Application of Bootstrap samples in Random Forest

In [4]:

```
import numpy as np
from sklearn.datasets import load_boston
from sklearn.metrics import mean_squared_error
```

- Load the boston house dataset

In [5]:

```
boston = load_boston()
x=boston.data #independent variables
y=boston.target #target variable
```

In [6]:

```
x.shape
```

Out[6]:

```
(506, 13)
```

In [7]:

```
y.shape
```

Out[7]:

```
(506,)
```

In [8]:

```
y = y.reshape(506,1)
x=np.append(x, y,axis=1)
x.shape
```

Out[8]:

```
(506, 14)
```

In [9]:

```
y
```

Out[9]:

```
array([[24. ],
       [21.6],
       [34.7],
       [33.4],
       [36.2],
       [28.7],
       [22.9],
       [27.1],
       [16.5],
       [18.9],
       [15. ],
       [18.9],
       [21.7],
       [20.4],
       [18.2],
       [19.9],
       [23.1],
       [17.5],
```

```
[20.2],
[18.2],
[13.6],
[19.6],
[15.2],
[14.5],
[15.6],
[13.9],
[16.6],
[14.8],
[18.4],
[21. ],
[12.7],
[14.5],
[13.2],
[13.1],
[13.5],
[18.9],
[20. ],
[21. ],
[24.7],
[30.8],
[34.9],
[26.6],
[25.3],
[24.7],
[21.2],
[19.3],
[20. ],
[16.6],
[14.4],
[19.4],
[19.7],
[20.5],
[25. ],
[23.4],
[18.9],
[35.4],
[24.7],
[31.6],
[23.3],
[19.6],
[18.7],
[16. ],
[22.2],
[25. ],
[33. ],
[23.5],
[19.4],
[22. ],
[17.4],
[20.9],
[24.2],
[21.7],
[22.8],
[23.4],
[24.1],
[21.4],
[20. ],
[20.8],
[21.2],
[20.3],
[28. ],
[23.9],
[24.8],
[22.9],
[23.9],
[26.6],
[22.5],
[22.2],
[23.6],
[28.7],
[22.6],
[22. ],
[22.9],
[25. ],
[20.6]
```

```
       [20.6],
       [28.4],
       [21.4],
       [38.7],
       [43.8],
       [33.2],
       [27.5],
       [26.5],
       [18.6],
       [19.3],
       [20.1],
       [19.5],
       [19.5],
       [20.4],
       [19.8],
       [19.4],
       [21.7],
       [22.8],
       [18.8],
       [18.7],
       [18.5],
       [18.3],
       [21.2],
       [19.2],
       [20.4],
       [19.3],
       [22. ],
       [20.3],
       [20.5],
       [17.3],
       [18.8],
       [21.4],
       [15.7],
       [16.2],
       [18. ],
       [14.3],
       [19.2],
       [19.6],
       [23. ],
       [18.4],
       [15.6],
       [18.1],
       [17.4],
       [17.1],
       [13.3],
       [17.8],
       [14. ],
       [14.4],
       [13.4],
       [15.6],
       [11.8],
       [13.8],
       [15.6],
       [14.6],
       [17.8],
       [15.4],
       [21.5],
       [19.6],
       [15.3],
       [19.4],
       [17. ],
       [15.6],
       [13.1],
       [41.3],
       [24.3],
       [23.3],
       [27. ],
       [50. ],
       [50. ],
       [50. ],
       [22.7],
       [25. ],
       [50. ],
       [23.8],
       [23.8],
       [22.3],
       [17.4],
       [19.1],
```

```
[19.1],
[23.1],
[23.6],
[22.6],
[29.4],
[23.2],
[24.6],
[29.9],
[37.2],
[39.8],
[36.2],
[37.9],
[32.5],
[26.4],
[29.6],
[50. ],
[32. ],
[29.8],
[34.9],
[37. ],
[30.5],
[36.4],
[31.1],
[29.1],
[50. ],
[33.3],
[30.3],
[34.6],
[34.9],
[32.9],
[24.1],
[42.3],
[48.5],
[50. ],
[22.6],
[24.4],
[22.5],
[24.4],
[20. ],
[21.7],
[19.3],
[22.4],
[28.1],
[23.7],
[25. ],
[23.3],
[28.7],
[21.5],
[23. ],
[26.7],
[21.7],
[27.5],
[30.1],
[44.8],
[50. ],
[37.6],
[31.6],
[46.7],
[31.5],
[24.3],
[31.7],
[41.7],
[48.3],
[29. ],
[24. ],
[25.1],
[31.5],
[23.7],
[23.3],
[22. ],
[20.1],
[22.2],
[23.7],
[17.6],
[18.5],
[24.3],
[20.5],
[24.5],
```

```
[24.5],
[26.2],
[24.4],
[24.8],
[29.6],
[42.8],
