# Bootstrap assignment

**There will be some functions that start with the word "grader" ex: grader_sampples(), grader_30().. etc, you should not change those function definition.**

**Every Grader function has to return True.</b>**

### Importing packages

In [49]:

```
import numpy as np # importing numpy for numerical computation
from sklearn.datasets import load_boston # here we are using sklearn's boston dataset
from sklearn.metrics import mean_squared_error # importing mean_squared_error metric
```

In [50]:

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

In [51]:

```
print(x[:5])
print(y[:5])
```

```
[[6.3200e-03 1.8000e+01 2.3100e+00 0.0000e+00 5.3800e-01 6.5750e+00
  6.5200e+01 4.0900e+00 1.0000e+00 2.9600e+02 1.5300e+01 3.9690e+02
  4.9800e+00]
 [2.7310e-02 0.0000e+00 7.0700e+00 0.0000e+00 4.6900e-01 6.4210e+00
  7.8900e+01 4.9671e+00 2.0000e+00 2.4200e+02 1.7800e+01 3.9690e+02
  9.1400e+00]
 [2.7290e-02 0.0000e+00 7.0700e+00 0.0000e+00 4.6900e-01 7.1850e+00
  6.1100e+01 4.9671e+00 2.0000e+00 2.4200e+02 1.7800e+01 3.9283e+02
  4.0300e+00]
 [3.2370e-02 0.0000e+00 2.1800e+00 0.0000e+00 4.5800e-01 6.9980e+00
  4.5800e+01 6.0622e+00 3.0000e+00 2.2200e+02 1.8700e+01 3.9463e+02
  2.9400e+00]
 [6.9050e-02 0.0000e+00 2.1800e+00 0.0000e+00 4.5800e-01 7.1470e+00
  5.4200e+01 6.0622e+00 3.0000e+00 2.2200e+02 1.8700e+01 3.9690e+02
  5.3300e+00]]
[24.  21.6 34.7 33.4 36.2]
```

# Task 1

### Step - 1

- **Creating samples**

  **Randomly create 30 samples from the whole boston data points**

  - **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**

    **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 replicate 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]**

- **Create 30 samples**

  - **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**
    **Ex: Assume we have 10 columns[1 ,2 ,3 ,4 ,5 ,6 ,7 ,8 ,9 ,10] 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... 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 regression trees on each of 30 samples.**
- **Computed the predicted values of each data point(506 data points) in your corpus.**
- **Predicted house price of $i^{th}$ data point**
  $y^i_{pred} = \frac{1}{30}\sum_{k=1}^{30}(\text{predicted value of } x^i \text{ with } k^{th} \text{ model})$
- **Now calculate the $MSE = \frac{1}{506}\sum_{i=1}^{506}(y^i - y^i_{pred})^2$**

### Step - 3

- **Calculating the OOB score**

- **Predicted house price of $i^{th}$ data point**
  $y^i_{pred} = \frac{1}{k}\sum_{\text{k= model which was buit on samples not included } x^i}(\text{predicted value of } x^i \text{ with } k^{th} \text{ model})$
  .
- **Now calculate the $OOBScore = \frac{1}{506}\sum_{i=1}^{506}(y^i - y^i_{pred})^2$.**

## Task 2

- **Computing CI of OOB Score and Train MSE**
  - **Repeat Task 1 for 35 times, and for each iteration store the Train MSE and OOB score </li>**
  - **After this we will have 35 Train MSE values and 35 OOB scores**
  - **using these 35 values (assume like a sample) find the confidence intravels of MSE and OOB Score**
  - **you need to report CI of MSE and CI of OOB Score**
  - **Note: Refer the Central_Limit_theorem.ipynb to check how to find the confidence intravel**
    **</ol>**

