Eren Barış Bostancı
68770

ENGR 421 – HW4

Data Importing:

First, I used np.genfromtxt method from numpy library to import the given csv files into arrays. I remove the first index since it is name of the data. For training set I take the first 150 entry and for testing set I take the last 122 entry.

```
data = np.genfromtxt("hw04_data_set.csv", delimiter=",")
data = data[1:]
x_train = data[0:150, 0]
y_train = data[0:150, 1]

x_test = data[150:, 0]
y_test = data[150:, 1]

K = np.max(y_train)
N = data.shape[0]
```

Set the bin_width, origin and min max values and left, right borders for Regressogram.

```
bin_width = 0.37
origin = 1.5

minimum_value = min(x_train)
maximum_value = max(x_train)

if origin < minimum_value:
    minimum_value = origin

left_borders = np.arange(minimum_value, maximum_value, bin_width)
right_borders = np.arange(minimum_value + bin_width, maximum_value + bin_width, bin_width)
```

Regressogram

For Regressogram I used the formula in Section 8.8 from the textbook.

$$(8.24) \quad \hat{g}(x) = \frac{\sum_{t-1}^{N} b(x, x^t) r^t}{\sum_{t-1}^{N} b(x, x^t)}$$

where

$$b(x, x^t) = \begin{cases} 1 & \text{if } x^t \text{ is the same bin with } x \\ 0 & \text{otherwise} \end{cases}$$

First, I defined the b function to make the calculation easier.

```
def B(x, left_border, right_border):
    tmp = np.zeros(len(x))
    for i in range(len(x)):
        if (left_border < x[i]) & (x[i] <= right_border):
            tmp[i] = 1
        else:
            tmp[i] = 0
    return tmp
```

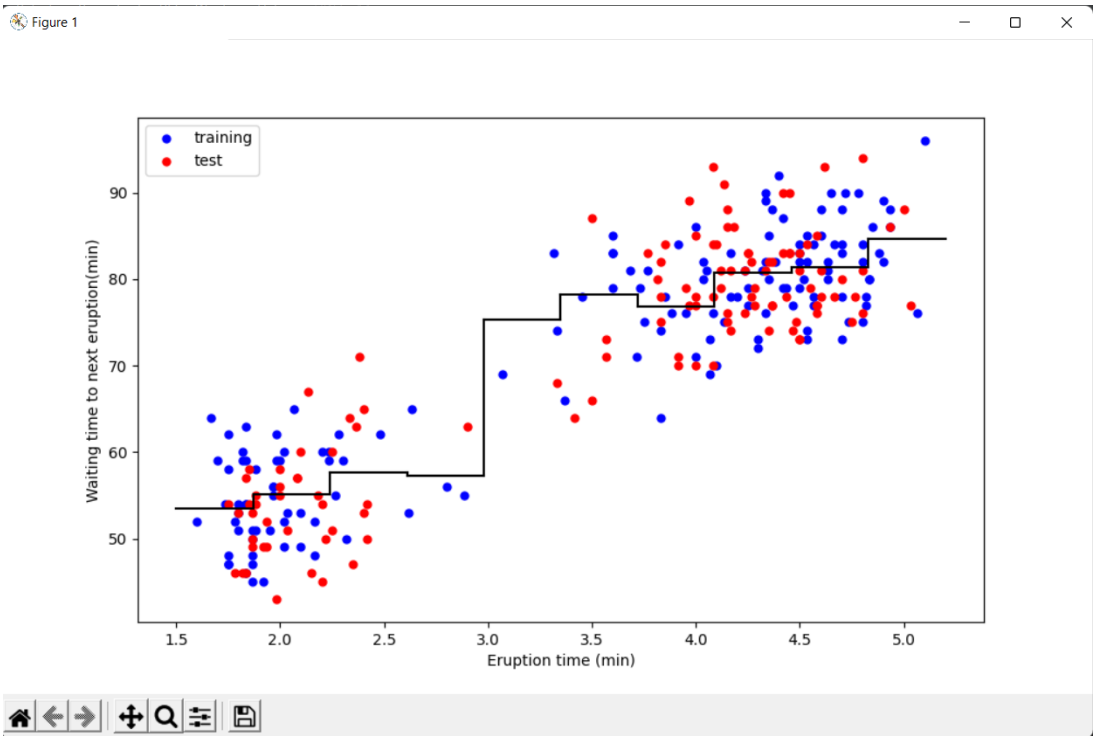
Then I calculate the p_hat and plot the graph.

```
p_hat_regressogram = np.zeros(len(left_borders))
for b in range(len(left_borders)):
    p_hat_regressogram[b] = np.sum(B(x_train, left_borders[b], right_borders[b]) * y_train) / np.sum(B(x_train, left_borders[b], right_borders[b]))

plt.figure(figsize=(10, 6))
plt.plot(x_train, y_train, "b.", markersize=10, label="training")
plt.plot(x_test, y_test, "r.", markersize=10, label="test")

for b in range(len(left_borders)):
    plt.plot([left_borders[b], right_borders[b]], [p_hat_regressogram[b], p_hat_regressogram[b]], "k-")
for b in range(len(left_borders) - 1):
    plt.plot([right_borders[b], right_borders[b]], [p_hat_regressogram[b], p_hat_regressogram[b + 1]], "k-")

plt.xlabel("Eruption time (min)")
plt.ylabel("Waiting time to next eruption(min)")
plt.legend(loc='upper left')
plt.show()
```