[21.9],
[20.9],
[44. ],
[50. ],
[36. ],
[30.1],
[33.8],
[43.1],
[48.8],
[31. ],
[36.5],
[22.8],
[30.7],
[50. ],
[43.5],
[20.7],
[21.1],
[25.2],
[24.4],
[35.2],
[32.4],
[32. ],
[33.2],
[33.1],
[29.1],
[35.1],
[45.4],
[35.4],
[46. ],
[50. ],
[32.2],
[22. ],
[20.1],
[23.2],
[22.3],
[24.8],
[28.5],
[37.3],
[27.9],
[23.9],
[21.7],
[28.6],
[27.1],
[20.3],
[22.5],
[29. ],
[24.8],
[22. ],
[26.4],
[33.1],
[36.1],
[28.4],
[33.4],
[28.2],
[22.8],
[20.3],
[16.1],
[22.1],
[19.4],
[21.6],
[23.8],
[16.2],
[17.8],
[19.8],
[23.1],
[21. ],
[23.8],
[23.1],
[20.4],
[18.5],
[25. ],
```

```
       [24.6],
       [23. ],
       [22.2],
       [19.3],
       [22.6],
       [19.8],
       [17.1],
       [19.4],
       [22.2],
       [20.7],
       [21.1],
       [19.5],
       [18.5],
       [20.6],
       [19. ],
       [18.7],
       [32.7],
       [16.5],
       [23.9],
       [31.2],
       [17.5],
       [17.2],
       [23.1],
       [24.5],
       [26.6],
       [22.9],
       [24.1],
       [18.6],
       [30.1],
       [18.2],
       [20.6],
       [17.8],
       [21.7],
       [22.7],
       [22.6],
       [25. ],
       [19.9],
       [20.8],
       [16.8],
       [21.9],
       [27.5],
       [21.9],
       [23.1],
       [50. ],
       [50. ],
       [50. ],
       [50. ],
       [50. ],
       [13.8],
       [13.8],
       [15. ],
       [13.9],
       [13.3],
       [13.1],
       [10.2],
       [10.4],
       [10.9],
       [11.3],
       [12.3],
       [ 8.8],
       [ 7.2],
       [10.5],
       [ 7.4],
       [10.2],
       [11.5],
       [15.1],
       [23.2],
       [ 9.7],
       [13.8],
       [12.7],
       [13.1],
       [12.5],
       [ 8.5],
       [ 5. ],
       [ 6.3],
       [ 5.6],
       [ 7.2],
```

```
       [12.1],
       [ 8.3],
       [ 8.5],
       [ 5. ],
       [11.9],
       [27.9],
       [17.2],
       [27.5],
       [15. ],
       [17.2],
       [17.9],
       [16.3],
       [ 7. ],
       [ 7.2],
       [ 7.5],
       [10.4],
       [ 8.8],
       [ 8.4],
       [16.7],
       [14.2],
       [20.8],
       [13.4],
       [11.7],
       [ 8.3],
       [10.2],
       [10.9],
       [11. ],
       [ 9.5],
       [14.5],
       [14.1],
       [16.1],
       [14.3],
       [11.7],
       [13.4],
       [ 9.6],
       [ 8.7],
       [ 8.4],
       [12.8],
       [10.5],
       [17.1],
       [18.4],
       [15.4],
       [10.8],
       [11.8],
       [14.9],
       [12.6],
       [14.1],
       [13. ],
       [13.4],
       [15.2],
       [16.1],
       [17.8],
       [14.9],
       [14.1],
       [12.7],
       [13.5],
       [14.9],
       [20. ],
       [16.4],
       [17.7],
       [19.5],
       [20.2],
       [21.4],
       [19.9],
       [19. ],
       [19.1],
       [19.1],
       [20.1],
       [19.9],
       [19.6],
       [23.2],
       [29.8],
       [13.8],
       [13.3],
       [16.7],
       [12. ],
       [14.6],
```

```
       [21.4],
       [23. ],
       [23.7],
       [25. ],
       [21.8],
       [20.6],
       [21.2],
       [19.1],
       [20.6],
       [15.2],
       [ 7. ],
       [ 8.1],
       [13.6],
       [20.1],
       [21.8],
       [24.5],
       [23.1],
       [19.7],
       [18.3],
       [21.2],
       [17.5],
       [16.8],
       [22.4],
       [20.6],
       [23.9],
       [22. ],
       [11.9]])
```

