CS 454
Introduction to Machine Learning and Artificial Neural Networks

Homework 2 Report

Nonparametric Regression

Author	*Uygar KAYA*
Instructor	*Prof. Ethem ALPAYDIN*
Submission Date	*11.04.2022*

1 Introduction

In this assignment, we implement a Nonparametric Regression, and we see how the best model selection can be done.

In this assignment, we have one Test data set with 100 instances and 10 Training data set files each containing 25 instances. In each data set, the first column in each row is the Input - X and the second column is the desired Output - R.

1.1 Nonparametric Regression

Nonparametric regression is a type of regression analysis in which the predictor does not take a fixed shape but is built based on data input. That is, no parametric form for the link between predictors and dependent variable is assumed.

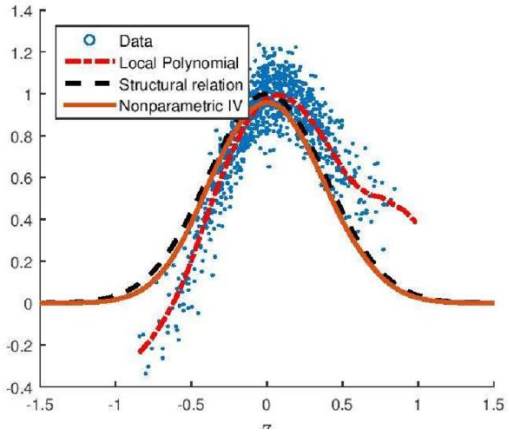


Figure 1: Example of Nonparametric Regression [1]

2 Methodology

In this assignment, we apply a Nonparametric Regression model methodology using Python Programming Language.

During this assignment, using only 4 different Python programming language libraries were used these are;

1. *Pandas*
2. *Scikit-Learn*
3. *Matplotlib*
4. *Math*

Pandas library is used to make a DataFrame. I used to Scikit-Learn library in order to use the mean_square_error function. I used the Matplotlib in order to plot and visualized the result data. Finally, I used the Math library to perform math operations.

3 Implementation Details

In this assignment, firstly we import the necessary libraries which are Pandas, Math, Matplotlib and Scikit-Learn. After importing these libraries, we read the csv file with the help of the pandas library in Python and we have 10 sample training data .csv file & 1 testing data .csv file and after reading the 10 sample training data, I append these data to the list.

Importing Required Libraries

```
import pandas as pd
import math

from matplotlib import pyplot as plt
from sklearn.metrics import mean_squared_error
```

[1] Python

Reading Train & Test Files

```
trainDataFrame = []

for trainDataIndex in range(1, 11):
    data = pd.read_csv(f'TrainingData/sample{trainDataIndex}.csv', names=['X', 'R'],  header=None)
    trainDataFrame.append(data)
```

[2] Python

```
testDataFrame = pd.read_csv('TestingData/test.csv', names=['X', 'R'], header=None)
```

[3] Python

Figure 2: Libraries and Reading Training & Test Files

After reading the training & testing data, I separate the Input – X & Output – R list for each testing and training data. I separate the input and output data in order to use these values in the calculation for the kernel smoother and etc.

Seperate Input - X & Output - R

```
train_x = []
train_r = []

for index in range(10):
    train_x.append(trainDataFrame[index]['X'].values)
    train_r.append(trainDataFrame[index]['R'].values)
```

[4] Python

```
test_x = testDataFrame['X'].values
test_r = testDataFrame['R'].values
```

[5] Python

Figure 3: Separate the Data to Input & Output Data

After this process, I plot the whole training data and testing data in order to see the visualize distribution for the data.

Ploting the Whole Training & Testing Data

```
def plotingData(data_x, data_r, title, color):
    plt.title(title, fontweight="bold", style="italic")
    plt.xlabel("Input - X", fontweight="bold", style="italic")
    plt.ylabel("Output - R", fontweight="bold", style="italic")
    plt.scatter(data_x, data_r, color=color)
    plt.show()
```

[6] Python

```
plotingData(train_x, train_r, 'Training Data', 'g')
plotingData(test_x, test_r, 'Testing Data', 'b')
```

[7] Python

Figure 4: Plotting the Existing Whole Data

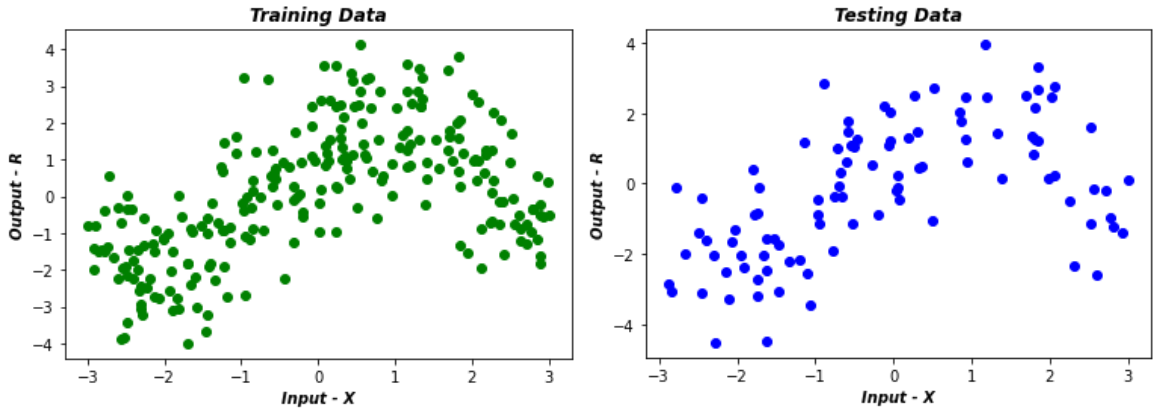


Figure 5: Visualizing Whole Training & Testing Data

After these steps, I coded the kernel smoother function in order to find the estimate x. I used Gaussian kernel for Kernel Function - K(u)

$$\hat{g}(x) = \frac{\sum_{t=1}^{N} K\left(\frac{x - x^t}{h}\right) r^t}{\sum_{t=1}^{N} K\left(\frac{x - x^t}{h}\right)} \qquad K(u) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{u^2}{2}\right]$$

Gaussian Kernel & Kernel Smoother Function

```
def gaussian_kernel(u):
    return 1/math.sqrt(2*math.pi) * math.exp(-(u**2)/2)
```

[8] Python

```
def kernel_smoother(test_x, train_x, train_r, h):
    kernelSmootherResult = []
    for tx in test_x:
        numerator = 0
        denominator = 0
        for index in range(len(train_x)):
            numerator += gaussian_kernel((tx-train_x[index])/h)*train_r[index]
            denominator += gaussian_kernel((tx-train_x[index])/h)
        kernelSmootherResult.append(numerator/denominator)
    return kernelSmootherResult
```

[9] Python

Figure 6: Gaussian Kernel & Kernel Smoother Function

3.1 Applying Nonparametric Regression Model Strategy

3.1.1 Kernel Smoothing of h value 0.05

In this part, I try the h values of degree 0.05, 0.1, 0.25, 0.5, 1, and 5 to the ten data sets separately using Kernel Smoother function. For each h value, I have ten different models for the ten training sets. I plot these ten models for each h value, and I calculate their mean square error values on the test set for each h value.

