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.

Importing packages

In [1]:

```
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 [2]:

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

In [3]:

```
x.shape, y.shape
```

Out[3]:

```
((506, 13), (506,))
```

In [4]:

```
x[0], y[0]
```

Out[4]:

```
(array([6.320e-03, 1.800e+01, 2.310e+00, 0.000e+00, 5.380e-01, 6.575e+00,
        6.520e+01, 4.090e+00, 1.000e+00, 2.960e+02, 1.530e+01, 3.969e+02,
        4.980e+00]), 24.0)
```

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
- **Note - While selecting the random 60% datapoints from the whole data, make sure that the selected datapoints are all exclusive, repetition is not allowed.**

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
  - 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

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.

A few key points

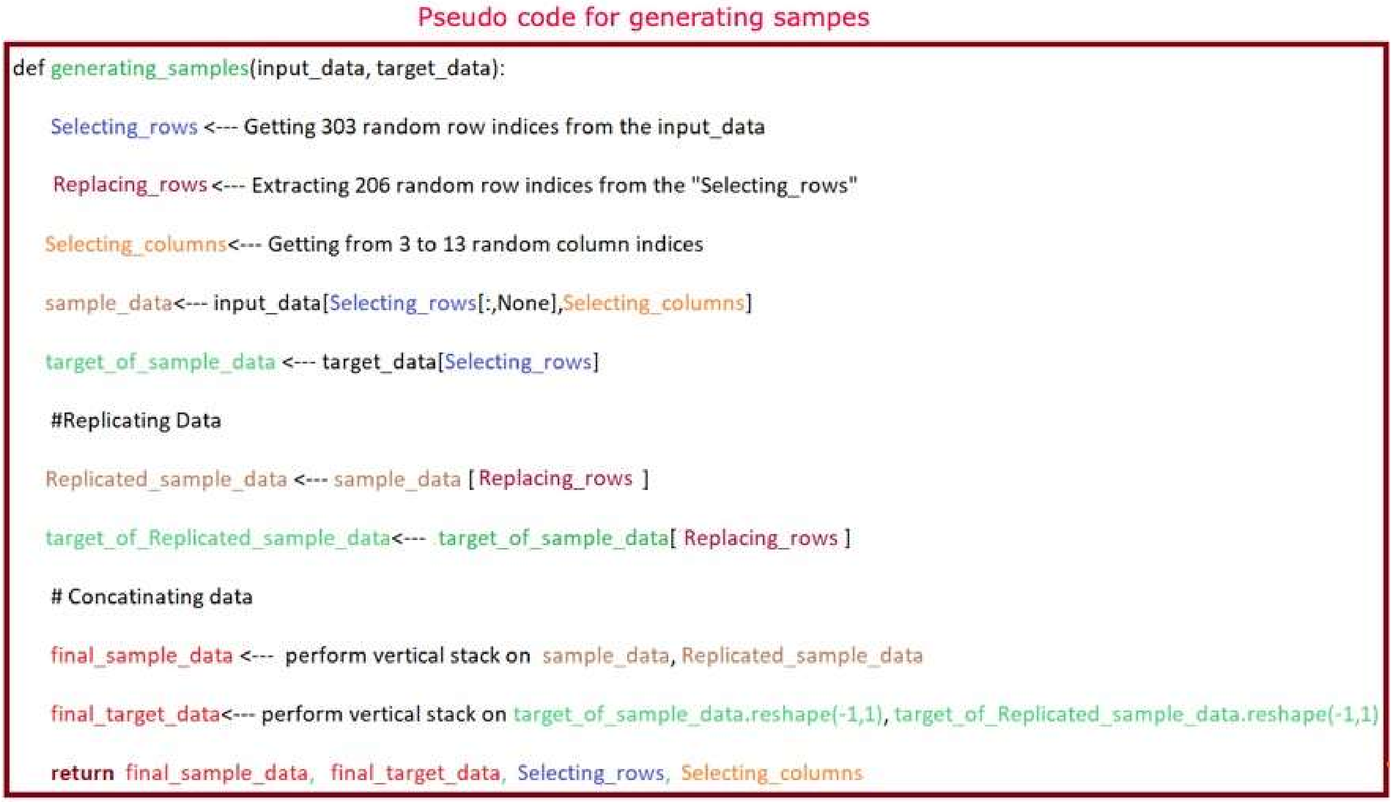
- Remember that the datapoints used for calculating MSE score contain some datapoints that were initially used while training the base learners (the 60% sampling). This makes these datapoints partially seen (i.e. the datapoints used for calculating the MSE score are a mixture of seen and unseen data). Whereas, the datapoints used for calculating OOB score have only the unseen data. This makes these datapoints completely unseen and therefore appropriate for testing the model's performance on unseen data.
- Given the information above, if your logic is correct, the calculated MSE score should be less than the OOB score.
- The MSE score must lie between 0 and 10.
- The OOB score must lie between 10 and 35.
- The difference between the left nad right confidence-interval values must not be more than 10. Make sure this is true for both MSE and OOB confidence-interval values.

Task - 1

Step - 1

- **Creating samples**

Algorithm



Pseudo code for generating sampes