In [10]:

```
x
```

Out[10]:

```
array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 3.9690e+02, 4.9800e+00,
        2.4000e+01],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 3.9690e+02, 9.1400e+00,
        2.1600e+01],
       [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 3.9283e+02, 4.0300e+00,
        3.4700e+01],
       ...,
       [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 3.9690e+02, 5.6400e+00,
        2.3900e+01],
       [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 3.9345e+02, 6.4800e+00,
        2.2000e+01],
       [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 3.9690e+02, 7.8800e+00,
        1.1900e+01]])
```

Task: 1

Step 1 Creating samples: Randomly create 30 samples from the whole boston data points.

1. Creating each sample: Consider any random 303(60% of 506) data points from whole data set and then replicate any 203 points from the sampled points
2. Ex: For better understanding of this procedure lets check this examples, assume we have 10 data points [1,2,3,4,5,6,7,8,9,10], first we take 6 data points randomly consider we have selected [4, 5, 7, 8, 9, 3] now we will replciate 4 points from [4, 5, 7, 8, 9, 3], consder they are [5, 8, 3,7] so our final sample will be [4, 5, 7, 8, 9, 3, 5, 8, 3,7]
3. we create 30 samples like this
4. Note that as a part of the Bagging when you are taking the random samples make sure each of the sample will have different set of columns
5. Ex: assume we have 10 columns for the first sample we will select [3, 4, 5, 9, 1, 2] and for the second sample [7, 9, 1, 4, 5, 6, 2] and so on...
6. Make sure each sample will have atleast 3 feautres/columns/attributes

Step 2 Building High Variance Models on each of the sample and finding train MSE value: Build a DecisionTreeRegressor on each of the sample.

1. Build a regression trees on each of 30 samples.
2. computed the predicted values of each data point(506 data points) in your corpus.
3. predicted house price of i^{th} data point $y^{i}_{pred} = \frac{1}{30}\sum_{k=1}^{30}(\text{predicted value of } x^{i} \text{ with } k^{th} \text{ model})$.

4. Now calculate the $MSE = \frac{1}{506}\sum_{i=1}^{506}(y^{i} - y^{i}_{pred})^{2}$.

Step 3 Calculating the OOB score :

1. Computed the predicted values of each data point(506 data points) in your corpus.
2. Predicted house price of i^{th} data point $y^{i}_{pred} = \frac{1}{k}\sum_{\text{k= model which was buit on samples not included } x^{i}}(\text{predicted value of } x^{i} \text{ with } k^{th} \text{ model})$.
3. Now calculate the $OOB Score = \frac{1}{506}\sum_{i=1}^{506}(y^{i} - y^{i}_{pred})^{2}$.

Task: 2

    Computing CI of OOB Score and Train MSE

    1. Repeat Task 1 for 35 times, and for each iteration store the Train MSE and OOB score

    2. After this we will have 35 Train MSE values and 35 OOB scores

    3. using these 35 values (assume like a sample) find the confidence intravels of MSE and O
       OB Score

    4. you need to report CI of MSE and CI of OOB Score

    5. Note: Refer the Central_Limit_theorem.ipynb to check how to find the confidence intrave
       l

Task: 3

    Given a single query point predict the price of house.

- Consider xq= [0.18,20.0,5.00,0.0,0.421,5.60,72.2,7.95,7.0,30.0,19.1,372.13,18.60] Predict the house price for this point as mentioned in the step 2 of Task 1.