Kernel Smoothing of h value 0.05

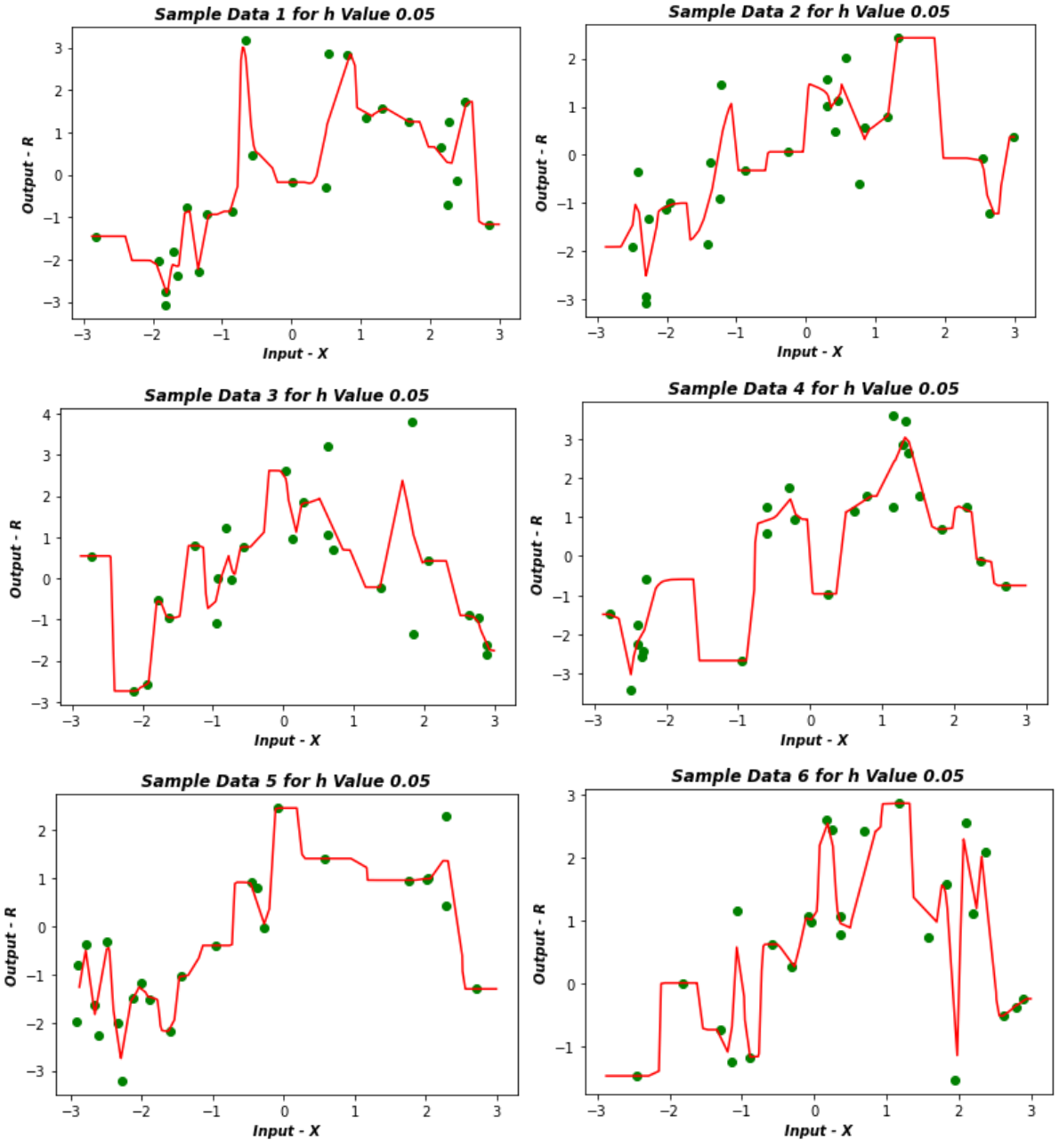
```
kernelSmoother_005 = []

for index in range(10):
    result = kernel_smoother(test_x, train_x[index], train_r[index], 0.05)
    kernelSmoother_005.append(result)
```

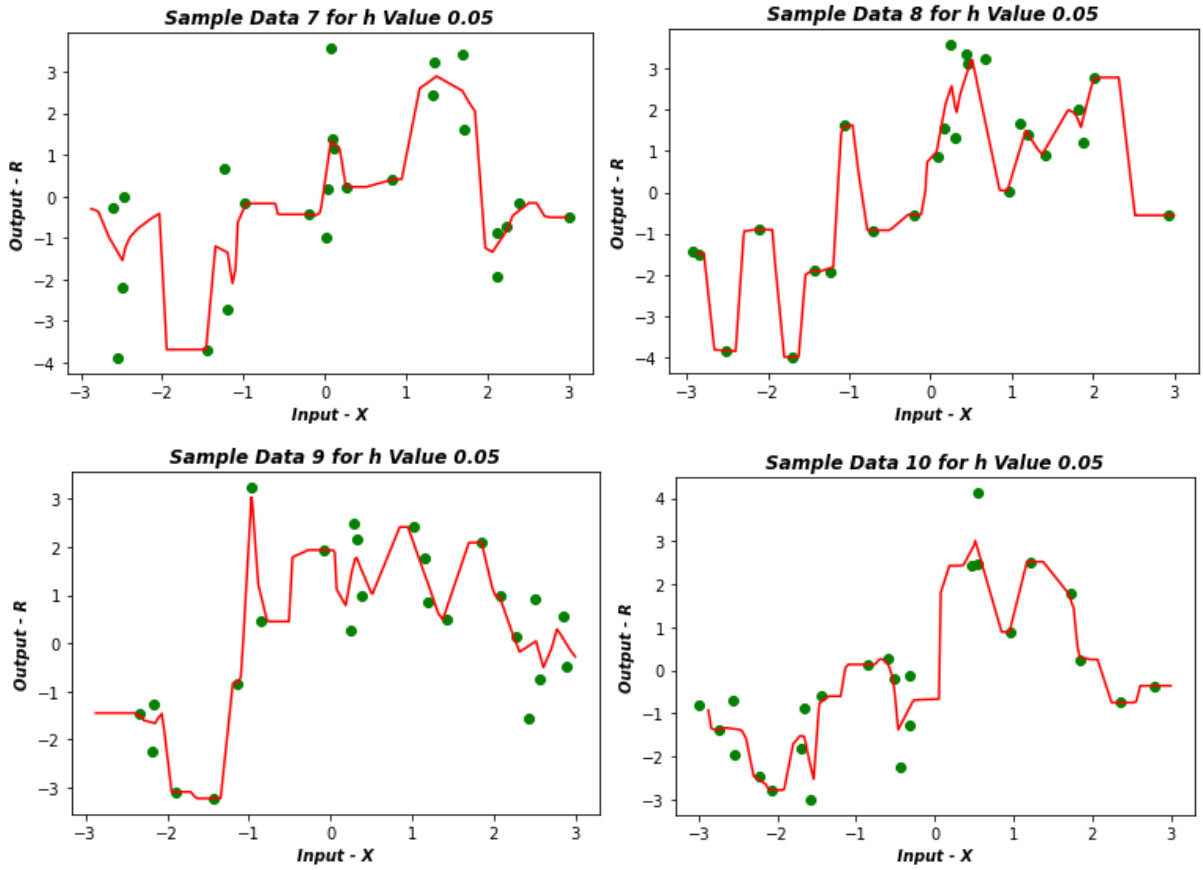
[10] Python

```
for index in range(10):
    plt.title(f'Sample Data {index+1} for h Value 0.05', fontweight="bold", style="italic")
    plt.xlabel("Input - X", fontweight="bold", style="italic")
    plt.ylabel("Output - R", fontweight="bold", style="italic")
    plt.scatter(train_x[index], train_r[index], color = 'g')
    plt.plot(test_x, kernelSmoother_005[index], color = 'r')
    plt.show()
```

[11] Python

Figure 7: Kernel Smoothing of h value 0.05





After Calculated the Kernel Smoother function for h value 0.05, I use the mean_squared_error function from the Scikit-Learn and I calculate the MSE value for each 10 model for h value 0.05.

Calculating Mean Squared Error for Each Sample

```
MeanSquaredError005 = []

for index in range(10):
    mse = mean_squared_error(test_r, kernelSmoother_005[index])
    MeanSquaredError005.append(mse)
```

[12] Python

```
for index, mse in enumerate(MeanSquaredError005):
    print(f'MSE for Sample {index+1} & h value 0.05: {mse}')
```

[13] Python

```
MSE for Sample 1 & h value 0.05: 2.239243757110487
MSE for Sample 2 & h value 0.05: 2.1496154655878574
MSE for Sample 3 & h value 0.05: 2.735620020103718
MSE for Sample 4 & h value 0.05: 2.4056606522357047
MSE for Sample 5 & h value 0.05: 1.9678604870381204
MSE for Sample 6 & h value 0.05: 2.5512667342926725
MSE for Sample 7 & h value 0.05: 2.87554128617407
MSE for Sample 8 & h value 0.05: 3.615801106970661
MSE for Sample 9 & h value 0.05: 2.317248136919054
MSE for Sample 10 & h value 0.05: 2.2296223365721555
```

Figure 8: h value 0.05 MSE for Each Model

After making the MSE calculation for each model, I calculate the average mean square error for h value's 0.05 model.

Calculating Avarage Mean Squared Error for h value 0.05

```
total = 0
length = len(MeanSquaredError005)

for index in range(10):
    total += MeanSquaredError005[index]

averageMSE_005 = total/length
print('Avarage MSE for h value 0.05:', averageMSE_005)
```

[14] Python

```
Avarage MSE for h value 0.05: 2.5087479983004504
```

Figure 9: Average MSE for h value 0.05

I repeat all the steps I have applied above for each h values.