## Task 3

- **Given a single query point predict the price of house.**

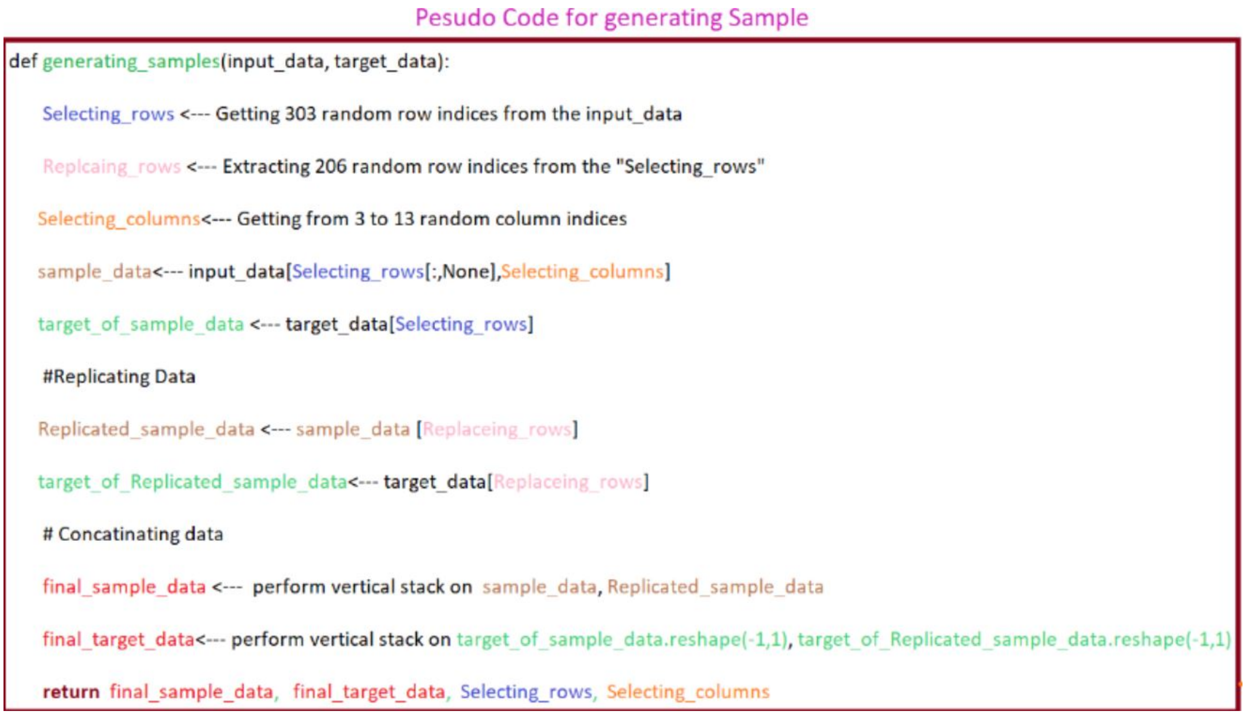
**Consider xq= [0.18,20.0,5.00,0.0,0.421,5.60,72.2,7.95,7.0,30.0,19.1,372.13,18.60] Predict the house price for this point as mentioned in the step 2 of Task 1.**

# Task - 1

### Step - 1

- **Creating samples**

### Algorithm



Pesudo Code for generating Sample

```
def generating_samples(input_data, target_data):

    Selecting_rows <--- Getting 303 random row indices from the input_data

    Replcaing_rows <--- Extracting 206 random row indices from the "Selecting_rows"

   Selecting_columns<--- Getting from 3 to 13 random column indices

   sample_data<--- input_data[Selecting_rows[:,None],Selecting_columns]

   target_of_sample_data <--- target_data[Selecting_rows]

    #Replicating Data

   Replicated_sample_data <--- sample_data [Replaceing_rows]

   target_of_Replicated_sample_data<--- target_data[Replaceing_rows]

    # Concatinating data

    final_sample_data <---  perform vertical stack on  sample_data, Replicated_sample_data

    final_target_data<--- perform vertical stack on target_of_sample_data.reshape(-1,1), target_of_Replicated_sample_data.reshape(-1,1)

    return  final_sample_data,  final_target_data, Selecting_rows, Selecting_columns
```

- **Write code for generating samples**

In [52]:

```
# '''In this function, we will write code for generating 30 samples '''
    # you can use random.choice to generate random indices without replacement
    # Please have a look at this link https://docs.scipy.org/doc/numpy-1.16.0/reference
    # Please follow above pseudo code for generating samples


    # return sampled_input_data , sampled_target_data,selected_rows,selected_columns
    #note please return as lists

def generating_samples(input_data, target_data):

    Selecting_rows = np.random.choice(506 ,303 ,replace=False)
    Selecting_columns = np.random.choice([0,1,2,3,4,5,6,7,8,9,10,11,12], np.random.choi
    sample_data = input_data[ Selecting_rows[:,None] , Selecting_columns ]
    target_of_sample_data = target_data[Selecting_rows]

    #Replicating data
    Replacing_rows = np.random.choice(303, 203)
    Replicated_sample_data = sample_data[Replacing_rows]
    target_of_Replicated_sample_data = target_data[Replacing_rows]
    #Concatenation
    final_sample_data = np.vstack((sample_data,Replicated_sample_data))
    final_target_data = np.vstack((target_of_sample_data.reshape(-1,1),target_of_Replic

    return final_sample_data,final_target_data,Selecting_rows,Selecting_columns
```