Task 1

In [11]:

```
from sklearn.tree import DecisionTreeRegressor
```

In [12]:

```
def sample(a):
    b = a[np.random.choice(a.shape[0], 303, replace=False), :]
    c = b[np.random.choice(b.shape[0], 203, replace=True), :]
    b = np.concatenate((b, c))
    f = np.random.choice(a.shape[1]-1, np.random.randint(3,a.shape[1]-1,1), replace=False)
    b = b[:,np.append(f,np.array([13]))]
    return b,f
```

In [13]:

```
bags = []
feature = []
for i in range(30):
    p,q = sample(x)
    bags.append(p)
    feature.append(q)
```

In [14]:

```
dt = []
for i in range(30):
```

```
    clf = DecisionTreeRegressor()
    v = bags[i]
    r=len(v[0])
    c=len(v[1])
    clf.fit(v[:,0:(r-1)],v[:,r-1])
    dt.append(clf)
```

In [15]:

```
train = []
for i in range(30):
    v = bags[i]
    r=len(v[0])
    train.append(dt[i].predict(v[:,0:(r-1)]))
```

In [16]:

```
mse = []
for i in range(30):
    v = bags[i]
    r=len(v[0])
    mse.append(mean_squared_error(v[:,r-1],train[i]))
```

In [17]:

```
for i in range(30):
    print('Train MSE of bagged sample {} is {}'.format(i+1,mse[i]))
```

```
Train MSE of bagged sample 1 is 4.302877664841882e-31
Train MSE of bagged sample 2 is 7.717117551075116e-31
Train MSE of bagged sample 3 is 1.8708163760182097e-32
Train MSE of bagged sample 4 is 2.5352476943346502
Train MSE of bagged sample 5 is 1.2534469719322005e-30
Train MSE of bagged sample 6 is 1.0289490068100154e-30
Train MSE of bagged sample 7 is 8.16923150861285e-31
Train MSE of bagged sample 8 is 6.797299499532829e-31
Train MSE of bagged sample 9 is 0.030902503293807646
Train MSE of bagged sample 10 is 2.3385204700227622e-32
Train MSE of bagged sample 11 is 4.489959302443704e-31
Train MSE of bagged sample 12 is 7.483265504072839e-31
Train MSE of bagged sample 13 is 5.986612403258271e-31
Train MSE of bagged sample 14 is 5.425367490452809e-31
Train MSE of bagged sample 15 is 6.236054586727366e-31
Train MSE of bagged sample 16 is 7.857428779276481e-31
Train MSE of bagged sample 17 is 7.857428779276481e-31
Train MSE of bagged sample 18 is 8.730476421418313e-31
Train MSE of bagged sample 19 is 3.1024371568968647e-31
Train MSE of bagged sample 20 is 1.0663653343303796e-30
Train MSE of bagged sample 21 is 6.968791000667832e-31
Train MSE of bagged sample 22 is 9.915326792896513e-31
Train MSE of bagged sample 23 is 0.16527009222661393
Train MSE of bagged sample 24 is 7.670347141674661e-31
Train MSE of bagged sample 25 is 5.097974624649622e-31
Train MSE of bagged sample 26 is 20.114192670009484
Train MSE of bagged sample 27 is 3.1803878392309567e-31
Train MSE of bagged sample 28 is 5.986612403258271e-31
Train MSE of bagged sample 29 is 4.9108929870478004e-31
Train MSE of bagged sample 30 is 2.8062245640273147e-31
```

In [18]:

```
x1 = boston.data
y1 = boston.target
cor = list()
for i in range(30):
    cor.append(dt[i].predict(x1[:,feature[i]]))

yp = np.zeros(x1.shape[0])
for i in range(x1.shape[0]):
    for j in range(30):
        yp[i] += cor[j][i]
    yp[i] /= 30
```

```
total_mse = mean_squared_error(y1,yp)

print(total_mse)
```

2.467458419981667

In [19]:

```
yp_oob = np.zeros(x1.shape[0])
for i in (range(x1.shape[0])):
    cnt1 = 0
    for j in range(30):
        cnt = 0
        for z in range(x1.shape[0]):
            if np.all(np.equal(x1[i,feature[j]],np.array(bags[j][z,0:len(bags[j][0])-1]))):
                cnt += 1
                break
        if cnt != 0:
            continue
        yp_oob[i] += cor[j][i]
        cnt1 += 1
    yp_oob[i] /= cnt1

oob_total = mean_squared_error(y1,yp_oob)
print(oob_total)
```

