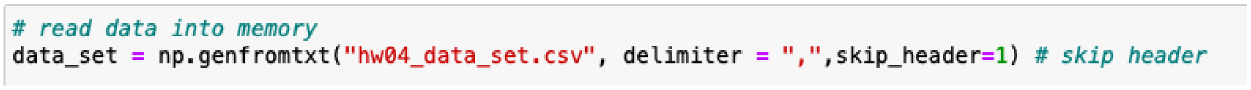
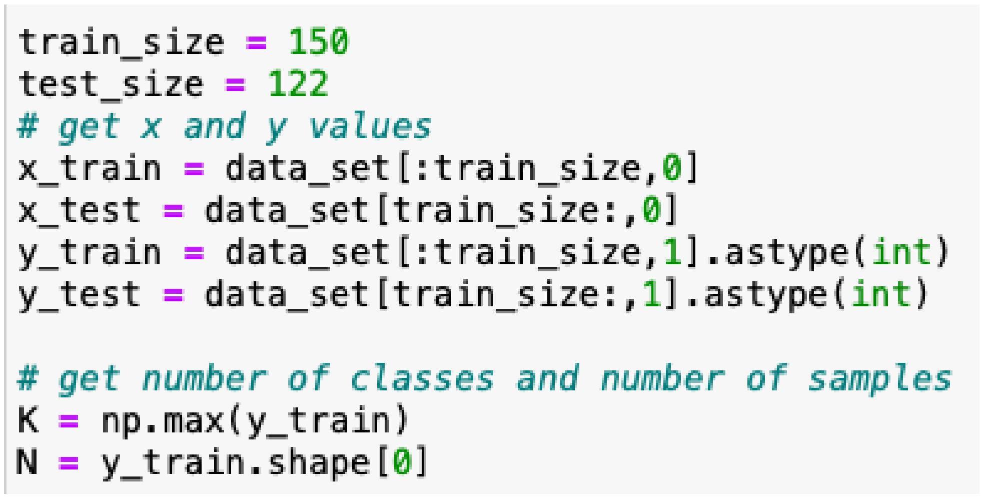
2)

First I read univariate regression data set. Since dataset has header, we will skip it.


```python
# read data into memory
data_set = np.genfromtxt("hw04_data_set.csv", delimiter = ",",skip_header=1) # skip header
```

3)

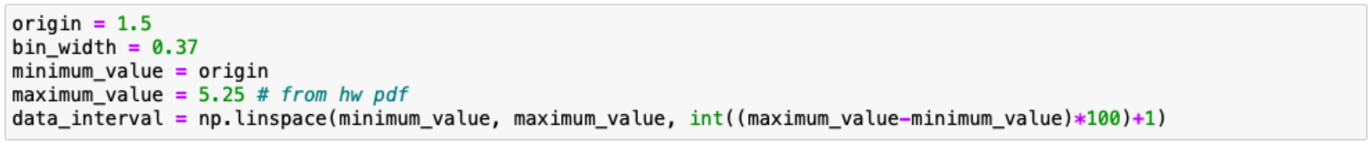
I Divided the data set into two parts by assigning the first 150 data points to the training set and the remaining 122 data points to the test set.


```python
train_size = 150
test_size = 122
# get x and y values
x_train = data_set[:train_size,0]
x_test = data_set[train_size:,0]
y_train = data_set[:train_size,1].astype(int)
y_test = data_set[train_size:,1].astype(int)

# get number of classes and number of samples
K = np.max(y_train)
N = y_train.shape[0]
```

4)

I set the bin width parameter to 0.37 and the origin parameter to 1.5. Since maximum value is not given. I put 5.25 by trying values which fits best for graphs.


```python
origin = 1.5
bin_width = 0.37
minimum_value = origin
maximum_value = 5.25 # from hw pdf
data_interval = np.linspace(minimum_value, maximum_value, int((maximum_value-minimum_value)*100)+1)
```

If we define an origin and a bin width and average the r values in the bin as in the histogram, we get a regressogram. So, to learn regressogram, I used following formulas which is given in the Section 8.8 of the textbook.