### Grader function - 1 </fongt>

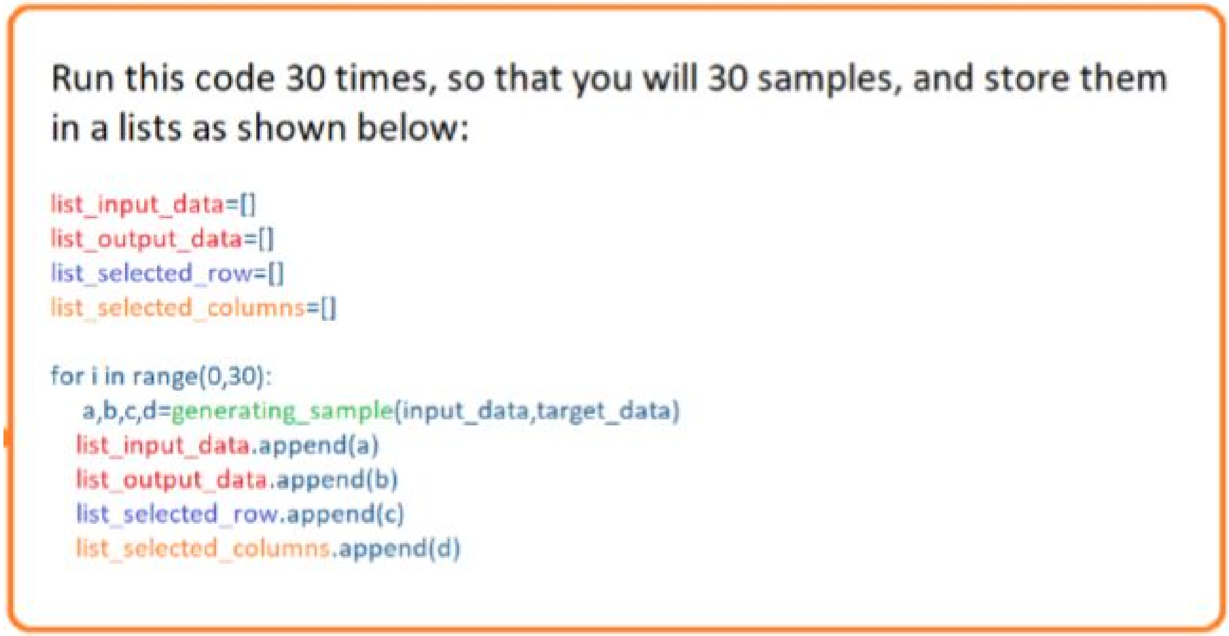
In [53]:

```
def grader_samples(a,b,c,d):
    length = (len(a)==506  and len(b)==506)
    sampled = (len(a)-len(set([str(i) for i in a]))==203)
    rows_length = (len(c)==303)
    column_length= (len(d)>=3)
    assert(length and sampled and rows_length and column_length)
    return True
a,b,c,d = generating_samples(x, y)
grader_samples(a,b,c,d)
```

Out[53]:

```
True
```

- **Create 30 samples**



Run this code 30 times, so that you will 30 samples, and store them in a lists as shown below:

```
list_input_data=[]
list_output_data=[]
list_selected_row=[]
list_selected_columns=[]

for i in range(0,30):
   a,b,c,d=generating_sample(input_data,target_data)
  list_input_data.append(a)
  list_output_data.append(b)
  list_selected_row.append(c)
  list_selected_columns.append(d)
```

In [54]:

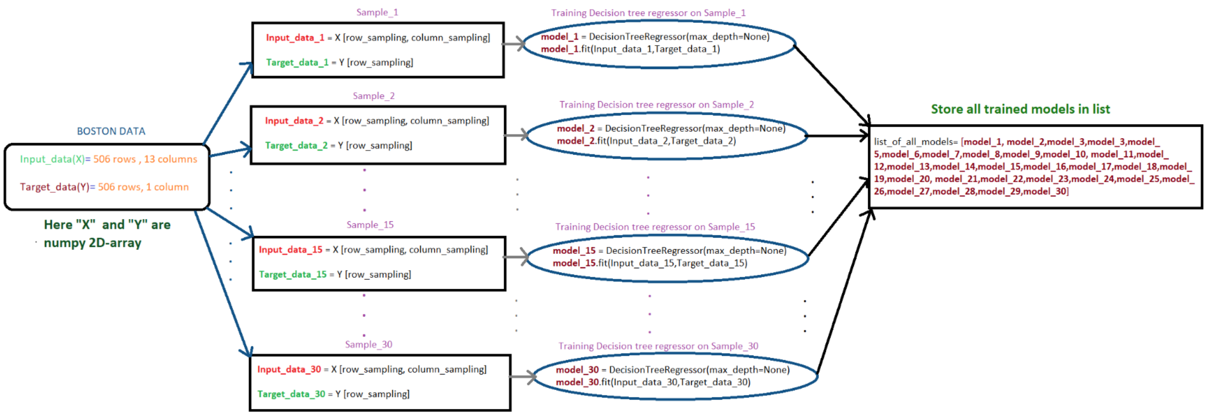
```
# Use generating_samples function to create 30 samples
# store these created samples in a list
list_input_data =[]
list_output_data =[]
list_selected_row= []
list_selected_columns=[]

for i in range(0,30):
    a,b,c,d = generating_samples(x,y)
    list_input_data.append(a)
    list_output_data.append(b)
    list_selected_row.append(c)
    list_selected_columns.append(d)
```