15.63819642368537

Task 2

In [20]:

```
from tqdm import tqdm
def calculate(a, boston):

    def sample(a):
        b = a[np.random.choice(a.shape[0], 303, replace=False), :]
        c = b[np.random.choice(b.shape[0], 203, replace=True), :]
        b = np.concatenate((b, c))
        f = np.random.choice(a.shape[1]-1, np.random.randint(3,a.shape[1]-1,1), replace=False)
        b = b[:,np.append(f,np.array([13]))]
        return b,f

    mse = []
    mse_oob = []
    for _ in tqdm(range(35)):

        bags = []
        feature = []
        for i in range(30):
            p,q = sample(a)
            bags.append(p)
            feature.append(q)


        dt = []
        for i in range(30):
            clf = DecisionTreeRegressor()
            v = bags[i]
            r=len(v[0])
            c=len(v[1])
            clf.fit(v[:,0:(r-1)],v[:,r-1])
            dt.append(clf)


        x1 = boston.data
        cor = []
        for i in range(30):
            cor.append(dt[i].predict(x1[:,feature[i]]))

        y1 = boston.target
```

```python
        yp = np.zeros(x1.shape[0])
        for i in range(x1.shape[0]):
            for j in range(30):
                yp[i] += cor[j][i]
            yp[i] /= 30

        mse.append(mean_squared_error(y1,yp))

        y_oob = np.zeros(x1.shape[0])
        for i in (range(x1.shape[0])):
            cnt1 = 0
            for j in range(30):
                cnt = 0
                for k in range(x1.shape[0]):
                    if np.all(np.equal(x1[i,feature[j]],np.array(bags[j][k,0:len(bags[j][0])-1]))):
                        cnt += 1
                        break
                if cnt != 0:
                    continue
                y_oob[i] += cor[j][i]
                cnt1 += 1
            y_oob[i] /= cnt1


        mse_oob.append(mean_squared_error(y1,y_oob))

    return mse, mse_oob
```

In [21]:

```python
mse, mse_oob = calculate(x,boston)
```

```
100%|██████████████████████████████████████████████████████████████████████████████████| 35/35
[26:08<00:00, 44.81s/it]
```

In [22]:

```python
mse_ = np.asarray(mse)
mse_oob_ = np.asarray(mse_oob)
```

In [34]:

```python
x = PrettyTable()
x = PrettyTable(["#samples", "Sample Size", "Sample mean", "Pop Mean", "Sample Std ","Left C.I","Ri
ght C.I","Catch"])
for i in range(10):
    sample=mse_[random.sample(range(0, mse_.shape[0]), 30)]
    sample_mean = sample.mean()
    sample_std =  sample.std()
    sample_size = len(sample)
    left_limit  = np.round(sample_mean - 2*(sample_std/np.sqrt(sample_size)), 3)
    right_limit = np.round(sample_mean + 2*(sample_std/np.sqrt(sample_size)), 3)
    row = []
    row.append(i+1)
    row.append(sample_size)
    row.append(sample_mean)
    row.append(np.round(sample_std/np.sqrt(sample_size),3))
    row.append(left_limit)
    row.append(right_limit)
    row.append(total_mse)
    row.append((total_mse <= right_limit) and (total_mse >= left_limit))
    x.add_row(row)
print(x)
```

```
+----------+-------------+--------------------+----------+-------------+----------+----------------
+-------+
| #samples | Sample Size |    Sample mean     | Pop Mean | Sample Std  | Left C.I |     Right C.I
| Catch |
+----------+-------------+--------------------+----------+-------------+----------+----------------
+-------+
|    1     |      30     | 2.502126437638832  |  0.047   |    2.408    |  2.596   | 2.4674584199816
7 |  True |
|    2     |      30     | 2.456004677643511  |  0.049   |    2.358    |  2.554   | 2.4674584199816
```

```
|     2     |      30     | 2.456004677643511  |  0.049   |    2.358    |  2.554   | 2.467458419981
7 |  True |
|     3     |      30     | 2.478398439934136  |  0.051   |    2.377    |  2.579   | 2.467458419981
7 |  True |
|     4     |      30     | 2.470426150904033  |   0.05   |    2.371    |   2.57   | 2.467458419981
7 |  True |
|     5     |      30     | 2.4628454644014806 |  0.047   |    2.368    |  2.557   | 2.467458419981
7 |  True |
|     6     |      30     | 2.488146028803584  |   0.05   |    2.389    |  2.587   | 2.467458419981
7 |  True |
|     7     |      30     | 2.429561716573523  |  0.039   |    2.351    |  2.509   | 2.467458419981
7 |  True |
|     8     |      30     | 2.459836778896213  |  0.044   |    2.371    |  2.548   | 2.467458419981
7 |  True |
|     9     |      30     | 2.4481276143912614 |  0.048   |    2.353    |  2.543   | 2.467458419981
7 |  True |
|     10    |      30     | 2.442112591596823  |  0.048   |    2.346    |  2.538   | 2.467458419981
7 |  True |
+-----------+-------------+--------------------+----------+-------------+----------+---------------
+--------+
```