Regressogram

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

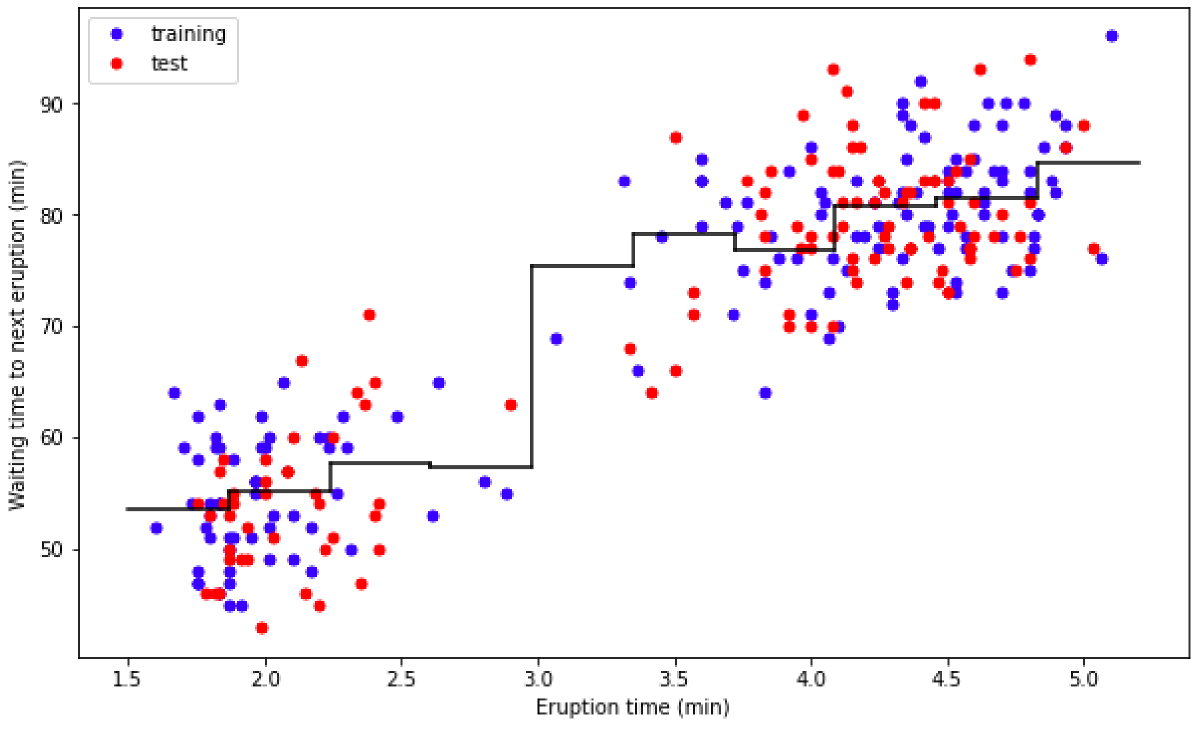
$$b(x, x^t) \begin{cases} 1 \text{ if } x^t \text{ is the same bin with } x \\ 0 \quad \text{otherwise} \end{cases}$$

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

p_hat = np.asarray([np.sum(((left_borders[i] < x_train) & (x_train <= right_borders[i])) * y_train)\
/np.sum((left_borders[i] < x_train) & (x_train <= right_borders[i])) for i in range(len(left_borders))])

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.xlabel("Eruption time (min)")
plt.ylabel("Waiting time to next eruption (min)")
plt.legend(loc = 'upper left')
for b in range(len(left_borders)):
    plt.plot([left_borders[b], right_borders[b]], [p_hat[b], p_hat[b]], "k-")
for b in range(len(left_borders) - 1):
    plt.plot([right_borders[b], right_borders[b]], [p_hat[b], p_hat[b + 1]], "k-")
plt.show()
```

the output is:

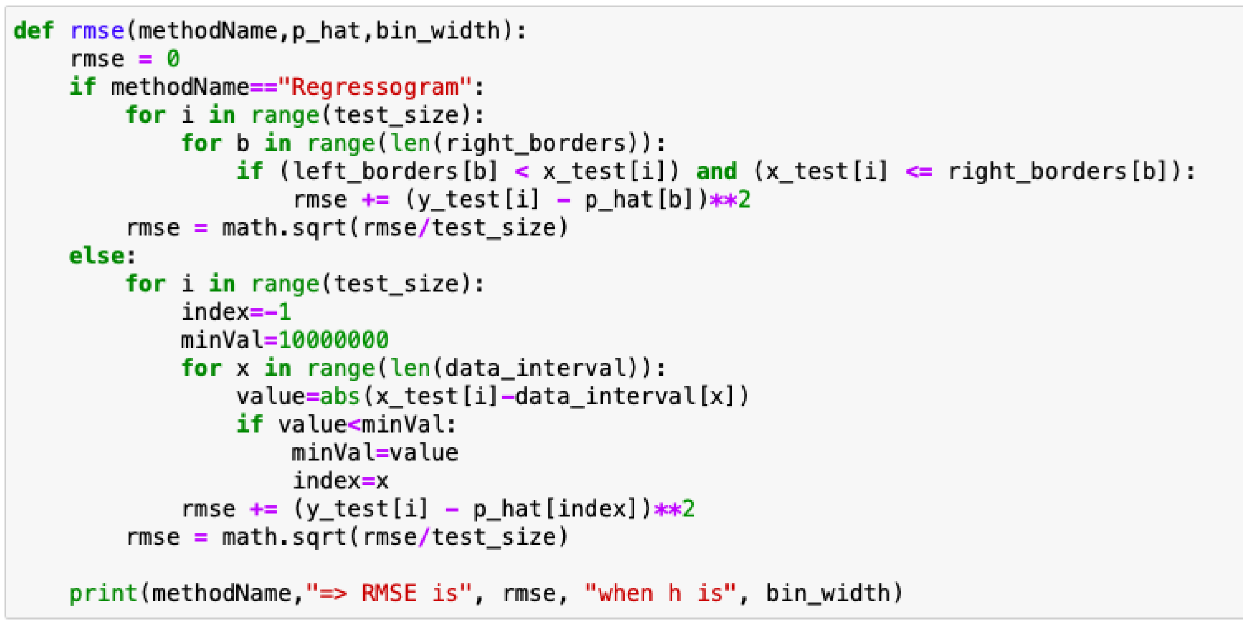


5)

Calculate the root mean squared error (RMSE) of your regressogram for test data points. The formula for RMSE is:

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

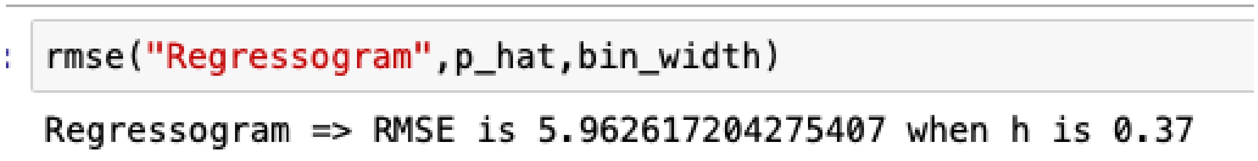
For regressogram we need to check borders to check whether test data is within bin_width. Otherwise numerator of condition is 0. Then we calculate rmse with given formula.



```
def rmse(methodName,p_hat,bin_width):
    rmse = 0
    if methodName=="Regressogram":
        for i in range(test_size):
            for b in range(len(right_borders)):
                if (left_borders[b] < x_test[i]) and (x_test[i] <= right_borders[b]):
                    rmse += (y_test[i] - p_hat[b])**2
        rmse = math.sqrt(rmse/test_size)
    else:
        for i in range(test_size):
            index=-1
            minVal=10000000
            for x in range(len(data_interval)):
                value=abs(x_test[i]-data_interval[x])
                if value<minVal:
                    minVal=value
                    index=x
            rmse += (y_test[i] - p_hat[index])**2
        rmse = math.sqrt(rmse/test_size)

    print(methodName,"=> RMSE is", rmse, "when h is", bin_width)
```

Output is:



```
rmse("Regressogram",p_hat,bin_width)
```

```
Regressogram => RMSE is 5.962617204275407 when h is 0.37
```

6)

Having discontinuities at bin boundaries is disturbing as is the need to fix an origin. As in the naive estimator, in the running mean smoother, we define a bin symmetric around x and average in there (from book).

Using formula from the book is shows us how to Learn a running mean smoother.

Running Mean Smoother

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

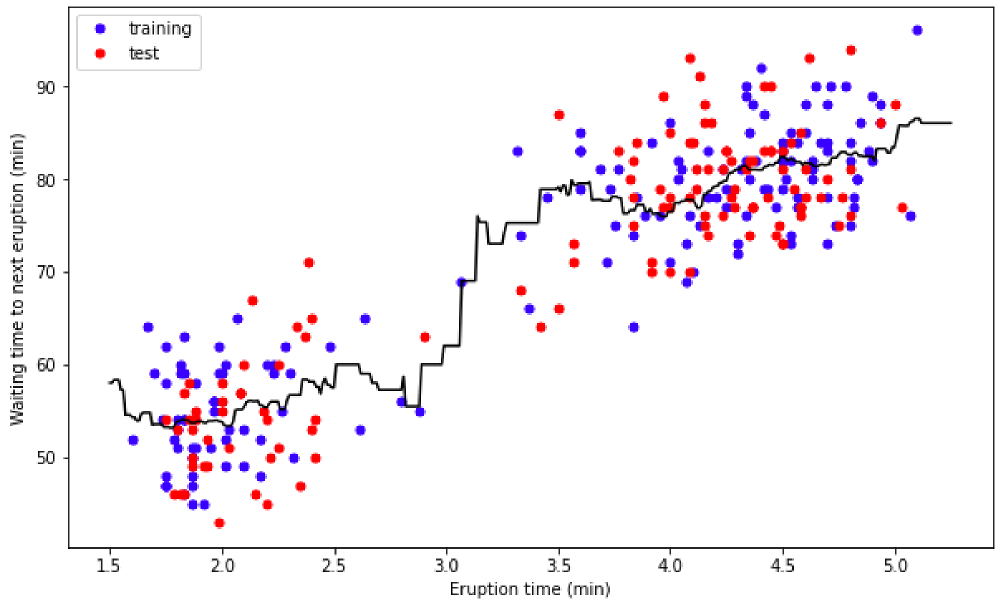
$$w(u) = \begin{cases} 1 \; if \; |u| < \frac{1}{2} \\ 0 \; otherwise \end{cases}$$

implementation of given formula is follows. As in the formula we should check |u|<1/2.