### Grader function - 2

```
In [55]: def grader_30(a):
             assert(len(a)==30 and len(a[0])==506)
             return True
         grader_30(list_input_data)

Out[55]: True
```

## Step - 2

### Flowchart for building tree

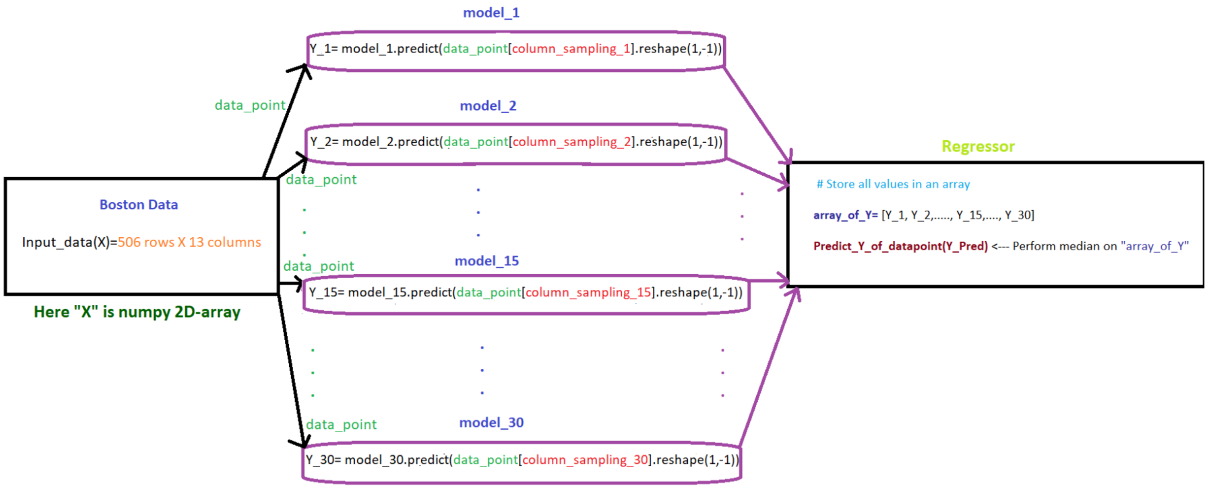

BOSTON DATA
Input_data(X)= 506 rows , 13 columns
Target_data(Y)= 506 rows, 1 column
Here "X" and "Y" are numpy 2D-array
Sample_1
Input_data_1 = X [row_sampling, column_sampling]
Target_data_1 = Y [row_sampling]
Sample_2
Input_data_2 = X [row_sampling, column_sampling]
Target_data_2 = Y [row_sampling]
Sample_15
Input_data_15 = X [row_sampling, column_sampling]
Target_data_15 = Y [row_sampling]
Sample_30
Input_data_30 = X [row_sampling, column_sampling]
Target_data_30 = Y [row_sampling]
Training Decision tree regressor on Sample_1
model_1 = DecisionTreeRegressor(max_depth=None)
model_1.fit(Input_data_1,Target_data_1)
Training Decision tree regressor on Sample_2
model_2 = DecisionTreeRegressor(max_depth=None)
model_2.fit(Input_data_2,Target_data_2)
Training Decision tree regressor on Sample_15
model_15 = DecisionTreeRegressor(max_depth=None)
model_15.fit(Input_data_15,Target_data_15)
Training Decision tree regressor on Sample_30
model_30 = DecisionTreeRegressor(max_depth=None)
model_30.fit(Input_data_30,Target_data_30)
Store all trained models in list
list_of_all_models= [model_1, model_2,model_3,model_3,model_5,model_6,model_7,model_8,model_9,model_10, model_11,model_12,model_13,model_14,model_15,model_16,model_17,model_18,model_19,model_20, model_21,model_22,model_23,model_24,model_25,model_26,model_27,model_28,model_29,model_30]

- **Write code for building regression trees**

```
In [56]: from sklearn.tree import DecisionTreeRegressor

         list_of_all_models = []

         for i in range(len(list_input_data)):
             dt = DecisionTreeRegressor(max_depth = None)
             trained_dt = dt.fit(list_input_data[i],list_output_data[i])
             list_of_all_models.append(trained_dt)

         print(list_of_all_models)    #storing all decision trees for each set of input and outp
```

```
[DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTree
Regressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeRegressor(),
DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeR
egressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), D
ecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeRe
gressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), De
cisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeReg
ressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), Dec
isionTreeRegressor(), DecisionTreeRegressor()]
```

### Flowchart for calculating MSE


model_1
Y_1= model_1.predict(data_point[column_sampling_1].reshape(1,-1))
data_point
model_2
Y_2= model_2.predict(data_point[column_sampling_2].reshape(1,-1))
Regressor
data_point
Boston Data
Input_data(X)=506 rows X 13 columns
# Store all values in an array
array_of_Y= [Y_1, Y_2,....., Y_15,...., Y_30]
Predict_Y_of_datapoint(Y_Pred) <--- Perform median on "array_of_Y"
data_point
model_15
Y_15= model_15.predict(data_point[column_sampling_15].reshape(1,-1))
Here "X" is numpy 2D-array
data_point
model_30
Y_30= model_30.predict(data_point[column_sampling_30].reshape(1,-1))

---

**After getting predicted_y for each data point, we can use sklearns mean_squared_error to calculate the MSE between predicted_y and actual_y.**

- **Write code for calculating MSE**

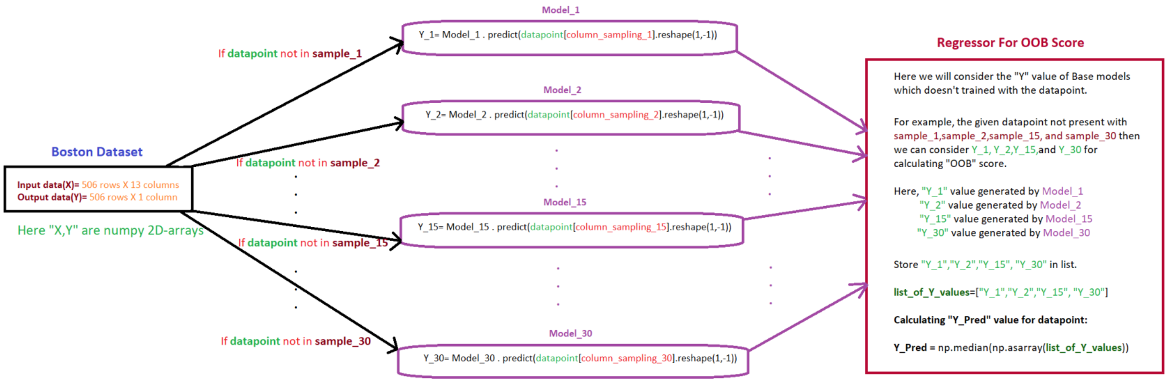
In [57]:

```
from sklearn.metrics import mean_squared_error

array_of_Y = []

for k in range(len(list_of_all_models)):

    y_predicted = list_of_all_models[k].predict(list_input_data[k])

    array_of_Y.append(y_predicted)

array_of_Y = np.array(array_of_Y)   #array of predicted Y all data points for each samp

predicted_y = []

for j in range(array_of_Y.shape[1]):

    sorted_array_of_Y = np.sort(array_of_Y[:,j])

    median = ( (sorted_array_of_Y[14] + sorted_array_of_Y[15]) / 2)

    predicted_y.append(median)   #median of prediction of each data point from all the


MSE = mean_squared_error(y , predicted_y )

print(MSE)
```

```
63.69688280789163
```

## Step - 3

## Flowchart for calculating OOB score


Boston Dataset
Input data(X)= 506 rows X 13 columns
Output data(Y)= 506 rows X 1 column
Here "X,Y" are numpy 2D-arrays
If datapoint not in sample_1
If datapoint not in sample_2
If datapoint not in sample_15
If datapoint not in sample_30
Model_1
Y_1= Model_1 . predict(datapoint[column_sampling_1].reshape(1,-1))
Model_2
Y_2= Model_2 . predict(datapoint[column_sampling_2].reshape(1,-1))
Model_15
Y_15= Model_15 . predict(datapoint[column_sampling_15].reshape(1,-1))
Model_30
Y_30= Model_30 . predict(datapoint[column_sampling_30].reshape(1,-1))
Regressor For OOB Score

Here we will consider the "Y" value of Base models which doesn't trained with the datapoint.

For example, the given datapoint not present with sample_1,sample_2,sample_15, and sample_30 then we can consider Y_1, Y_2,Y_15,and Y_30 for calculating "OOB" score.