In [31]:

```python
from prettytable import PrettyTable
import random
x = PrettyTable()
x = PrettyTable(["#samples", "Sample Size", "Sample mean", "Population Mean","Sample Std","Left C.I
","Right C.I", "Catch"])
for i in range(10):
    sample=mse_oob_[random.sample(range(0, mse_oob_.shape[0]), 30)]
    sample_mean = sample.mean()
    sample_std =  sample.std()
    sample_size = len(sample)
    left_limit  = np.round(sample_mean - 2*(sample_std/np.sqrt(sample_size)), 3)
    right_limit = np.round(sample_mean + 2*(sample_std/np.sqrt(sample_size)), 3)
    row = []
    row.append(i+1)
    row.append(sample_size)
    row.append(sample_mean)
    row.append(np.round(sample_std/np.sqrt(sample_size),3))
    row.append(left_limit)
    row.append(right_limit)
    row.append(oob_total)
    row.append((oob_total <= right_limit) and (oob_total >= left_limit))
    x.add_row(row)
print(x)
```

```
+-----------+-------------+--------------------+-----------------+------------+----------+----------
------+-------+
| #samples | Sample Size |     Sample mean     | Population Mean | Sample Std | Left C.I |
Right C.I      | Catch |
+-----------+-------------+--------------------+-----------------+------------+----------+----------
------+-------+
|     1     |      30     | 14.155567545298394 |      0.223      |   13.71    |  14.601  | 15.638196
2368537 | False |
|     2     |      30     | 14.073305489077159 |      0.233      |   13.608   |  14.538  | 15.638196
2368537 | False |
|     3     |      30     | 13.93433876104412  |      0.226      |   13.482   |  14.387  | 15.638196
2368537 | False |
|     4     |      30     | 14.035183513242268 |      0.231      |   13.573   |  14.497  | 15.638196
2368537 | False |
|     5     |      30     | 13.87501388836068  |      0.215      |   13.445   |  14.305  | 15.638196
2368537 | False |
|     6     |      30     | 13.966057514983497 |      0.228      |   13.509   |  14.423  | 15.638196
2368537 | False |
|     7     |      30     |  14.0307046758107  |      0.227      |   13.577   |  14.484  | 15.638196
2368537 | False |
|     8     |      30     | 14.18325340045787  |      0.224      |   13.736   |  14.63   | 15.638196
2368537 | False |
|     9     |      30     | 14.08924263588909  |      0.237      |   13.615   |  14.564  | 15.638196
2368537 | False |
|     10    |      30     | 14.137812266589126 |      0.225      |   13.688   |  14.588  | 15.638196
2368537 | False |
+-----------+-------------+--------------------+-----------------+------------+----------+----------
------+-------+
```

Task 3

In [35]:

```
xq = [0.18,20.0,5.00,0.0,0.421,5.60,72.2,7.95,7.0,30.0,19.1,372.13,18.60]
xq = np.asarray(xq)
cor = list()
for i in range(30):
    s = xq[feature[i]].reshape(1,-1)
    cor.append(dt[i].predict(s))
```

In [36]:

```
yp = 0
for j in range(30):
    yp += cor[j]
yp /= 30

print("Predicted value of xq is {}".format(yp))
```

```
Predicted value of xq is [20.72404762]
```

In [ ]:

In [ ]:

In [ ]: