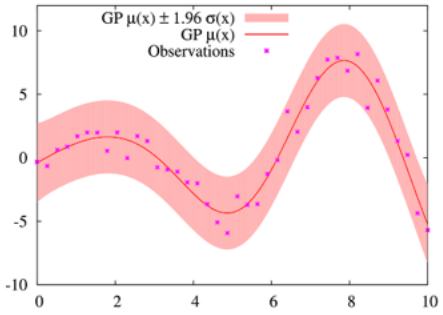
```
LinearRegression Model - Output Variables 1 - RMSE: 0.1347
LinearRegression Model - Output Variables 2 - RMSE: 0.1529
LinearRegression Model - Output Variables 3 - RMSE: 0.1888
LinearRegression Model - Output Variables 4 - RMSE: 0.1388
LinearRegression Model - Output Variables 5 - RMSE: 0.1249
LinearRegression Model - Output Variables 6 - RMSE: 0.0750
LinearRegression Model - Output Variables 7 - RMSE: 0.1606
LinearRegression Model - Output Variables 8 - RMSE: 0.1482
RandomForest Model - Output Variables 1 - RMSE: 0.1162
RandomForest Model - Output Variables 2 - RMSE: 0.1259
RandomForest Model - Output Variables 3 - RMSE: 0.1506
RandomForest Model - Output Variables 4 - RMSE: 0.1177
RandomForest Model - Output Variables 5 - RMSE: 0.1028
RandomForest Model - Output Variables 6 - RMSE: 0.0650
RandomForest Model - Output Variables 7 - RMSE: 0.1393
RandomForest Model - Output Variables 8 - RMSE: 0.1264
SVM Model - Output Variables 1 - RMSE: 0.1355
SVM Model - Output Variables 2 - RMSE: 0.1565
SVM Model - Output Variables 3 - RMSE: 0.1916
SVM Model - Output Variables 4 - RMSE: 0.1422
SVM Model - Output Variables 5 - RMSE: 0.1259
```

```
SVM Model - Output Variables 6 - RMSE: 0.0770
SVM Model - Output Variables 7 - RMSE: 0.1615
SVM Model - Output Variables 8 - RMSE: 0.1493
```

```
for model_name = fieldnames(rmse_values)'
    model_name = model_name{1};
    avg_rmse = mean(rmse_values.(model_name));
    fprintf('%s - Average RMSE: %.4f\n', model_name, avg_rmse);
end
```

```
LinearRegression - Average RMSE: 0.1405
RandomForest - Average RMSE: 0.1180
SVM - Average RMSE: 0.1424
```

4. Result & Discussion



* <Gaussian Process Regression>

Non-parametric, probabilistic regression method that learns the **distribution** of functions.

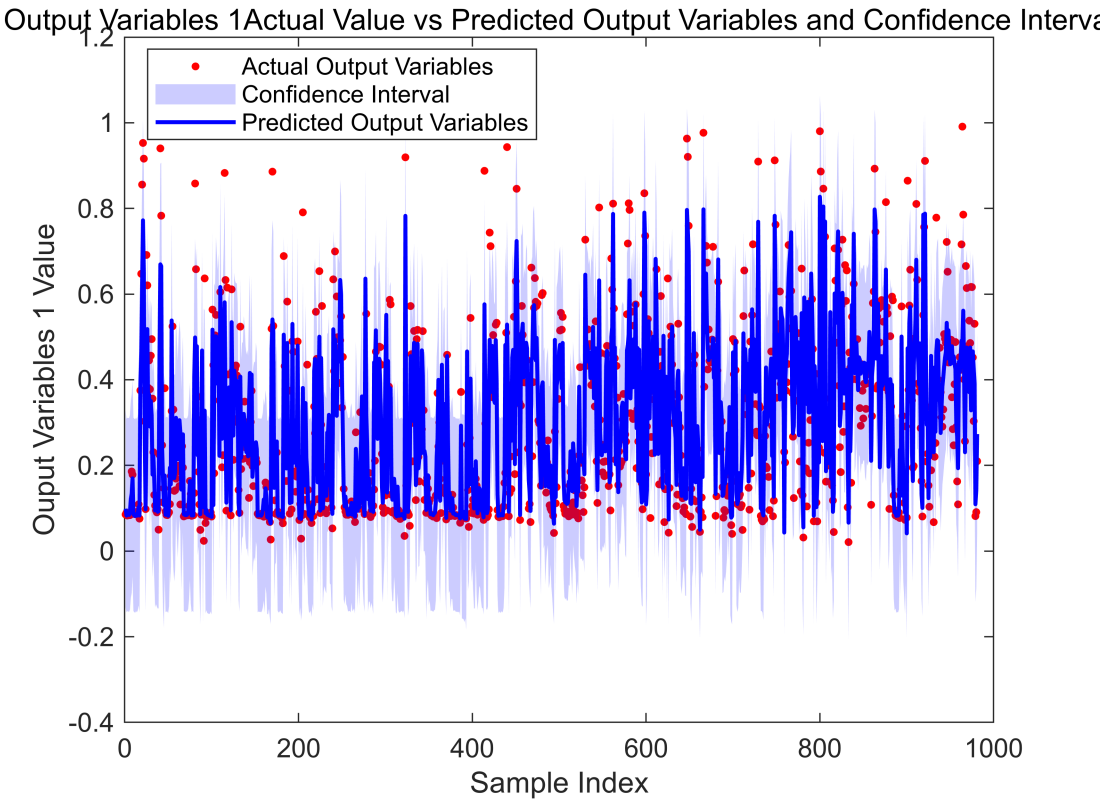
Measures the **correlation** (similarity) between data points.