RMSE for Regressogram

I defined a function to make calculating easier and used the given formula on HW's manual.

$$\sqrt{\frac{\sum_{i=1}^{N_{test}}(y_i-\hat{y}_i)^2}{N_{test}}}.$$

```
def RMSE(x_test, y_test, p_hat, left_borders, right_borders):
    rmse = 0
    for i in range(len(y_test)):
        test = x_test[i]
        for j in range(len(left_borders)):
            if (left_borders[j] < test) & (test <= right_borders[j]):
                rmse = rmse + (y_test[i] - p_hat[j]) * (y_test[i] - p_hat[j])
    return np.sqrt(rmse / len(y_test))
```

Then calculated the RMSE.

```
rmse_regressogram = RMSE(x_test, y_test, p_hat_regressogram, left_borders, right_borders)

print("Regressogram => RMSE is {:f} when h is {:f}".format(rmse_regressogram, bin_width))
```

```
Regressogram => RMSE is 5.962617 when h is 0.370000
```

Running Mean Smoother

For RMS I defined data_interval and again used the formula, which is given in Section 8.8 from the textbook.

```
bin_width = 0.37

data_interval = np.linspace(minimum_value, maximum_value, 6001)
```

$$(8.25) \quad \hat{g}(x) = \frac{\sum_{t-1}^{N} w\left(\frac{x-x^t}{h}\right) r^t}{\sum_{t-1}^{N} w\left(\frac{x-x^t}{h}\right)}$$

where

$$w(u) = \begin{cases} 1 & \text{if } |u| < 1 \\ 0 & \text{otherwise} \end{cases}$$

I defined W function and calculated the p_hat according to the formula.

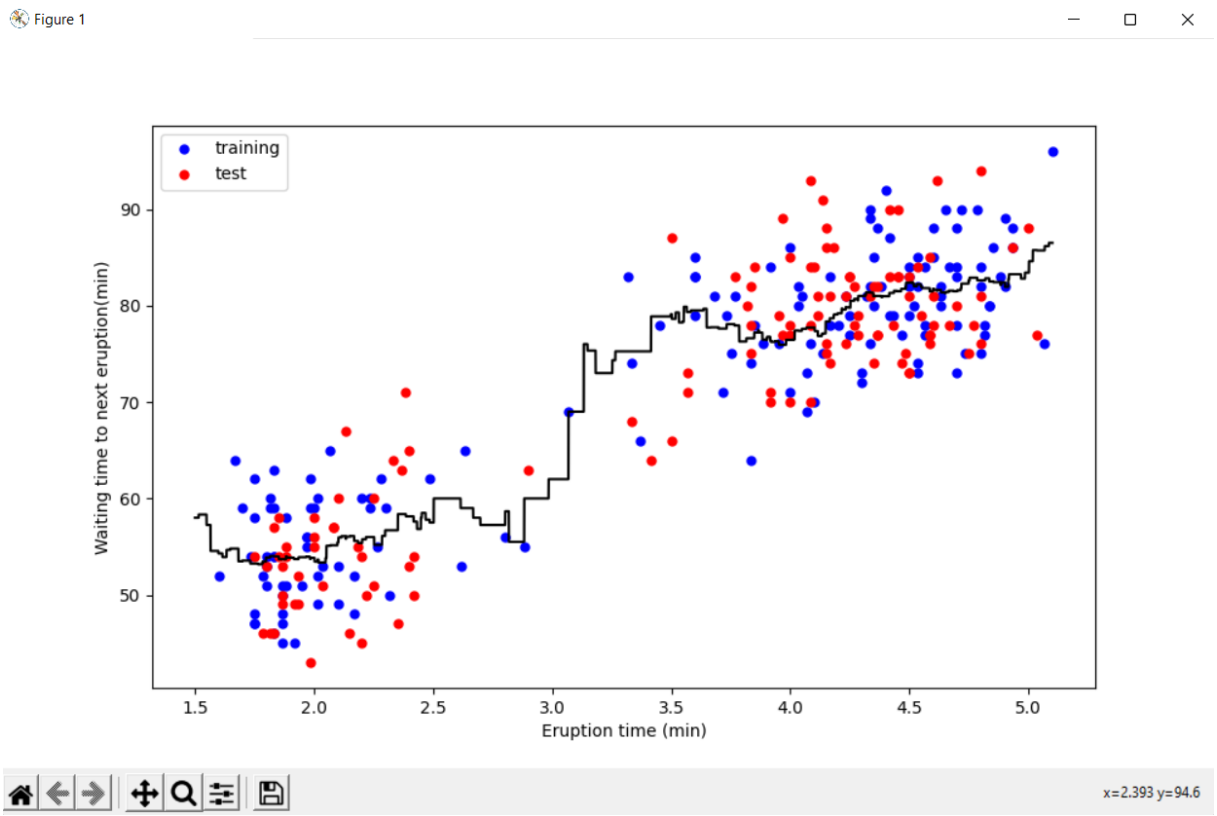
```
def W(x):
    for i in range(len(x)):
        if np.abs(x[i]) < 1:
            x[i] = 1
        else:
            x[i] = 0
    return x
```

```
p_hat_rms = np.zeros(len(data_interval))
for i in range(len(data_interval)):
    p_hat_rms[i] = np.sum(W(((data_interval[i] - x_train) / (bin_width / 2))) * y_train) / np.sum(W(((data_interval[i] - x_train) / (bin_width / 2))))

plt.figure(figsize=(10, 6))
plt.plot(x_train, y_train, "b.", markersize=10, label="training")
plt.plot(x_test, y_test, "r.", markersize=10, label="test")

plt.plot(data_interval, p_hat_rms, "k-")

plt.xlabel("Eruption time (min)")
plt.ylabel("Waiting time to next eruption(min)")
plt.legend(loc='upper left')
plt.show()
```



RMSE for Running Mean Smoother

I defined a new function for Running Mean Smoother and Kernel Smoother since they are based on data_interval instead of left and right borders and used same given formula to calculate the RMSE.

```
def RMSE2(x_test, y_test, p_hat, data_interval):
    rmse = 0
    for i in range(len(p_hat) - 1):
        for j in range(len(x_test)):
            test = x_test[j]
            if (data_interval[i] < test) & (test <= data_interval[i + 1]):
                rmse = rmse + (y_test[j] - p_hat[i]) * (y_test[j] - p_hat[i])

    return math.sqrt(rmse / len(y_test))
```

```
rmse_rms = RMSE2(x_test, y_test, p_hat_rms, data_interval)

print("Running Mean Smoother => RMSE is {:f} when h is {:f}".format(rmse_rms, bin_width))
```

```
D:\Users\erenb\PycharmProjects\ML-HW4\venv\Scripts\python.exe D:/Users/erenb/PycharmProjects/ML-HW4/A4.py
Regressogram => RMSE is 5.962617 when h is 0.370000
Running Mean Smoother => RMSE is 6.089003 when h is 0.370000
```