```
#p_hat=np.asarray([np.sum(((((x_train - x)/bin_width) <= 0.5) & (((x_train - x)/bin_width) >= -0.5))*y_train)/
p_hat=np.asarray([np.sum((np.abs((x_train - x)/bin_width) <= 0.5)*y_train)/
                  np.sum(np.abs((x_train - x)/bin_width) <= 0.5)for x in data_interval])


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.xlabel("Eruption time (min)")
plt.ylabel("Waiting time to next eruption (min)")
plt.legend(loc = 'upper left')
plt.plot(data_interval, p_hat, "k-")
plt.show()
```

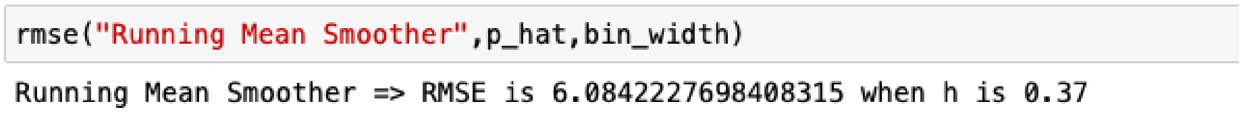
the output is:



7)

rmse is calculated with given formula and I specified implementation in part 5 of this report.

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



```
rmse("Running Mean Smoother",p_hat,bin_width)
```

```
Running Mean Smoother => RMSE is 6.0842227698408315 when h is 0.37
```

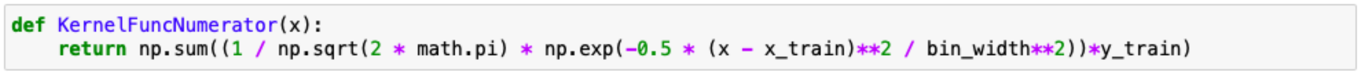
8)

As In the kernel estimator, we can use a kernel giving less weight to further points, and we get the kernel smoother. I implemented given formula from the book.

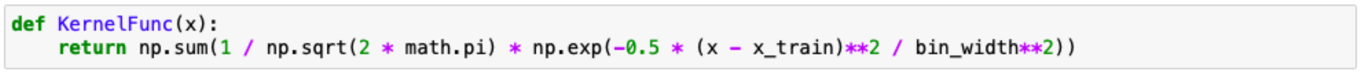
Kernel Smoother

$$\hat{g}(x) = \frac{\sum_{t=1}^{N} K(\frac{x-x^t}{h})r^t}{\sum_{t=1}^{N} K(\frac{x-x^t}{h})}$$

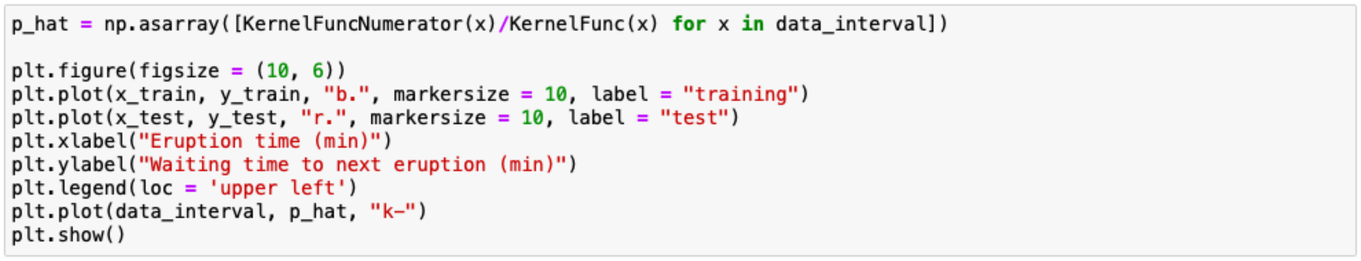
$$K(u) = K(\frac{x-x^t}{h}) = \frac{1}{\sqrt{2\pi}}e^{[\frac{-u^2}{2}]} = \frac{1}{\sqrt{2\pi}}e^{[\frac{-(\frac{x-x_i}{h})^2}{2}]}$$