3.1.2 Kernel Smoothing of h value 0.1

Kernel Smoothing of h value 0.1

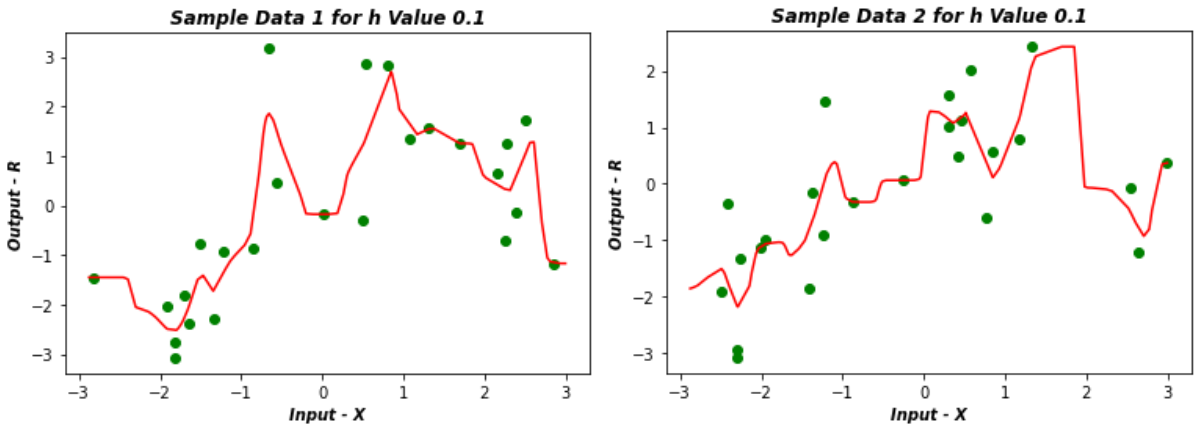
```
kernelSmoother_01 = []

for index in range(10):
    result = kernel_smoother(test_x, train_x[index], train_r[index], 0.1)
    kernelSmoother_01.append(result)
```

[15] Python

```
for index in range(10):
    plt.title(f'Sample Data {index+1} for h Value 0.1', fontweight="bold", style="italic")
    plt.xlabel("Input - X", fontweight="bold", style="italic")
    plt.ylabel("Output - R", fontweight="bold", style="italic")
    plt.scatter(train_x[index], train_r[index], color = 'g')
    plt.plot(test_x, kernelSmoother_01[index], color = 'r')
    plt.show()
```

[16] Python

Figure 10: Kernel Smoothing of h value 0.1



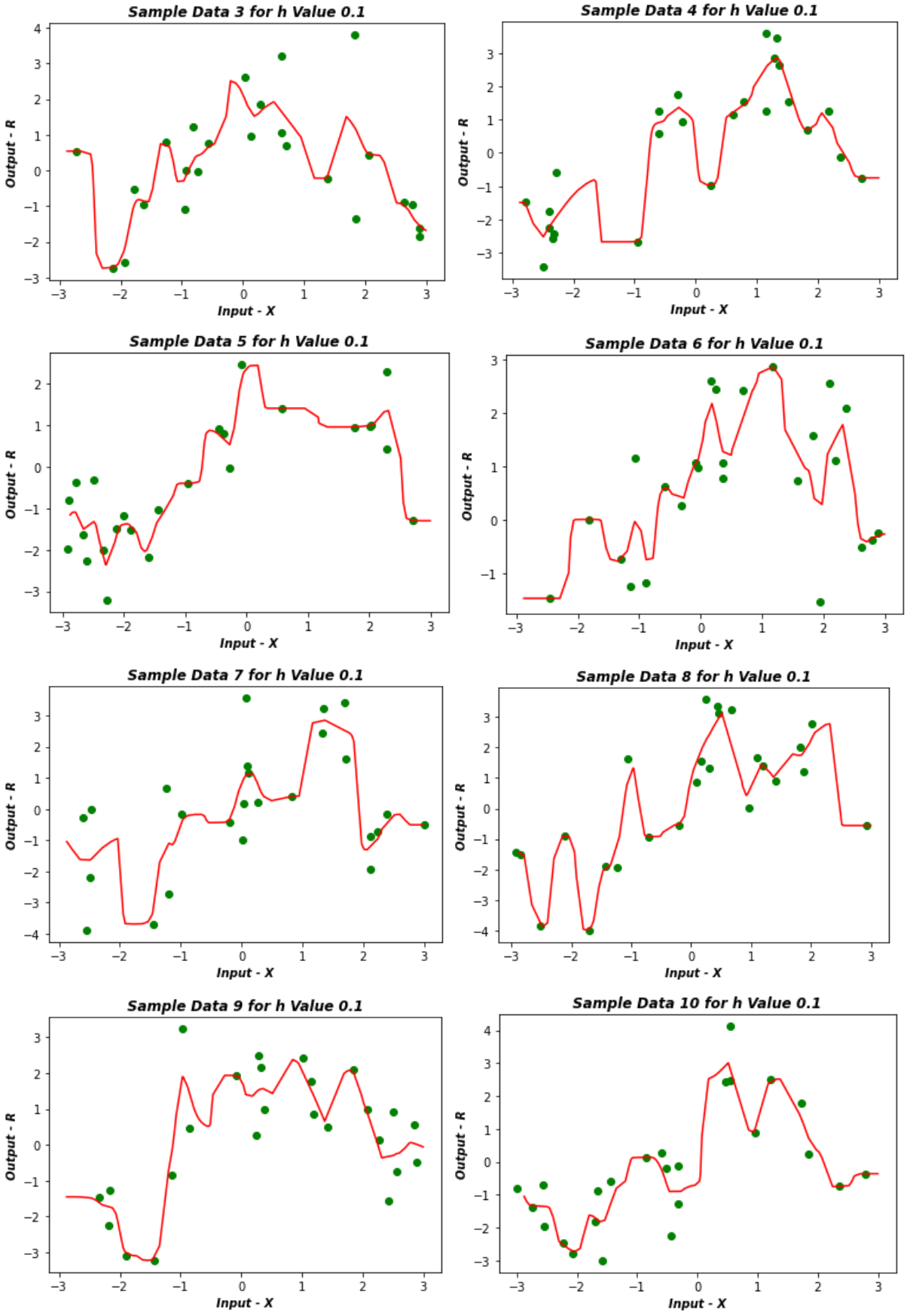


Figure 11: Plotting the result of h value 0.1

Calculating Mean Squared Error for Each Sample

```
MeanSquaredError01 = []

for index in range(10):
    mse = mean_squared_error(test_r, kernelSmoother_01[index])
    MeanSquaredError01.append(mse)
```

[17] Python

```
for index, mse in enumerate(MeanSquaredError01):
    print(f'MSE for Sample {index+1} & h value 0.1: {mse}')
```

[18] Python

```
MSE for Sample 1 & h value 0.1: 1.9463208489861115
MSE for Sample 2 & h value 0.1: 2.0562267158582044
MSE for Sample 3 & h value 0.1: 2.6346877202598176
MSE for Sample 4 & h value 0.1: 2.1838292463168014
MSE for Sample 5 & h value 0.1: 1.86686625068704
MSE for Sample 6 & h value 0.1: 2.5292208609555527
MSE for Sample 7 & h value 0.1: 2.4737205784875913
MSE for Sample 8 & h value 0.1: 3.2614362735140685
MSE for Sample 9 & h value 0.1: 2.161173227762924
MSE for Sample 10 & h value 0.1: 2.143475159368925
```

Calculating Avarage Mean Squared Error for h value 0.1

```
total = 0
length = len(MeanSquaredError01)

for index in range(10):
    total += MeanSquaredError01[index]

averageMSE_01 = total/length
print('Avarage MSE for h value 0.1:', averageMSE_01)
```

[19] Python

```
Avarage MSE for h value 0.1: 2.325695688219704
```

Figure 12: MSE and Average MSE for h value 0.1

3.1.3 Kernel Smoothing of h value 0.25

Kernel Smoothing of h value 0.25

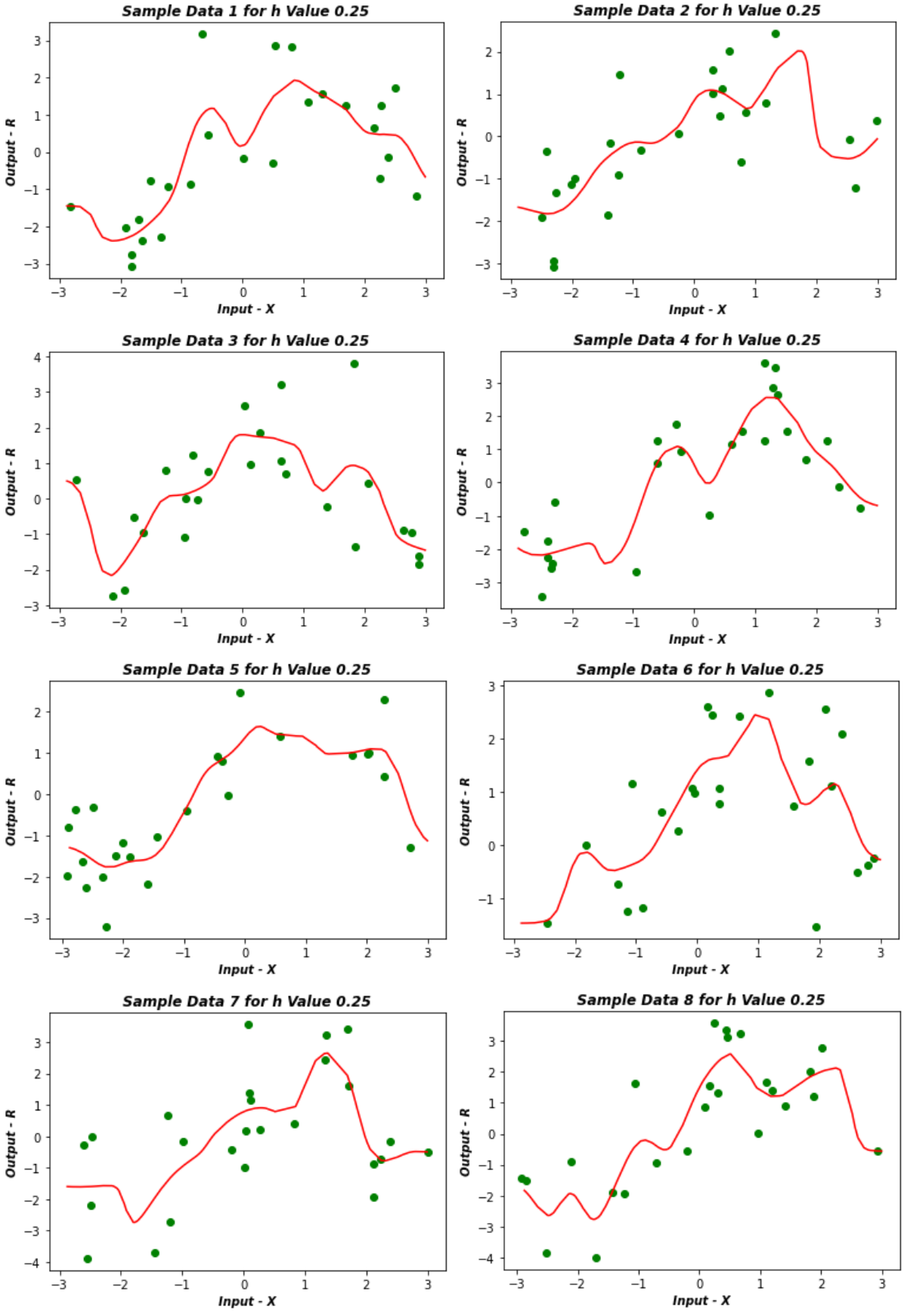
```
kernelSmoother_025 = []

for index in range(10):
    result = kernel_smoother(test_x, train_x[index], train_r[index], 0.25)
    kernelSmoother_025.append(result)
```