Data points that are **closer** to each other have **higher similarity**.

Provides **uncertainty estimates** for the data, allowing evaluation of predictions along with **confidence intervals**.

```
figure;

% Plot the Actual Value
plot(1:test_size, Y_test(:, 1), 'r.', 'MarkerSize', 10);
hold on;

% Plot the Confidence Area
fill([1:test_size, fliplr(1:test_size)], ...
     [ypred_test(:, 1) - conf_interval(:, 1); flipud(ypred_test(:, 1) +
conf_interval(:, 1))], ...
     'b', 'FaceAlpha', 0.2, 'EdgeColor', 'none');

% Plot the Y_prediction
plot(1:test_size, ypred_test(:, 1), 'b-', 'LineWidth', 1.5);

xlabel('Sample Index');
ylabel(['Ouput Variables ' num2str(1) ' Value']);
legend({'Actual Output Variables', 'Confidence Interval', 'Predicted Output
Variables'}, 'Location', 'Best');
```

```
title(['Output Variables ' num2str(1) 'Actual Value vs Predicted Output Variables
and Confidence Interval']);
hold off;
```



Gaussian Process Regression (GPR) is a non-parametric, probabilistic regression method that predicts outputs based on a distribution over possible functions, offering uncertainty estimates in predictions through a covariance structure defined by a chosen kernel.

Gaussian Process Regression (GPR), particularly in the form of Multi-Output Gaussian Processes (MOGP), can outperform other regression models on complex datasets like the BOSCH Engine Datasets due to its ability to capture the underlying relationships between multiple correlated output variables. In the BOSCH Engine Datasets, output variables such as particle numbers, CO, CO2, and other emissions are not independent; they exhibit correlations influenced by shared input variables like engine speed, load, and fuel cutoff. MOGP leverages these interdependencies by modeling the joint distribution of outputs, rather than treating each output independently, as is common in traditional regression models. This approach allows MOGP to use information from one output to inform the predictions of others, potentially improving accuracy.

Moreover, GPR and MOGP provide a probabilistic framework that includes uncertainty estimation, which is valuable in identifying prediction confidence levels, especially for data points in regions with less information. This feature can enhance reliability in applications requiring robust predictions, like fault detection in engine systems. The flexibility of GPR's kernel-based approach also allows for the modeling of non-linear relationships, which is often more accurate than simpler linear models. Therefore, when applied to the BOSCH Engine Datasets, MOGP can potentially yield lower RMSE values and more accurate predictions than linear regression or other simpler regression methods.

MATLAB	GPR	Linear Regression	Random Forest	SVM
RMSE	**0.1174**	0.1405	0.1180	0.1424

In Gaussian Process Regression (GPR) or Multi-Output Gaussian Process (MOGP), the high computational cost primarily arises from calculating the kernel matrix. GPR requires calculating the kernel function for every pair of training samples, resulting in an **N X N** kernel matrix, where **N** is the number of training samples. During prediction, the model needs to invert this matrix, which has a time complexity of **O(N^3)**. This computational demand increases drastically with more data. For MOGP, the computational cost is even higher as it models the relationships across multiple outputs, leading to a more complex kernel matrix.