```
def KernelFuncNumerator(x):
    return np.sum((1 / np.sqrt(2 * math.pi) * np.exp(-0.5 * (x - x_train)**2 / bin_width**2))*y_train)
```



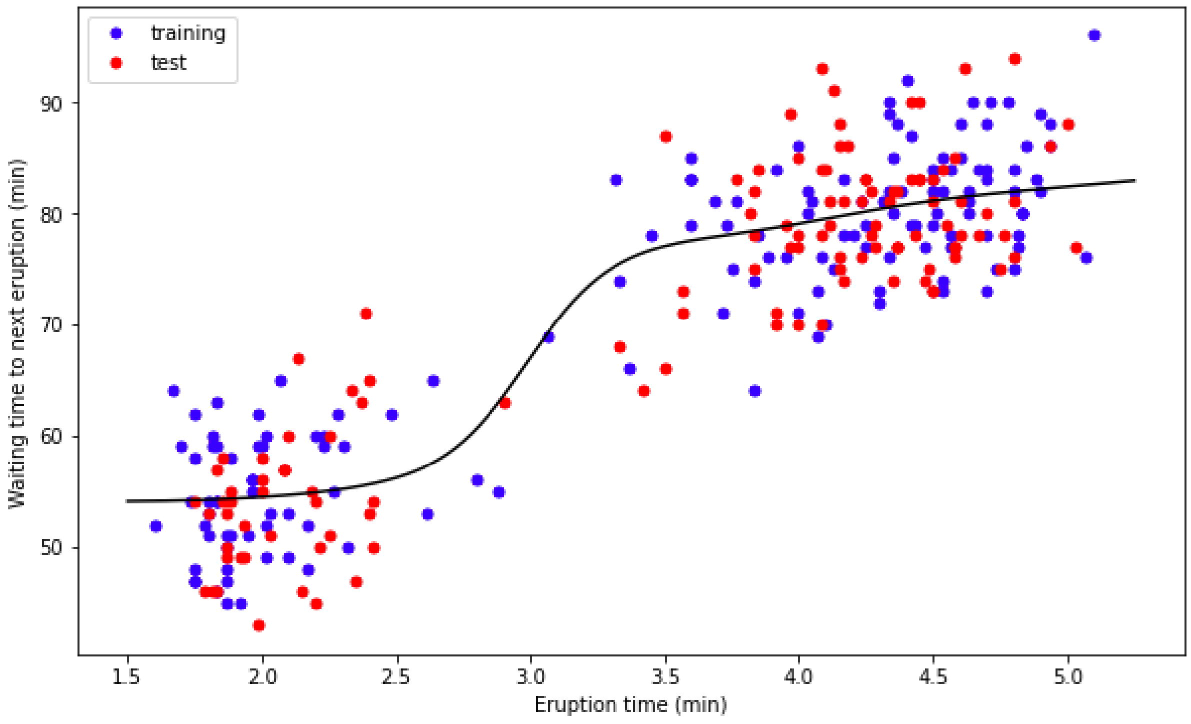
```
def KernelFunc(x):
    return np.sum(1 / np.sqrt(2 * math.pi) * np.exp(-0.5 * (x - x_train)**2 / bin_width**2))
```



```
p_hat = np.asarray([KernelFuncNumerator(x)/KernelFunc(x) for x in data_interval])

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.xlabel("Eruption time (min)")
plt.ylabel("Waiting time to next eruption (min)")
plt.legend(loc = 'upper left')
plt.plot(data_interval, p_hat, "k-")
plt.show()
```

the output is:



9)

RMSE value of Kernel Smoother, is similar to the Running Mean Smoother. And implemented with given formula from the book.

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

```
rmse("Kernel Smoother",p_hat,bin_width)
```

```
Kernel Smoother => RMSE is 5.875422447171615 when h is 0.37
```