[20] Python

```
for index in range(10):
    plt.title(f'Sample Data {index+1} for h Value 0.25', fontweight="bold", style="italic")
    plt.xlabel("Input - X", fontweight="bold", style="italic")
    plt.ylabel("Output - R", fontweight="bold", style="italic")
    plt.scatter(train_x[index], train_r[index], color = 'g')
    plt.plot(test_x, kernelSmoother_025[index], color = 'r')
    plt.show()
```

[21] Python

Figure 13: Kernel Smoothing of h value 0.25


Sample Data 1 for h Value 0.25
Sample Data 2 for h Value 0.25
Sample Data 3 for h Value 0.25
Sample Data 4 for h Value 0.25
Sample Data 5 for h Value 0.25
Sample Data 6 for h Value 0.25
Sample Data 7 for h Value 0.25
Sample Data 8 for h Value 0.25
Output - R
Input - X

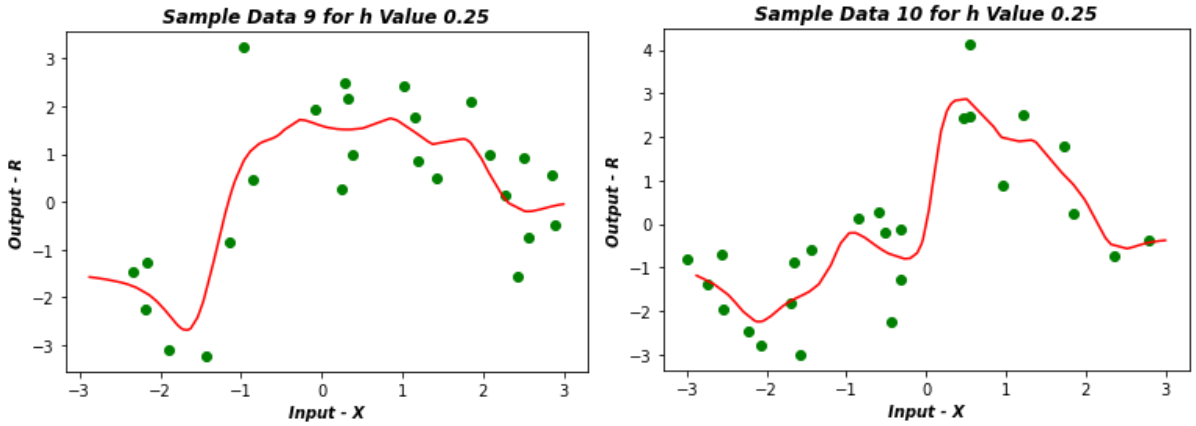


Figure 14: Plotting the result of h value 0.25

Calculating Mean Squared Error for Each Sample

```
MeanSquaredError025 = []

for index in range(10):
    mse = mean_squared_error(test_r, kernelSmoother_025[index])
    MeanSquaredError025.append(mse)
```

[22] Python

```
for index, mse in enumerate(MeanSquaredError025):
    print(f'MSE for Sample {index+1} & h value 0.25: {mse}')
```

[23] Python

```
MSE for Sample 1 & h value 0.25: 1.6656802451920276
MSE for Sample 2 & h value 0.25: 1.8825524216281033
MSE for Sample 3 & h value 0.25: 2.272151289530907
MSE for Sample 4 & h value 0.25: 1.5111307247340966
MSE for Sample 5 & h value 0.25: 1.7036898095935766
MSE for Sample 6 & h value 0.25: 2.3383562594900216
MSE for Sample 7 & h value 0.25: 1.7579485051783987
MSE for Sample 8 & h value 0.25: 2.0855804381170584
MSE for Sample 9 & h value 0.25: 1.9446654884319612
MSE for Sample 10 & h value 0.25: 1.9991650767538318
```

Calculating Avarage Mean Squared Error for h value 0.25

```
total = 0
length = len(MeanSquaredError025)

for index in range(10):
    total += MeanSquaredError025[index]

averageMSE_025 = total/length
print('Avarage MSE for h value 0.25:', averageMSE_025)
```

[24] Python

```
Avarage MSE for h value 0.25: 1.9160920258649985
```

Figure 15: MSE and Average MSE for h value 0.25

3.1.4 Kernel Smoothing of h value 0.5

Kernel Smoothing of h value 0.5

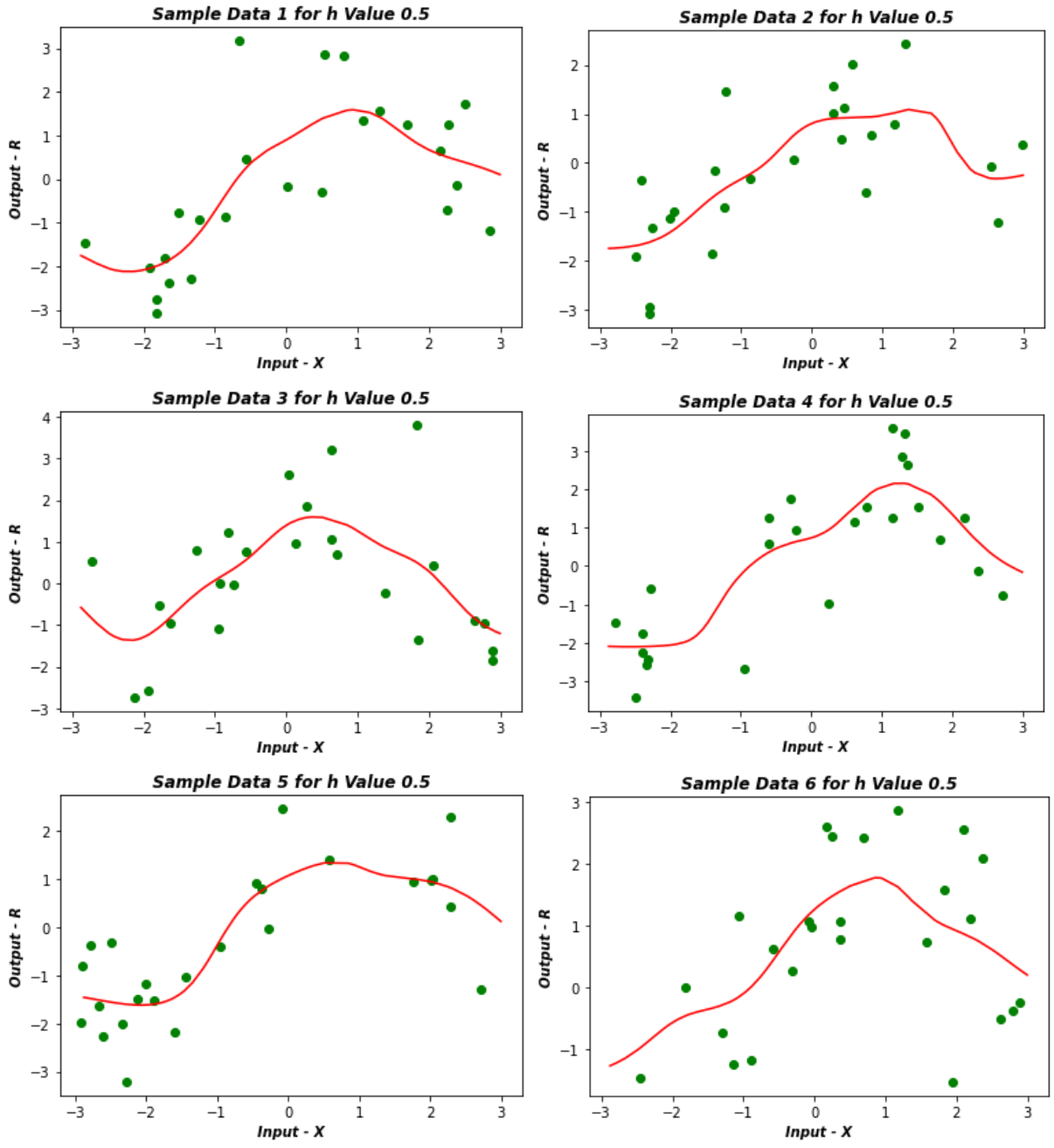
```
kernelSmoother_05 = []

for index in range(10):
    result = kernel_smoother(test_x, train_x[index], train_r[index], 0.5)
    kernelSmoother_05.append(result)
```