Kernel Smoother

For Kernel Smoother, I again used data_interval and used the formula, which is given in Section 8.8 from the textbook. And for K function I take it from the LAB6.

$$(8.26) \quad \hat{g}(x) = \frac{\sum_t K\left(\frac{x-x^t}{h}\right) r^t}{\sum_t K\left(\frac{x-x^t}{h}\right)}$$

$$K(u) = \frac{1}{2\pi} \exp\left(-\frac{u^2}{2}\right)$$

I defined the K function and calculated the p_hat and plotted the graph.

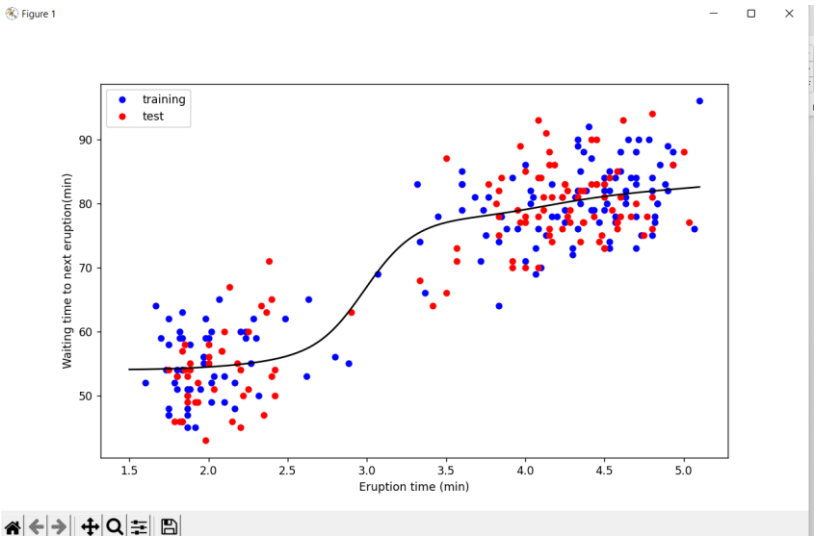
```
def K(x):
    for i in range(len(x)):
        x[i] = np.exp(-0.5 * (x[i] * x[i])) / np.sqrt(2 * math.pi)
    return x
```

```
p_hat_ks = np.zeros(len(data_interval))
for i in range(len(data_interval)):
    p_hat_ks[i] = np.sum(K(((data_interval[i] - x_train) / bin_width)) * y_train) / np.sum(K(((data_interval[i] - x_train) / bin_width)))


plt.figure(figsize=(10, 6))
plt.plot(x_train, y_train, "b.", markersize=10, label="training")
plt.plot(x_test, y_test, "r.", markersize=10, label="test")

plt.plot(data_interval, p_hat_ks, "k-")

plt.xlabel("Eruption time (min)")
plt.ylabel("Waiting time to next eruption(min)")
plt.legend(loc='upper left')
plt.show()
```



Eren Barış Bostancı
68770

RMSE for Kernel Smoother

I called the RMSE2 function, which I defined before to use on Running Mean Smoother part.

```
rmse_ks = RMSE2(x_test, y_test, p_hat_ks, data_interval)

print("Kernel Smoother => RMSE is {:f} when h is {:f}".format(rmse_ks, bin_width))
```

```
D:\Users\erenb\PycharmProjects\ML-HW4\venv\Scripts\python.exe D:/Users/erenb/PycharmProjects/ML-HW4/A4.py
Regressogram => RMSE is 5.962617 when h is 0.370000
Running Mean Smoother => RMSE is 6.089003 when h is 0.370000
Kernel Smoother => RMSE is 5.874299 when h is 0.370000
```