When more data is used for training, the model benefits by learning richer patterns that reflect various conditions, especially in complex systems like engines. With additional data, the model can better capture the relationships and patterns between output variables, leading to improved prediction accuracy. Moreover, GPR or MOGP trained on abundant data can provide more reliable uncertainty estimates, enhancing the model's prediction reliability on new data.

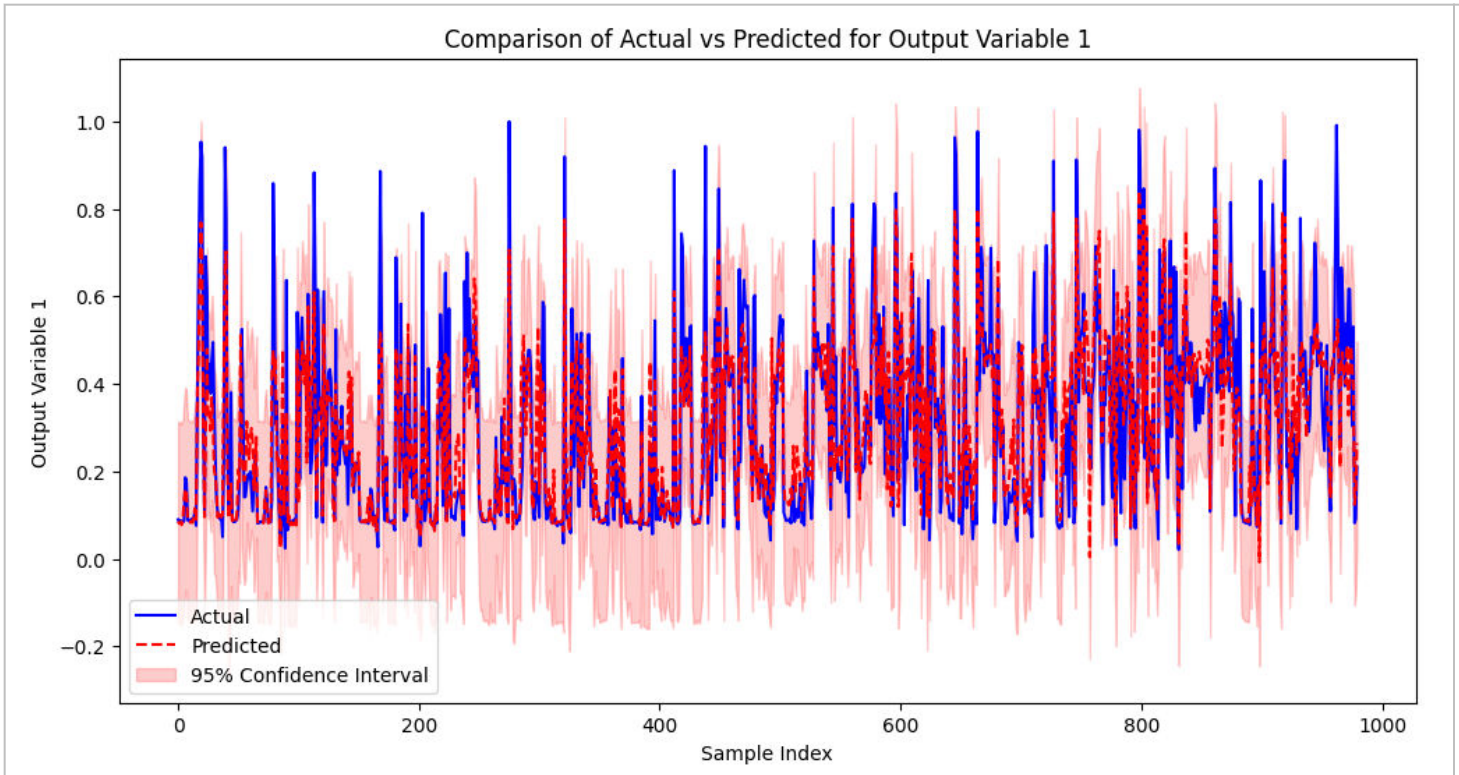
Thus, while computational cost is a drawback in GPR and MOGP, training on a sufficient amount of data significantly improves model accuracy and predictive confidence, yielding more precise results in complex systems.

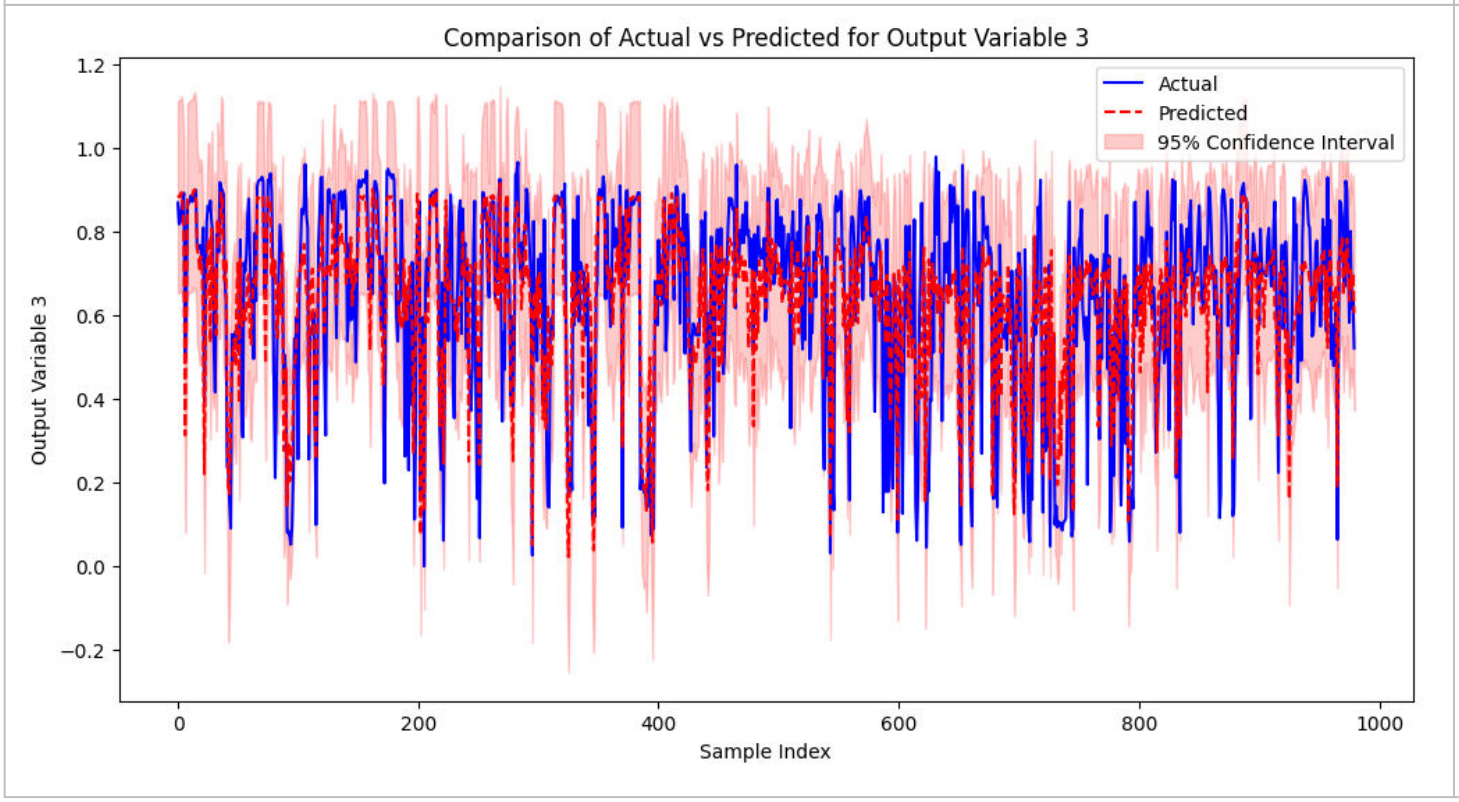
Python Result

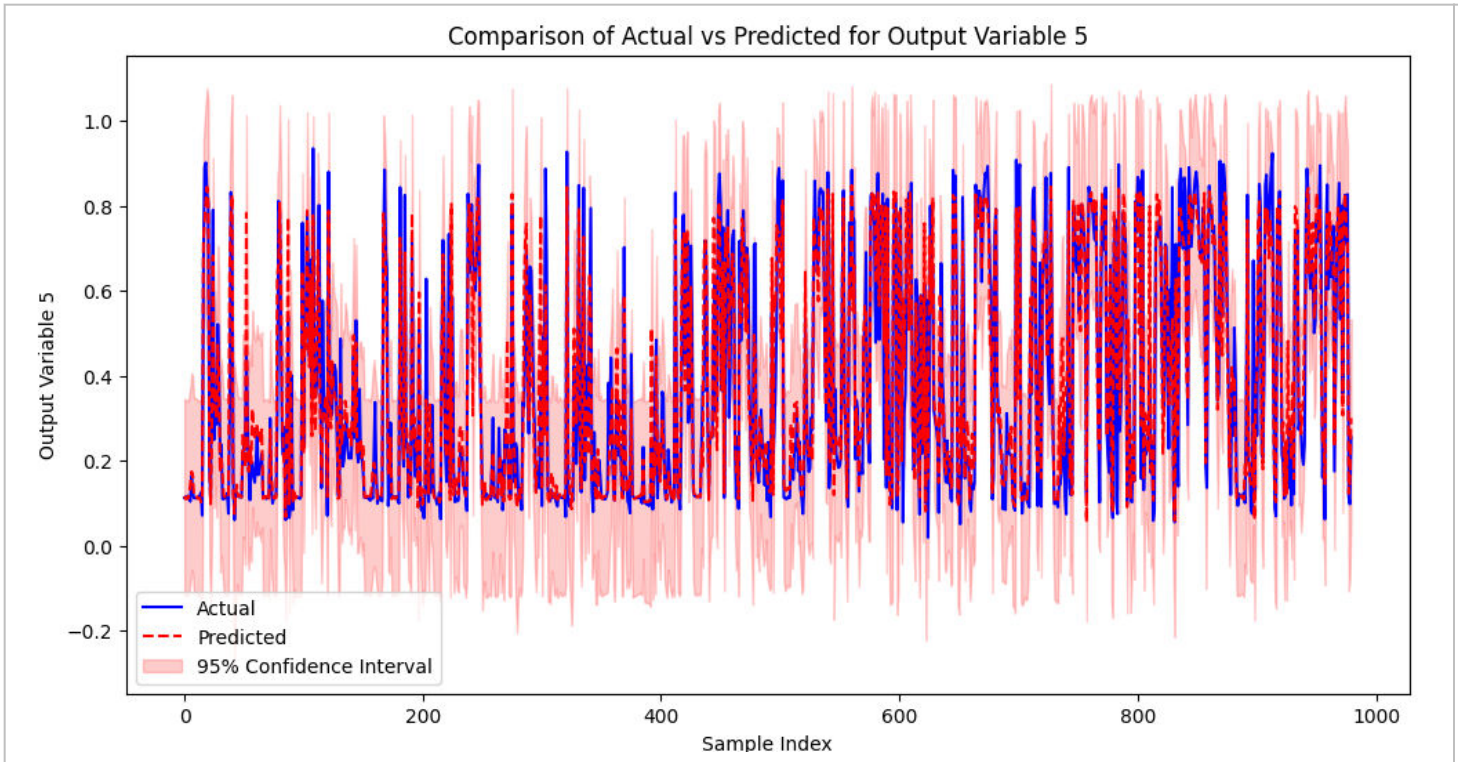
MATLAB does not support Multi-Output Gaussian Process functions, so MOGP was implemented using the GPy module in Python.

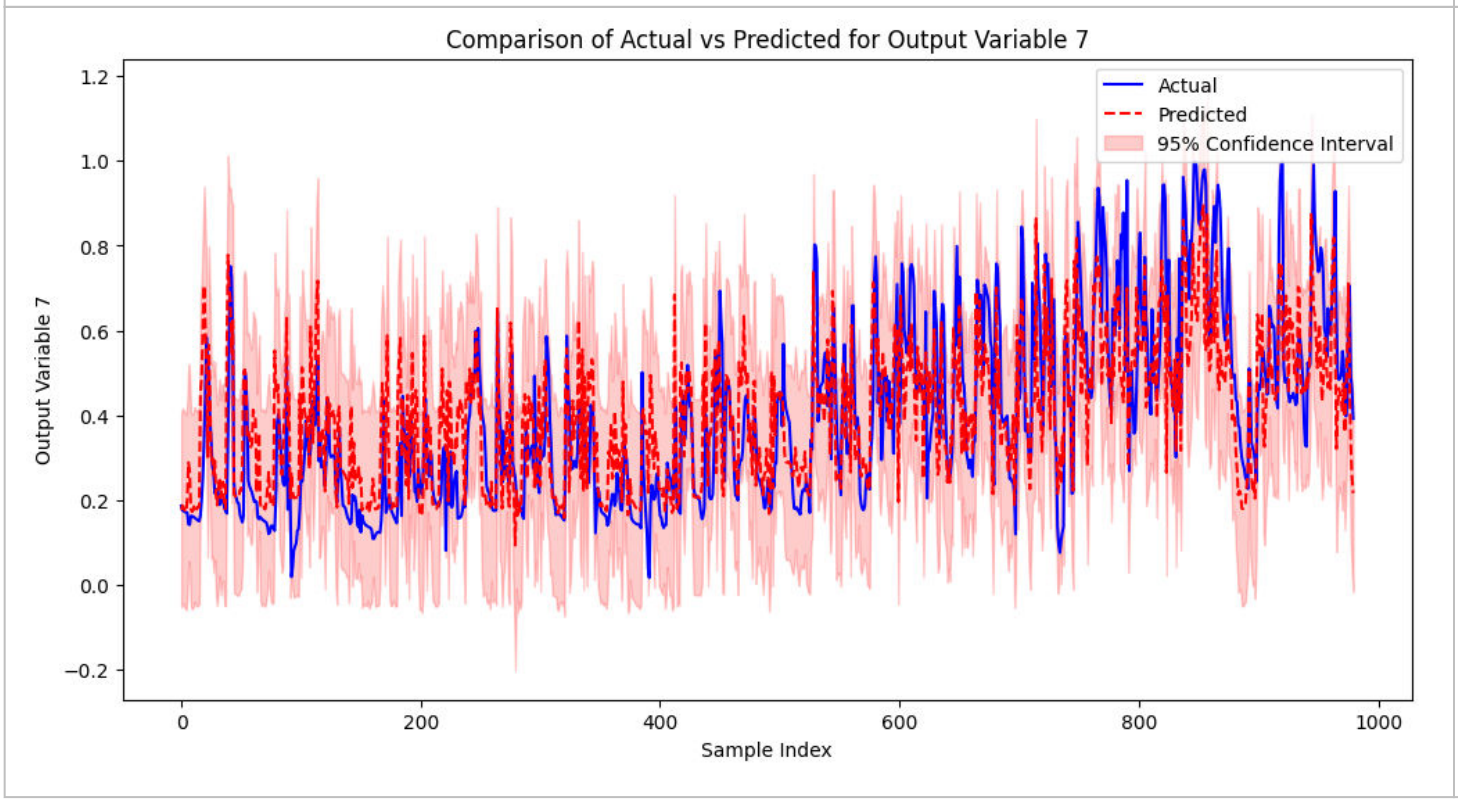
As a result, the RMSE value did not differ significantly from that in MATLAB.

Python	MOGP	Linear Regression	Random Forest	SVM
RMSE	**0.1188**	0.1405	0.1198	0.1234


Comparison of Actual vs Predicted for Output Variable 1
Output Variable 1
Sample Index
Actual
Predicted
95% Confidence Interval
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
200
400
600
800
1000


Comparison of Actual vs Predicted for Output Variable 3
Output Variable 3
Sample Index
Actual
Predicted
95% Confidence Interval
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
200
400
600
800
1000


Comparison of Actual vs Predicted for Output Variable 5
Output Variable 5
Sample Index
Actual
Predicted
95% Confidence Interval
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
200
400
600
800
1000


Comparison of Actual vs Predicted for Output Variable 7
Output Variable 7
Sample Index
Actual
Predicted
95% Confidence Interval
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
200
400
600
800
1000