[25] Python

```
for index in range(10):
    plt.title(f'Sample Data {index+1} for h Value 0.5', fontweight="bold", style="italic")
    plt.xlabel("Input - X", fontweight="bold", style="italic")
    plt.ylabel("Output - R", fontweight="bold", style="italic")
    plt.scatter(train_x[index], train_r[index], color = 'g')
    plt.plot(test_x, kernelSmoother_05[index], color = 'r')
    plt.show()
```

[26] Python

Figure 16: Kernel Smoothing of h value 0.5



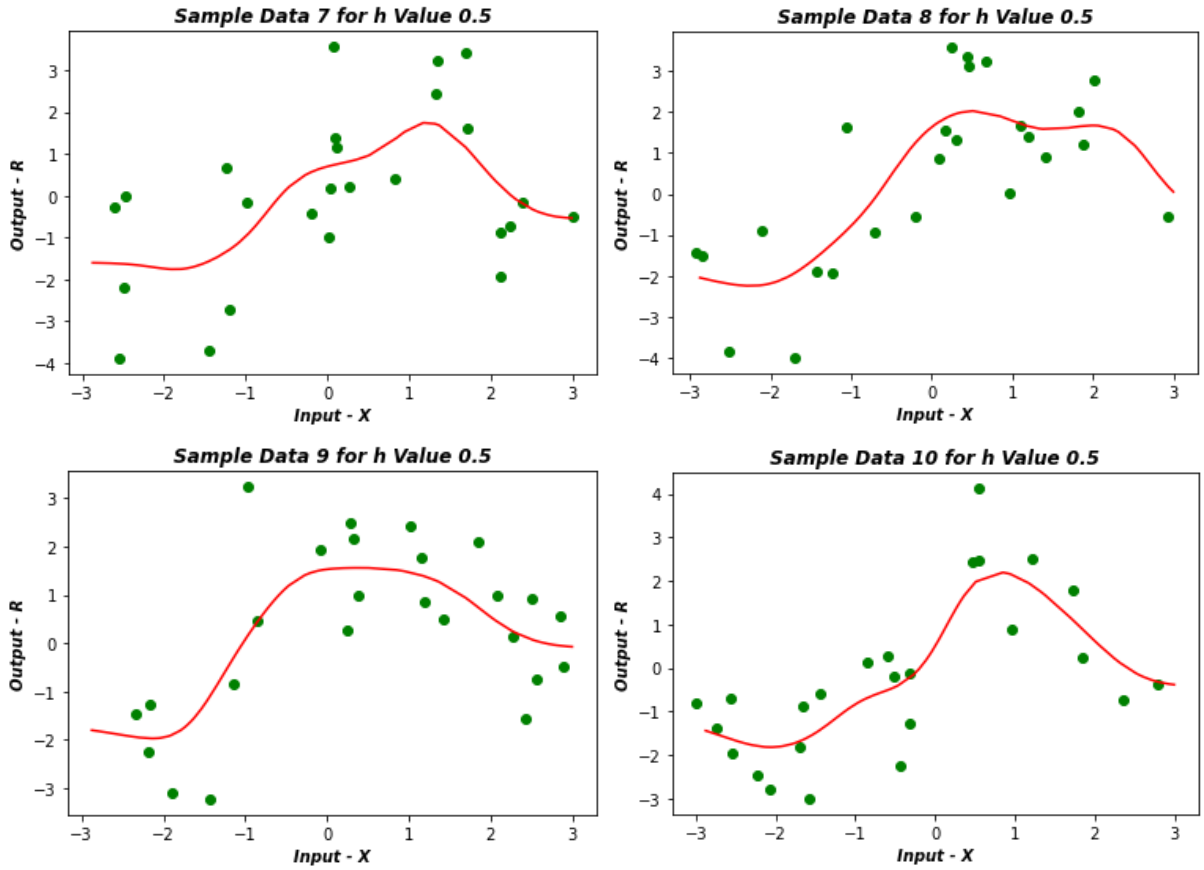


Figure 17: Plotting the result of h value 0.5

Calculating Mean Squared Error for Each Sample

```
MeanSquaredError05 = []

for index in range(10):
    mse = mean_squared_error(test_r, kernelSmoother_05[index])
    MeanSquaredError05.append(mse)
```

[27] Python

```
for index, mse in enumerate(MeanSquaredError05):
    print(f'MSE for Sample {index+1} & h value 0.5: {mse}')
```

[28] Python

```
MSE for Sample 1 & h value 0.5: 1.639330672456036
MSE for Sample 2 & h value 0.5: 1.836829889337494
MSE for Sample 3 & h value 0.5: 2.1095992986169483
MSE for Sample 4 & h value 0.5: 1.5724149497667372
MSE for Sample 5 & h value 0.5: 1.7723281221820535
MSE for Sample 6 & h value 0.5: 2.323273279531599
MSE for Sample 7 & h value 0.5: 1.586051956345641
MSE for Sample 8 & h value 0.5: 1.8661414309609226
MSE for Sample 9 & h value 0.5: 1.8172029390986768
MSE for Sample 10 & h value 0.5: 1.7397594413580786
```

Calculating Avarage Mean Squared Error for h value 0.5

```
total = 0
length = len(MeanSquaredError05)

for index in range(10):
    total += MeanSquaredError05[index]

averageMSE_05 = total/length
print('Avarage MSE for h value 0.5:', averageMSE_05)
```

[29] Python

```
Avarage MSE for h value 0.5: 1.8262931979654187
```

Figure 18: MSE and Average MSE for h value 0.5

3.1.5 Kernel Smoothing of h value 1

Kernel Smoothing of h value 1

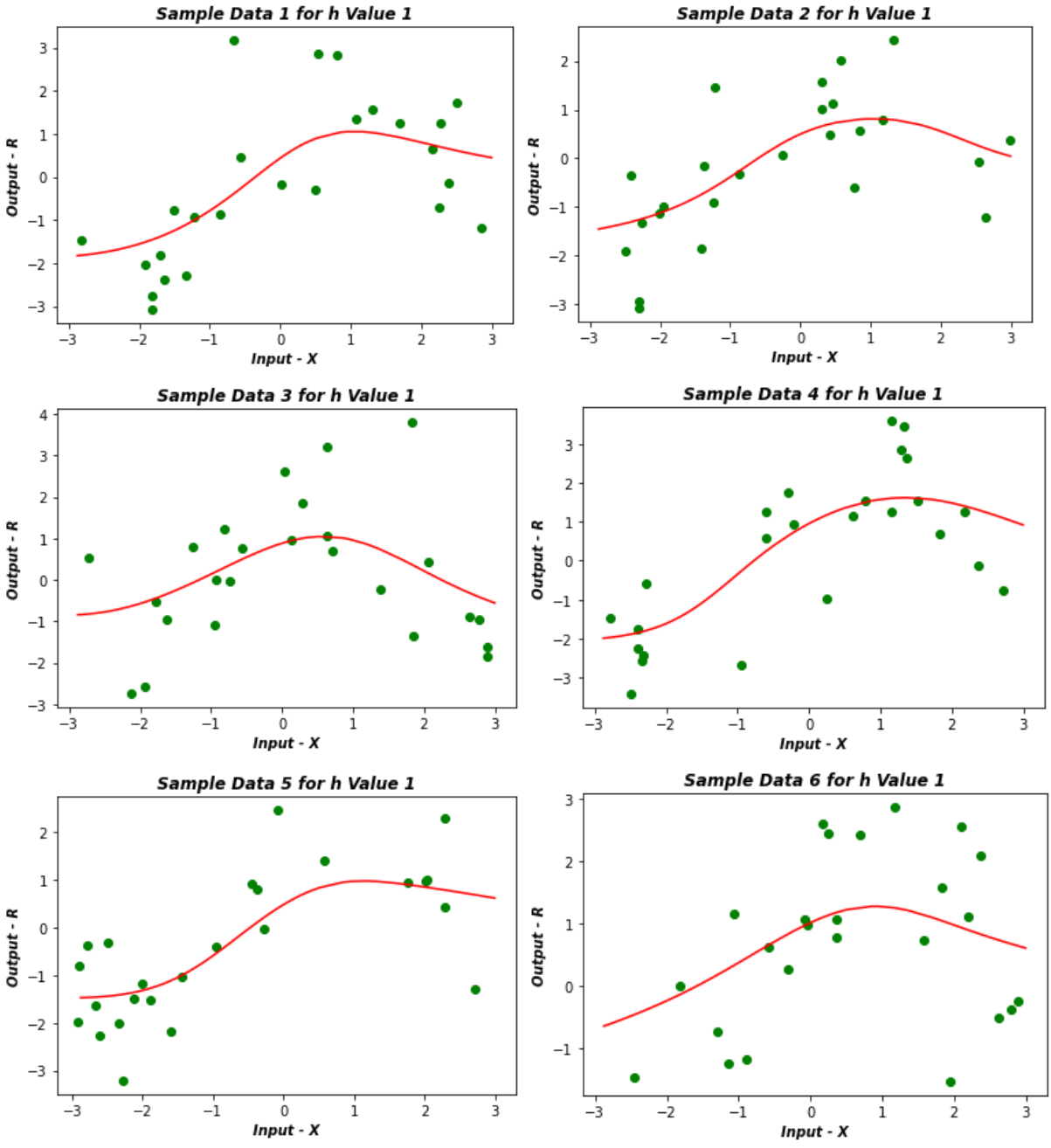
```
kernelSmoother_1 = []

for index in range(10):
    result = kernel_smoother(test_x, train_x[index], train_r[index], 1)
    kernelSmoother_1.append(result)
```