Here, "Y_1" value generated by Model_1
"Y_2" value generated by Model_2
"Y_15" value generated by Model_15
"Y_30" value generated by Model_30

Store "Y_1","Y_2","Y_15", "Y_30" in list.

list_of_Y_values=["Y_1","Y_2","Y_15", "Y_30"]

**Calculating "Y_Pred" value for datapoint:**

Y_Pred = np.median(np.asarray(list_of_Y_values))

**Now calculate the** $OOBScore = \frac{1}{506}\sum_{i=1}^{506}(y^i - y^i_{pred})^2$.

- **Write code for calculating OOB score**

In [58]:
```
final_model_list = []    #for ith data point not in which samples/models

for i in range(len(x)):
    temp = []
    for j in range(len(list_selected_row)):
        if (i not in list_selected_row[j]):  #for each data point , storing the models
            temp.append(j)
    final_model_list.append(temp)

print(final_model_list[:5])   #length of 506
```

```
[[4, 11, 16, 21, 23, 24, 25, 26, 28], [4, 6, 7, 9, 10, 11, 13, 19, 20, 22, 25, 27], [0,
2, 4, 5, 13, 14, 15, 17, 19, 21, 24, 26, 27], [1, 7, 10, 11, 13, 14, 18, 19, 21, 22, 2
7], [2, 3, 4, 5, 8, 10, 11, 15, 17, 18, 20, 22, 23, 24, 26, 29]]
```

In [59]:
```
#array_of_Y

predicted_y = []

for j in range(array_of_Y.shape[1]):


    sorted_array_of_Y = np.sort(array_of_Y[final_model_list[j],j])  #from array of all

    median = np.median(sorted_array_of_Y)

    predicted_y.append(median)


OOB = mean_squared_error(y , predicted_y )

print(OOB)
```

```
71.79590533935607
```

# Task 2

In [60]:

```
MSE_35 = []      #runinng above code for 35 times and storing MSE of each run
OOB_35 = []      #runinng above code for 35 times and storing OOB of each run

for z in range(35):

    def generating_samples(input_data, target_data):

        Selecting_rows = np.random.choice(506 ,303 ,replace=False)
        Selecting_columns = np.random.choice([0,1,2,3,4,5,6,7,8,9,10,11,12], np.random.
        sample_data = input_data[ Selecting_rows[:,None] , Selecting_columns ]
        target_of_sample_data = target_data[Selecting_rows]

        #Replicating data
        Replacing_rows = np.random.choice(303, 203)
        Replicated_sample_data = sample_data[Replacing_rows]
        target_of_Replicated_sample_data = target_data[Replacing_rows]

        #Concatenation
        final_sample_data = np.vstack((sample_data,Replicated_sample_data))
        final_target_data = np.vstack((target_of_sample_data.reshape(-1,1),target_of_Re

        return final_sample_data,final_target_data,Selecting_rows,Selecting_columns

    list_input_data =[]
    list_output_data =[]
    list_selected_row= []
    list_selected_columns=[]

    for i in range(0,30):
        a,b,c,d = generating_samples(x,y)
        list_input_data.append(a)
        list_output_data.append(b)
        list_selected_row.append(c)
        list_selected_columns.append(d)

    from sklearn.tree import DecisionTreeRegressor

    list_of_all_models = []

    for i in range(len(list_input_data)):
        dt = DecisionTreeRegressor(max_depth = None)
        trained_dt = dt.fit(list_input_data[i],list_output_data[i])
        list_of_all_models.append(trained_dt)

    from sklearn.metrics import mean_squared_error

    array_of_Y = []

    for k in range(len(list_of_all_models)):

        y_predicted = list_of_all_models[k].predict(list_input_data[k])

        array_of_Y.append(y_predicted)

    array_of_Y = np.array(array_of_Y)

    predicted_y = []

    for j in range(array_of_Y.shape[1]):

        sorted_array_of_Y = np.sort(array_of_Y[:,j])
        median = ( (sorted_array_of_Y[14] + sorted_array_of_Y[15]) / 2)

        predicted_y.append(median)


    MSE_35.append(mean_squared_error(y , predicted_y ))

    predicted_y = []

    for j in range(array_of_Y.shape[1]):


        sorted_array_of_Y = np.sort(array_of_Y[final_model_list[j],j])  #from array of

        median = np.median(sorted_array_of_Y)

        predicted_y.append(median)


    OOB_35.append(mean_squared_error(y , predicted_y ))

print("35 MSE Scores : \n" , MSE_35)
print("\n35 OOB Scores : \n" ,OOB_35)
```

```
35 MSE Scores :
 [62.87163587690403, 60.35236882548309, 59.79737173501318, 58.55734358805446, 61.6072886
4832737, 59.13110365597571, 60.85119559956448, 59.462603955314, 59.58577976538483, 60.28
560263543 2085, 60.992275470510286, 59.55276001376097, 58.57498569938515, 60.222766533541
936, 62.57789076654909, 58.5964568190851, 60.062999633563905, 60.3442502566 4251, 57.7855
2229501027, 57.590621983118, 60.17272204744875, 59.28634695734519, 58.93810888489843, 6
0.61029242589493, 60.9463873836524, 60.401829611989996, 60.686077669356614, 59.259083845
24593, 62.27745807502021, 57.702811970703415, 59.317539797682905, 61.60275844962491, 58.
1902653101392, 61.75259881327696, 60.40014180075935]

35 OOB Scores :
 [69.43917390594916, 63.79730142594422, 61.169187136028654, 61.05299072655906, 64.721161
85161234, 66.25589068979342, 64.18830108970135, 64.81108712532937, 65.78733633069828, 6
6.12273521674413, 63.23785816631881, 63.15940151715171, 64.07090561182477, 64.7759007232
6527, 69.89834648065349, 64.8960658338669, 65.25904114102987, 61.71226485918972, 64.5052
3379442906, 61.24813392075373, 64.79482293946563, 66.41368525609354, 63.329238909263246,
66.49039194000227, 64.5371852912443, 65.28065537495262, 63.45298277898775, 63.5508875452
8986, 64.17475977921364, 60.815683701751816, 64.84481757109134, 65.22717109552853, 62.64
6836398117976, 67.39895942989679, 67.54913715928875]
```