[30] Python

```
for index in range(10):
    plt.title(f'Sample Data {index+1} for h Value 1', fontweight="bold", style="italic")
    plt.xlabel("Input - X", fontweight="bold", style="italic")
    plt.ylabel("Output - R", fontweight="bold", style="italic")
    plt.scatter(train_x[index], train_r[index], color = 'g')
    plt.plot(test_x, kernelSmoother_1[index], color = 'r')
    plt.show()
```

[31] Python

Figure 19: Kernel Smoothing of h value 1



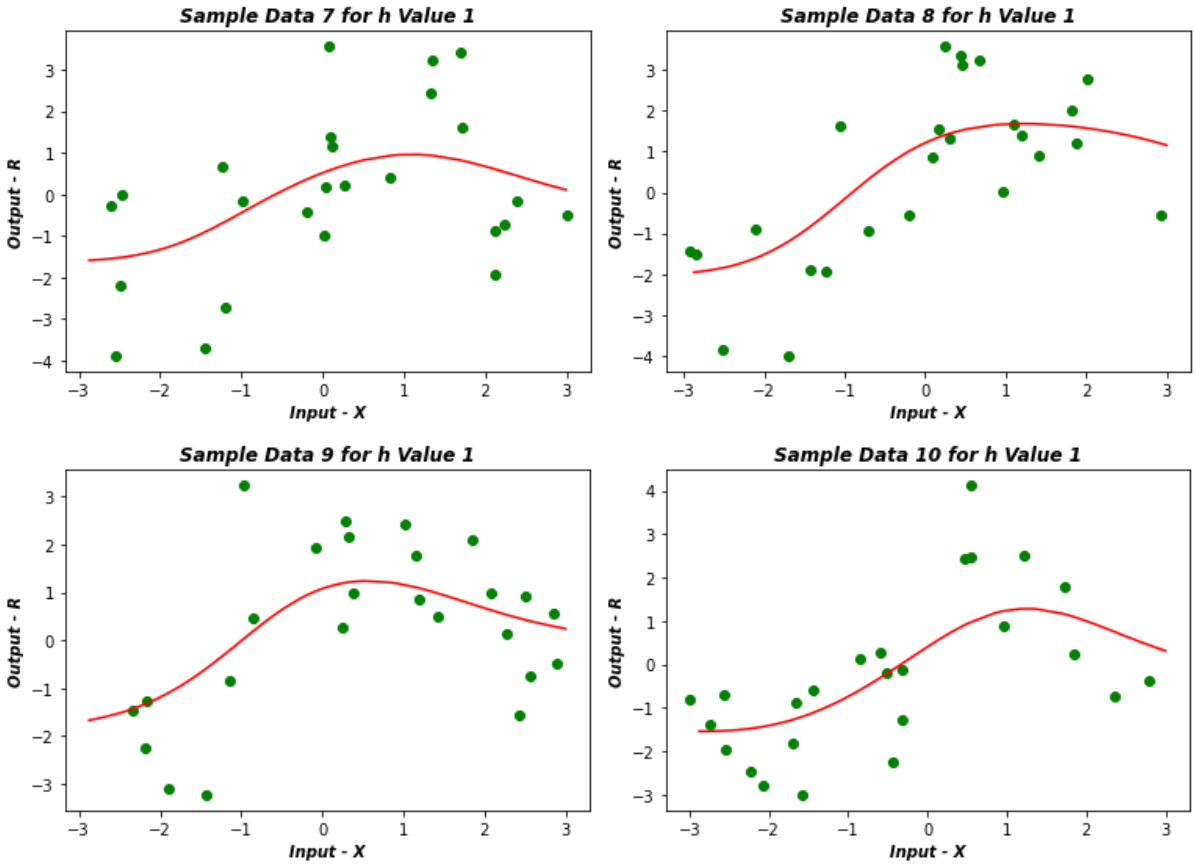


Figure 20: Plotting result of h value 1

Calculating Mean Squared Error for Each Sample

```
MeanSquaredError1 = []

for index in range(10):
    mse = mean_squared_error(test_r, kernelSmoother_1[index])
    MeanSquaredError1.append(mse)
```

[32] Python

```
for index, mse in enumerate(MeanSquaredError1):
    print(f'MSE for Sample {index+1} & h value 1: {mse}')
```

[33] Python

```
MSE for Sample 1 & h value 1: 1.9032892254637963
MSE for Sample 2 & h value 1: 2.0573668105579164
MSE for Sample 3 & h value 1: 2.4803815882053275
MSE for Sample 4 & h value 1: 1.9328095470027522
MSE for Sample 5 & h value 1: 2.013073060115863
MSE for Sample 6 & h value 1: 2.8547680351610736
MSE for Sample 7 & h value 1: 1.9325335171248066
MSE for Sample 8 & h value 1: 2.0903311735280177
MSE for Sample 9 & h value 1: 2.0258605487526973
MSE for Sample 10 & h value 1: 1.8888946670616116
```

Calculating Avarage Mean Squared Error for h value 1

```
total = 0
length = len(MeanSquaredError1)

for index in range(10):
    total += MeanSquaredError1[index]

averageMSE_1 = total/length
print('Avarage MSE for h value 1:', averageMSE_1)
```