In [61]:

```
#Confidence Interval using the python notebook on central limit theorem

s_MSE_std = np.std(np.array(MSE_35))
x_MSE_mean = np.round(np.mean(np.array(MSE_35)), 3)
size_MSE = len(MSE_35)

left_limit  = np.round(x_MSE_mean - 2*(s_MSE_std/np.sqrt(size_MSE)), 3)
right_limit  = np.round(x_MSE_mean + 2*(s_MSE_std/np.sqrt(size_MSE)), 3)

print("95% CI of MSE : " , [left_limit,right_limit])


s_OOB_std = np.std(np.array(OOB_35))
x_OOB_mean = np.round(np.mean(np.array(OOB_35)), 3)
size_OOB = len(OOB_35)

left_limit  = np.round(x_OOB_mean - 2*(s_OOB_std/np.sqrt(size_OOB)), 3)
```

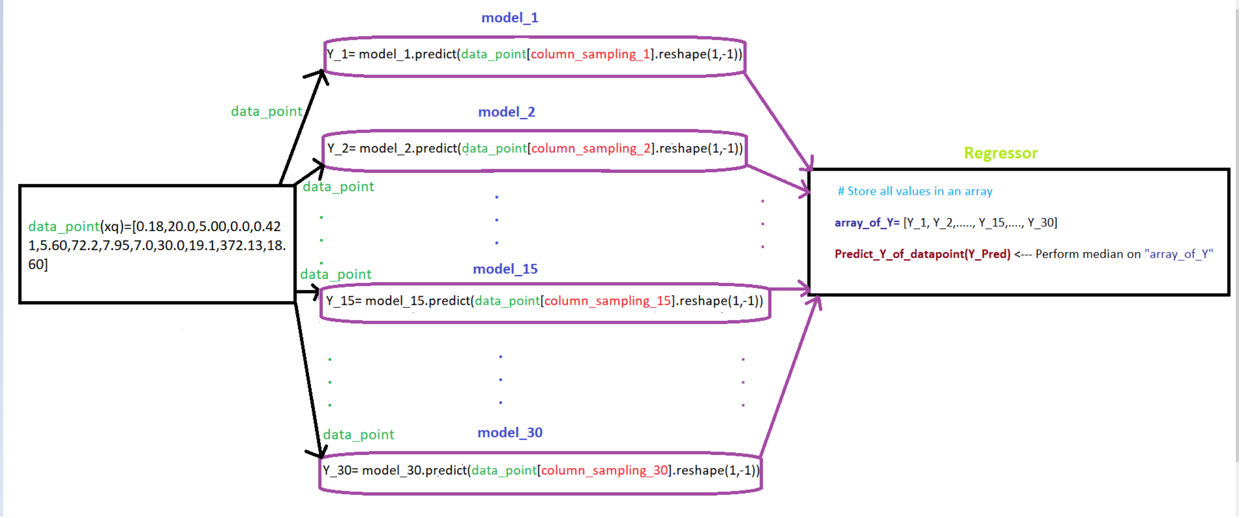
```
right_limit  = np.round(x_OOB_mean + 2*(s_OOB_std/np.sqrt(size_OOB)), 3)

print("95% CI of OOB : " , [left_limit,right_limit])
```

```
95% CI of MSE :  [59.56, 60.46]
95% CI of OOB :  [63.879, 65.299]
```

# Task 3

### Flowchart for Task 3

**Hint: We created 30 models by using 30 samples in TASK-1. Here, we need send query point "xq" to 30 models and perform the regression on the output generated by 30 models.**



```
data_point(xq)=[0.18,20.0,5.00,0.0,0.42
1,5.60,72.2,7.95,7.0,30.0,19.1,372.13,18.
60]

Y_1= model_1.predict(data_point[column_sampling_1].reshape(1,-1))
Y_2= model_2.predict(data_point[column_sampling_2].reshape(1,-1))
Y_15= model_15.predict(data_point[column_sampling_15].reshape(1,-1))
Y_30= model_30.predict(data_point[column_sampling_30].reshape(1,-1))

# Store all values in an array
array_of_Y= [Y_1, Y_2,....., Y_15,...., Y_30]
Predict_Y_of_datapoint(Y_Pred) <--- Perform median on "array_of_Y"
```

- **Write code for TASK 3**

In [62]:
```
def generating_samples(input_data, target_data):

    Selecting_rows = np.random.choice(506 ,303 ,replace=False)
    Selecting_columns = np.random.choice([0,1,2,3,4,5,6,7,8,9,10,11,12], np.random.choi
    sample_data = input_data[ Selecting_rows[:,None] , Selecting_columns ]
    target_of_sample_data = target_data[Selecting_rows]

    #Replicating data
    Replacing_rows = np.random.choice(303, 203)
    Replicated_sample_data = sample_data[Replacing_rows]
    target_of_Replicated_sample_data = target_data[Replacing_rows]

    #Concatenation
    final_sample_data = np.vstack((sample_data,Replicated_sample_data))
    final_target_data = np.vstack((target_of_sample_data.reshape(-1,1),target_of_Replic

    return final_sample_data,final_target_data,Selecting_rows,Selecting_columns

list_input_data =[]
list_output_data =[]
list_selected_row= []
list_selected_columns=[]

for i in range(0,30):
    a,b,c,d = generating_samples(x,y)
    list_input_data.append(a)
    list_output_data.append(b)
    list_selected_row.append(c)
    list_selected_columns.append(d)

from sklearn.tree import DecisionTreeRegressor

list_of_all_models = []

for i in range(len(list_input_data)):
    dt = DecisionTreeRegressor(max_depth = None)
    trained_dt = dt.fit(list_input_data[i],list_output_data[i])
    list_of_all_models.append(trained_dt)

from sklearn.metrics import mean_squared_error

array_of_Y = []

for k in range(len(list_of_all_models)):

    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.array(xq)

    nfeat = list_of_all_models[k].n_features_

    y_predicted = list_of_all_models[k].predict(  xq[:,:nfeat] )

    array_of_Y.append(y_predicted)

array_of_Y = np.array(array_of_Y)

predicted_y = []

for j in range(array_of_Y.shape[1]):

    sorted_array_of_Y = np.sort(array_of_Y[:,j])

    median = ( (sorted_array_of_Y[14] + sorted_array_of_Y[15]) / 2)

    predicted_y.append(median)

print("Price of the house for given Xq: " , predicted_y)
```

```
Price of the house for given Xq:  [23.049999999999997]
```

## Write observations for task 1, task 2, task 3 indetail

**Task 1 Observations :**

**MSE is 63.697 OOB is 71.796**

**Task 2 Observations :**

**35 MSE Scores : [62.87163587690403, 60.35236882548309, 59.79737173501318, 58.55734358805446, 61.60728864832737, 59.13110365597571, 60.85119559956448, 59.462603955314, 59.58577976538483, 60.285602635432085, 60.992275470510286,**

**59.55276001376097, 58.57498569938515, 60.222766533541936, 62.57789076654909, 58.5964568190851, 60.062999633563905, 60.34425025664251, 57.78552229501027, 57.590621983118, 60.17272204744875, 59.28634695734519, 58.93810888489843, 60.61029242589493, 60.9463873836524, 60.401829611989996, 60.686077669356614, 59.25908384524593, 62.27745807502021, 57.702811970703415, 59.317539797682905, 61.60275844962491, 58.1902653101392, 61.75259881327696, 60.40014180075935]**

**35 OOB Scores : [69.43917390594916, 63.79730142594422, 61.169187136028654, 61.05299072655906, 64.72116185161234, 66.25589068979342, 64.18830108970135, 64.81108712532937, 65.78733633069828, 66.12273521674413, 63.23785816631881, 63.15940151715171, 64.07090561182477, 64.77590072326527, 69.89834648065349, 64.8960658338669, 65.25904114102987, 61.71226485918972, 64.50523379442906, 61.24813392075373, 64.79482293946563, 66.41368525609354, 63.329238909263246, 66.49039194000227, 64.5371852912443, 65.28065537495262, 63.45298277898775, 63.55088754528986, 64.17475977921364, 60.815683701751816, 64.84481757109134, 65.22717109552853, 62.646836398117976, 67.39895942989679, 67.54913715928875]**

**Task 3 Observations :**

**Price of the house for given Xq: [23.05]**