[34] Python

```
Avarage MSE for h value 1: 2.1179308172973856
```

Figure 21: MSE and Average MSE for h value 1

3.1.6 Kernel Smoothing of h value 5

Kernel Smoothing of h value 5

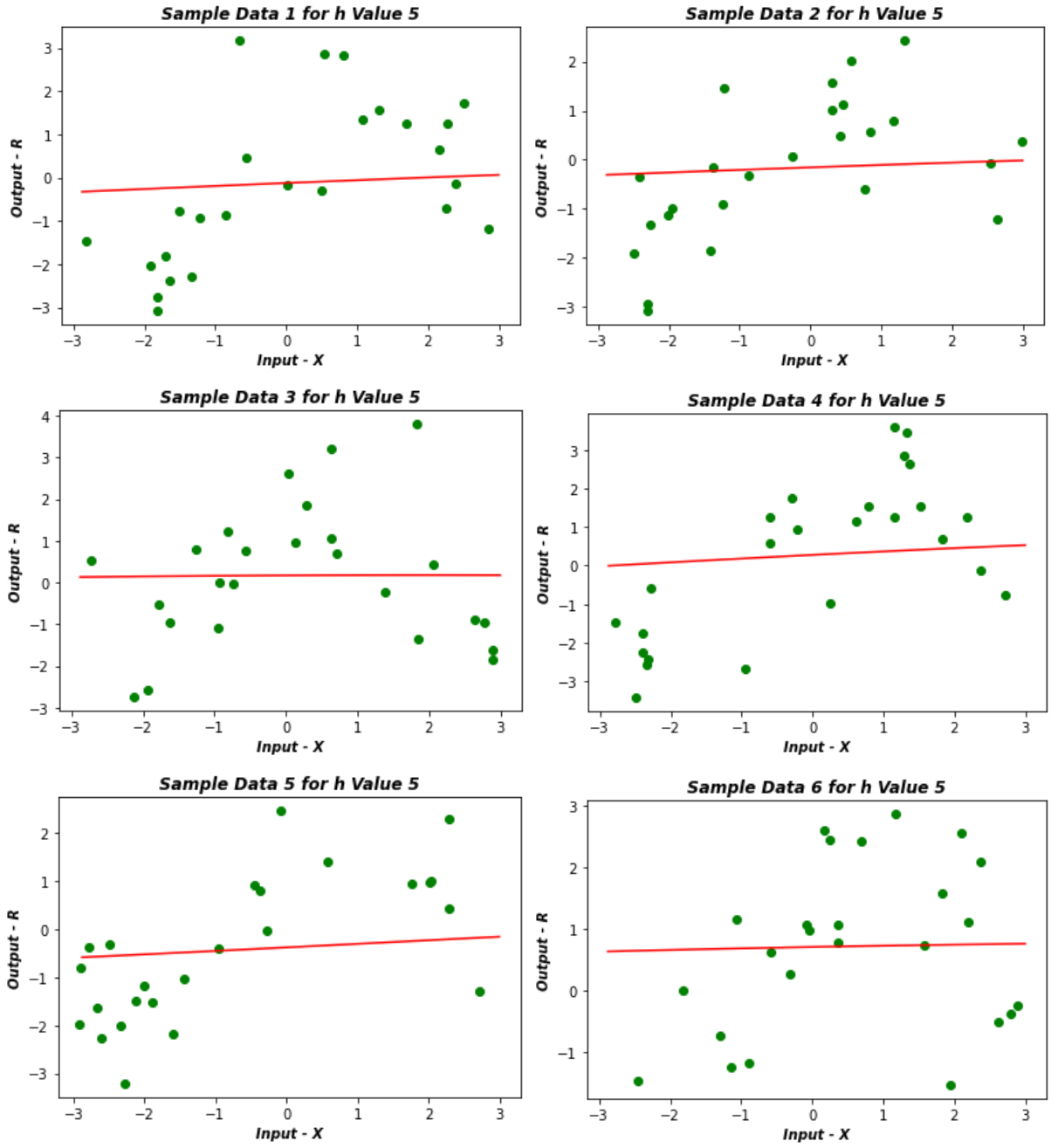
```
kernelSmoother_5 = []

for index in range(10):
    result = kernel_smoother(test_x, train_x[index], train_r[index], 5)
    kernelSmoother_5.append(result)
```

[35] Python

```
for index in range(10):
    plt.title(f'Sample Data {index+1} for h Value 5', fontweight="bold", style="italic")
    plt.xlabel("Input - X", fontweight="bold", style="italic")
    plt.ylabel("Output - R", fontweight="bold", style="italic")
    plt.scatter(train_x[index], train_r[index], color = 'g')
    plt.plot(test_x, kernelSmoother_5[index], color = 'r')
    plt.show()
```

[36] Python

Figure 22: Kernel Smoothing of h value 5



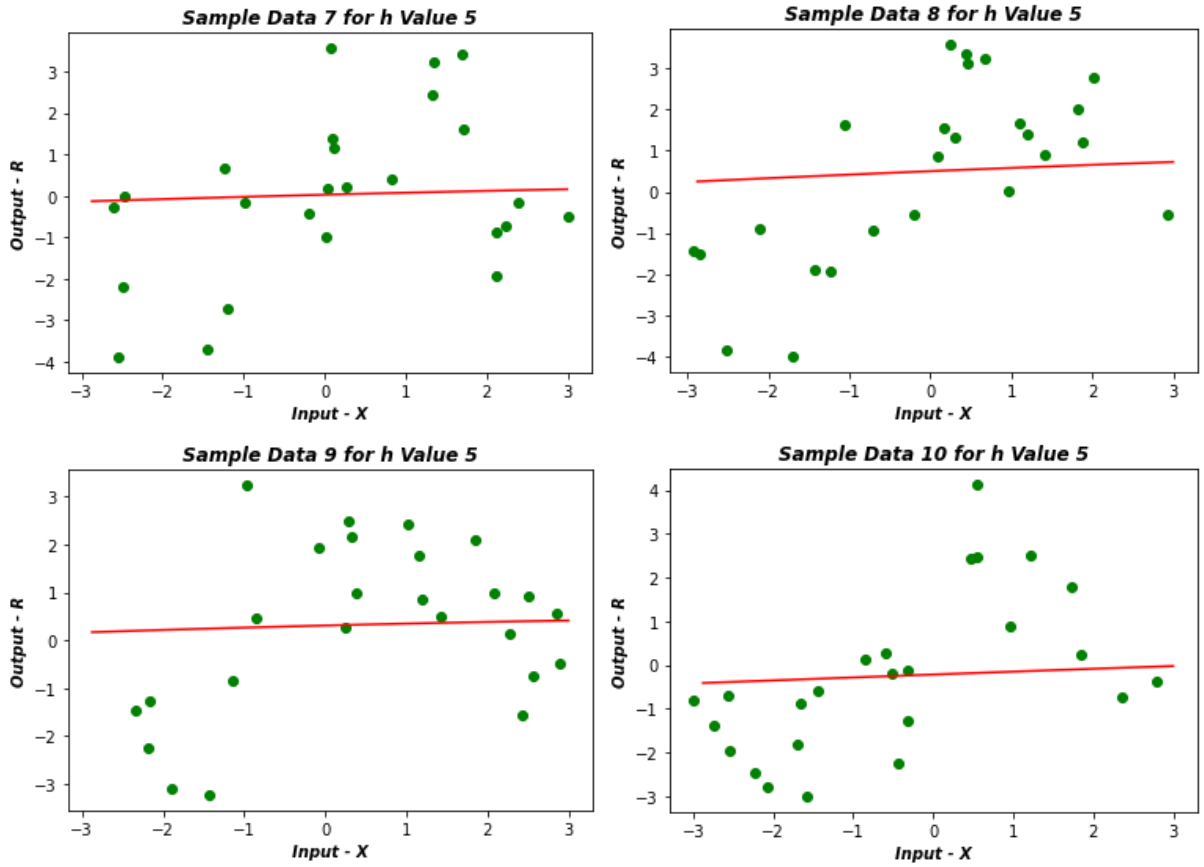


Figure 23: Plotting result of h value 5

Calculating Mean Squared Error for Each Sample

```
MeanSquaredError5 = []

for index in range(10):
    mse = mean_squared_error(test_r, kernelSmoother_5[index])
    MeanSquaredError5.append(mse)
```

[37] Python

```
for index, mse in enumerate(MeanSquaredError5):
    print(f'MSE for Sample {index+1} & h value 5: {mse}')
```

[38] Python

```
MSE for Sample 1 & h value 5: 3.3066496638999805
MSE for Sample 2 & h value 5: 3.3517416866112506
MSE for Sample 3 & h value 5: 3.6145320067427336
MSE for Sample 4 & h value 5: 3.442878837049337
MSE for Sample 5 & h value 5: 3.3222877048539976
MSE for Sample 6 & h value 5: 4.263009270945942
MSE for Sample 7 & h value 5: 3.39901752591922
MSE for Sample 8 & h value 5: 3.7334706524982386
MSE for Sample 9 & h value 5: 3.622125095537891
MSE for Sample 10 & h value 5: 3.3056548582758905
```

Calculating Avarage Mean Squared Error for h value 5

```
total = 0
length = len(MeanSquaredError5)

for index in range(10):
    total += MeanSquaredError5[index]

averageMSE_5 = total/length
print('Avarage MSE for h value 5:', averageMSE_5)
```

[39] Python

```
Avarage MSE for h value 5: 3.5361367302334488
```

Figure 24: MSE and Average MSE for h value 5

4 Results

In this part, I plot the Average Mean Square Error for each h value.

Plotting Average Mean Squared Error for Each h Value

```
averageMSE = [averageMSE_005, averageMSE_01, averageMSE_025, averageMSE_05, averageMSE_1, averageMSE_5]
labels = ['0.05', '0.1', '0.25', '0.5', '1', '5']

for index in range(6):
    plt.title('Average Mean Squared Error for Each h Values', fontweight="bold", style="italic")
    plt.xlabel("h Value", fontweight="bold", style="italic")
    plt.ylabel("Average Mean Squared Error", fontweight="bold", style="italic")
    plt.scatter(f'{labels[index]}', averageMSE[index])
```

[40] Python

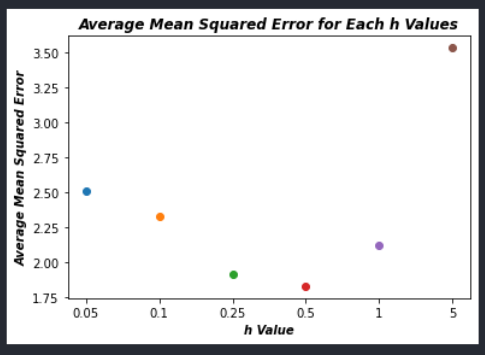


Figure 25: Average Mean Square Error of Models with Different h values

As seen in Figure 25, the lowest error rate was found in the h value 0.5 which is 1.8263 and the highest error rate was found in the h value 5 which is 3.5361. This demonstrates the importance of h value of 0.5 models in the best model selection.

5 Source Code

```
import pandas as pd
import math
from matplotlib import pyplot as plt
from sklearn.metrics import mean_squared_error

trainDataFrame = []
for trainDataIndex in range(1, 11):
    data = pd.read_csv(f'TrainingData/sample{trainDataIndex}.csv', names=['X',
'R'],  header=None)
    trainDataFrame.append(data)
```

```
testDataFrame = pd.read_csv('TestingData/test.csv', names=['X', 'R'],
header=None)

train_x = []
train_r = []
for index in range(10):
    train_x.append(trainDataFrame[index]['X'].values)
    train_r.append(trainDataFrame[index]['R'].values)

test_x = testDataFrame['X'].values
test_r = testDataFrame['R'].values

def plotingData(data_x, data_r, title, color):
    plt.title(title, fontweight="bold", style="italic")
    plt.xlabel("Input - X", fontweight="bold", style="italic")
    plt.ylabel("Output - R", fontweight="bold", style="italic")
    plt.scatter(data_x, data_r, color=color)
    plt.show()

plotingData(train_x, train_r, 'Training Data', 'g')
plotingData(test_x, test_r, 'Testing Data', 'b')

def gaussian_kernel(u):
    return 1/math.sqrt(2*math.pi) * math.exp(-(u**2)/2)

def kernel_smoother(test_x, train_x, train_r, h):
    kernelSmootherResult = []
    for tx in test_x:
        numerator = 0
        denominator = 0
        for index in range(len(train_x)):
            numerator += gaussian_kernel((tx-train_x[index])/h)*train_r[index]
            denominator += gaussian_kernel((tx-train_x[index])/h)
        kernelSmootherResult.append(numerator/denominator)
    return kernelSmootherResult

kernelSmoother_005 = []
for index in range(10):
    result = kernel_smoother(test_x, train_x[index], train_r[index], 0.05)
    kernelSmoother_005.append(result)

for index in range(10):
    plt.title(f'Sample Data {index+1} for h Value 0.05', fontweight="bold",
style="italic")
    plt.xlabel("Input - X", fontweight="bold", style="italic")
    plt.ylabel("Output - R", fontweight="bold", style="italic")
    plt.scatter(train_x[index], train_r[index], color = 'g')
    plt.plot(test_x, kernelSmoother_005[index], color = 'r')
    plt.show()
```

```
MeanSquaredError005 = []
for index in range(10):
    mse = mean_squared_error(test_r, kernelSmoother_005[index])
    MeanSquaredError005.append(mse)

for index, mse in enumerate(MeanSquaredError005):
    print(f'MSE for Sample {index+1} & h value 0.05: {mse}')

total = 0
length = len(MeanSquaredError005)
for index in range(10):
    total += MeanSquaredError005[index]

averageMSE_005 = total/length
print('Avarage MSE for h value 0.05:', averageMSE_005)

kernelSmoother_01 = []
for index in range(10):
    result = kernel_smoother(test_x, train_x[index], train_r[index], 0.1)
    kernelSmoother_01.append(result)

for index in range(10):
    plt.title(f'Sample Data {index+1} for h Value 0.1', fontweight="bold",
style="italic")
    plt.xlabel("Input - X", fontweight="bold", style="italic")
    plt.ylabel("Output - R", fontweight="bold", style="italic")
    plt.scatter(train_x[index], train_r[index], color = 'g')
    plt.plot(test_x, kernelSmoother_01[index], color = 'r')
    plt.show()

MeanSquaredError01 = []
for index in range(10):
    mse = mean_squared_error(test_r, kernelSmoother_01[index])
    MeanSquaredError01.append(mse)

for index, mse in enumerate(MeanSquaredError01):
    print(f'MSE for Sample {index+1} & h value 0.1: {mse}')

total = 0
length = len(MeanSquaredError01)
for index in range(10):
    total += MeanSquaredError01[index]
averageMSE_01 = total/length
print('Avarage MSE for h value 0.1:', averageMSE_01)

kernelSmoother_025 = []
for index in range(10):
    result = kernel_smoother(test_x, train_x[index], train_r[index], 0.25)
    kernelSmoother_025.append(result)
```

```
for index in range(10):
    plt.title(f'Sample Data {index+1} for h Value 0.25', fontweight="bold",
style="italic")
    plt.xlabel("Input - X", fontweight="bold", style="italic")
    plt.ylabel("Output - R", fontweight="bold", style="italic")
    plt.scatter(train_x[index], train_r[index], color = 'g')
    plt.plot(test_x, kernelSmoother_025[index], color = 'r')
    plt.show()

MeanSquaredError025 = []
for index in range(10):
    mse = mean_squared_error(test_r, kernelSmoother_025[index])
    MeanSquaredError025.append(mse)

for index, mse in enumerate(MeanSquaredError025):
    print(f'MSE for Sample {index+1} & h value 0.25: {mse}')

total = 0
length = len(MeanSquaredError025)
for index in range(10):
    total += MeanSquaredError025[index]

averageMSE_025 = total/length
print('Avarage MSE for h value 0.25:', averageMSE_025)

kernelSmoother_05 = []
for index in range(10):
    result = kernel_smoother(test_x, train_x[index], train_r[index], 0.5)
    kernelSmoother_05.append(result)

for index in range(10):
    plt.title(f'Sample Data {index+1} for h Value 0.5', fontweight="bold",
style="italic")
    plt.xlabel("Input - X", fontweight="bold", style="italic")
    plt.ylabel("Output - R", fontweight="bold", style="italic")
    plt.scatter(train_x[index], train_r[index], color = 'g')
    plt.plot(test_x, kernelSmoother_05[index], color = 'r')
    plt.show()

MeanSquaredError05 = []
for index in range(10):
    mse = mean_squared_error(test_r, kernelSmoother_05[index])
    MeanSquaredError05.append(mse)

for index, mse in enumerate(MeanSquaredError05):
    print(f'MSE for Sample {index+1} & h value 0.5: {mse}')

total = 0
length = len(MeanSquaredError05)
```

```
for index in range(10):
    total += MeanSquaredError05[index]

averageMSE_05 = total/length
print('Avarage MSE for h value 0.5:', averageMSE_05)

kernelSmoother_1 = []
for index in range(10):
    result = kernel_smoother(test_x, train_x[index], train_r[index], 1)
    kernelSmoother_1.append(result)

for index in range(10):
    plt.title(f'Sample Data {index+1} for h Value 1', fontweight="bold",
style="italic")
    plt.xlabel("Input - X", fontweight="bold", style="italic")
    plt.ylabel("Output - R", fontweight="bold", style="italic")
    plt.scatter(train_x[index], train_r[index], color = 'g')
    plt.plot(test_x, kernelSmoother_1[index], color = 'r')
    plt.show()

MeanSquaredError1 = []
for index in range(10):
    mse = mean_squared_error(test_r, kernelSmoother_1[index])
    MeanSquaredError1.append(mse)

for index, mse in enumerate(MeanSquaredError1):
    print(f'MSE for Sample {index+1} & h value 1: {mse}')

total = 0
length = len(MeanSquaredError1)
for index in range(10):
    total += MeanSquaredError1[index]

averageMSE_1 = total/length
print('Avarage MSE for h value 1:', averageMSE_1)

kernelSmoother_5 = []
for index in range(10):
    result = kernel_smoother(test_x, train_x[index], train_r[index], 5)
    kernelSmoother_5.append(result)

for index in range(10):
    plt.title(f'Sample Data {index+1} for h Value 5', fontweight="bold",
style="italic")
    plt.xlabel("Input - X", fontweight="bold", style="italic")
    plt.ylabel("Output - R", fontweight="bold", style="italic")
    plt.scatter(train_x[index], train_r[index], color = 'g')
    plt.plot(test_x, kernelSmoother_5[index], color = 'r')
    plt.show()
```

```
MeanSquaredError5 = []
for index in range(10):
    mse = mean_squared_error(test_r, kernelSmoother_5[index])
    MeanSquaredError5.append(mse)

for index, mse in enumerate(MeanSquaredError5):
    print(f'MSE for Sample {index+1} & h value 5: {mse}')

total = 0
length = len(MeanSquaredError5)
for index in range(10):
    total += MeanSquaredError5[index]

averageMSE_5 = total/length
print('Avarage MSE for h value 5:', averageMSE_5)

averageMSE = [averageMSE_005, averageMSE_01, averageMSE_025, averageMSE_05,
averageMSE_1, averageMSE_5]
labels = ['0.05', '0.1', '0.25', '0.5', '1', '5']

for index in range(6):
    plt.title('Average Mean Squared Error for Each h Values',
fontweight="bold", style="italic")
    plt.xlabel("h Value", fontweight="bold", style="italic")
    plt.ylabel("Average Mean Squared Error", fontweight="bold",
style="italic")
    plt.scatter(f'{labels[index]}', averageMSE[index])
```

6 References

[1] *Endogeneity in the nonparametric regression and nonparametric IV. (2020, October 19). ResearchGate. https://www.researchgate.net/figure/Endogeneity-in-the-nonparametric-regression-and-nonparametric-IV_fig2_333175176*