```
def generating_samples(input_data, target_data):

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

    Replacing_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 [Replacing_rows ]

    target_of_Replicated_sample_data<---  target_of_sample_data[ Replacing_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 [5]:

```
import numpy as np
import random

'''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/gen
    # 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(x, y):
    row_indices = np.random.choice(x.shape[0], size=303, replace=False)
    row_replacing_indices = np.random.choice(row_indices, size=203, replace=False)
    selecting_rows = x[row_indices, :]
    selecting_rows_y = y[row_indices]
    selecting_replacing_rows = x[row_replacing_indices, :]
    selecting_replacing_rows_y = y[row_replacing_indices]
    sample_rw_data = np.array(x)
    sample_rw_data_y = np.array(y)
    sample_rw_data = np.append(selecting_rows,selecting_replacing_rows,axis=0)
    sample_rw_data_y = np.append(selecting_rows_y,selecting_replacing_rows_y,axis=0)
    col_var = random.choice(range(3,13))
    col_indices = np.random.choice(x.shape[1], size=col_var, replace=False)
    final_sample_data = sample_rw_data[:,col_indices]
    final_target_data = sample_rw_data_y
    return final_sample_data,final_target_data,row_indices,col_indices
```

Grader function - 1

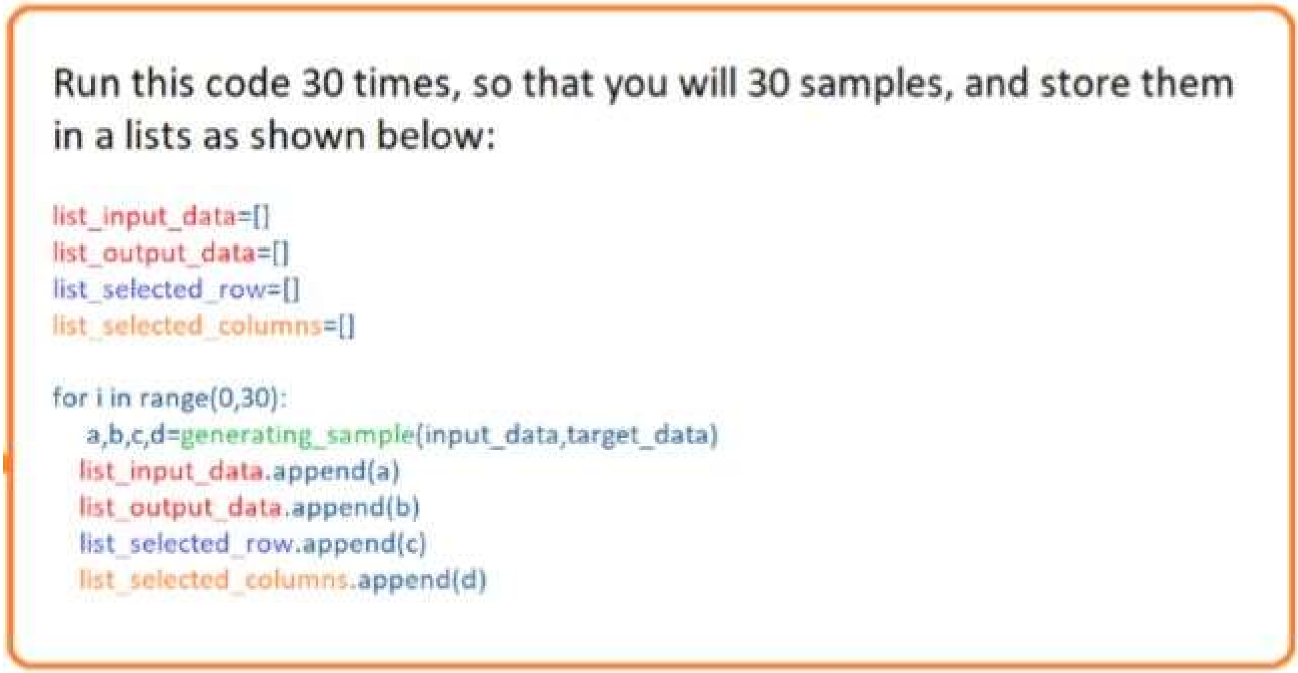
In [6]:

```
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[6]:

```
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 [7]:

```
# 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 [8]:

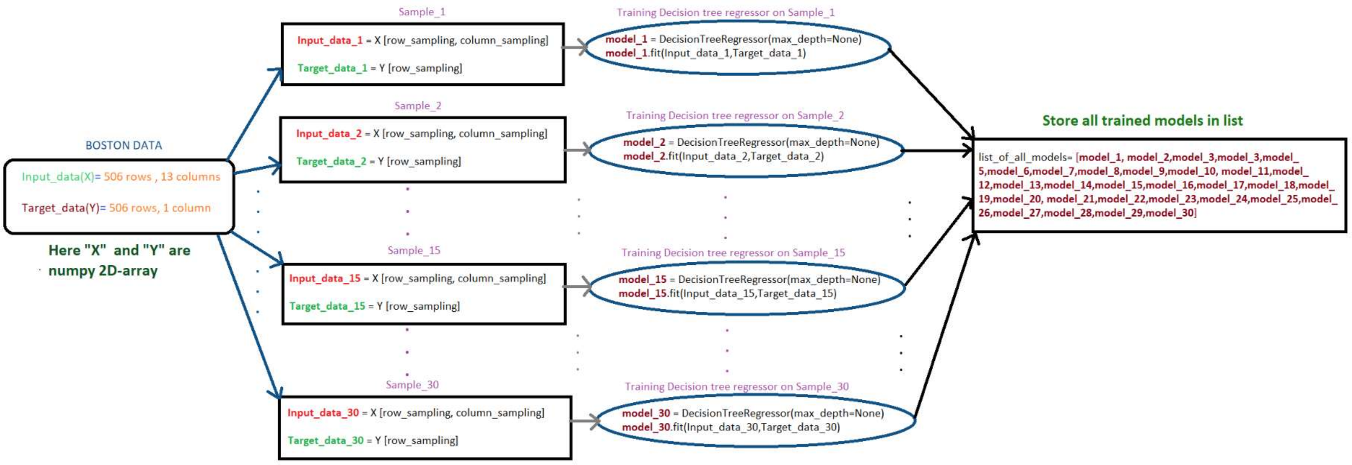
```
def grader_30(a):
    assert(len(a)==30 and len(a[0])==506)
    return True
grader_30(list_input_data)
```

Out[8]:

```
True
```

Step - 2

Flowchart for building tree

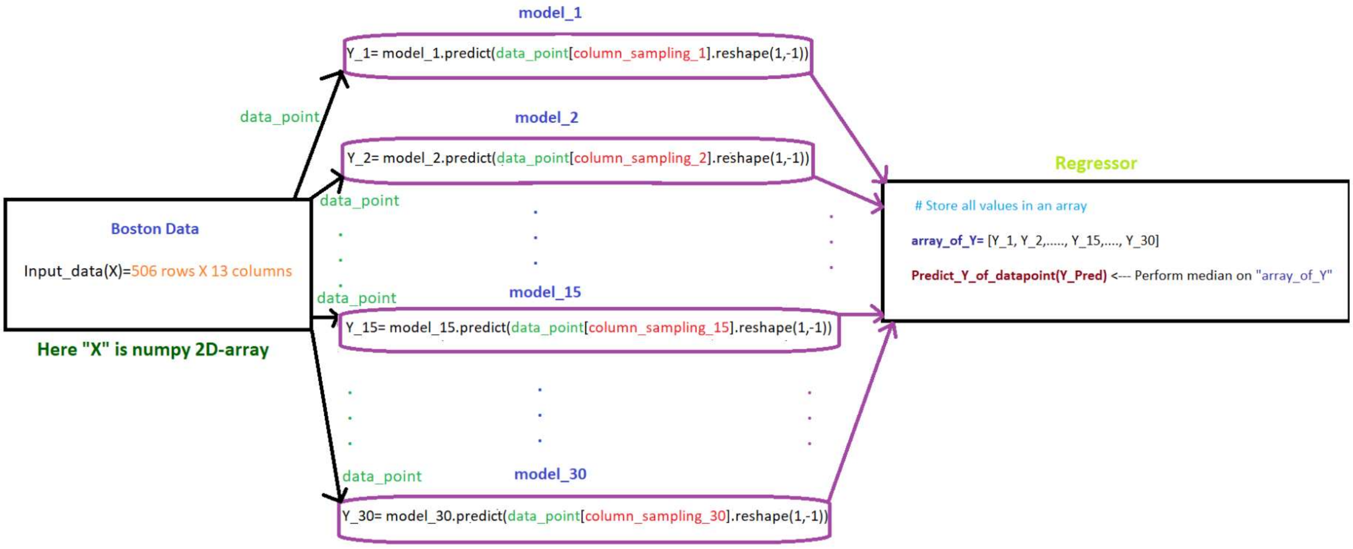

BOSTON DATA
Input_data(X)= 506 rows , 13 columns
Target_data(Y)= 506 rows, 1 column
Here "X" and "Y" are numpy 2D-array
Sample_1
Input_data_1 = X [row_sampling, column_sampling]
Target_data_1 = Y [row_sampling]
Sample_2
Input_data_2 = X [row_sampling, column_sampling]
Target_data_2 = Y [row_sampling]
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 [9]:

```
from sklearn.tree import DecisionTreeRegressor
rgrsr_modl_lst = []
for i in range(0,30):
    Model_i = DecisionTreeRegressor(max_depth=None)
    Model_i.fit(list_input_data[i],list_output_data[i])
    rgrsr_modl_lst.append(Model_i)
```

Flowchart for calculating MSE


Boston Data
Input_data(X)=506 rows X 13 columns
Here "X" is numpy 2D-array
data_point
data_point
data_point
data_point
model_1
Y_1= model_1.predict(data_point[column_sampling_1].reshape(1,-1))
model_2
Y_2= model_2.predict(data_point[column_sampling_2].reshape(1,-1))
model_15
Y_15= model_15.predict(data_point[column_sampling_15].reshape(1,-1))
model_30
Y_30= model_30.predict(data_point[column_sampling_30].reshape(1,-1))
Regressor
# 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"

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**

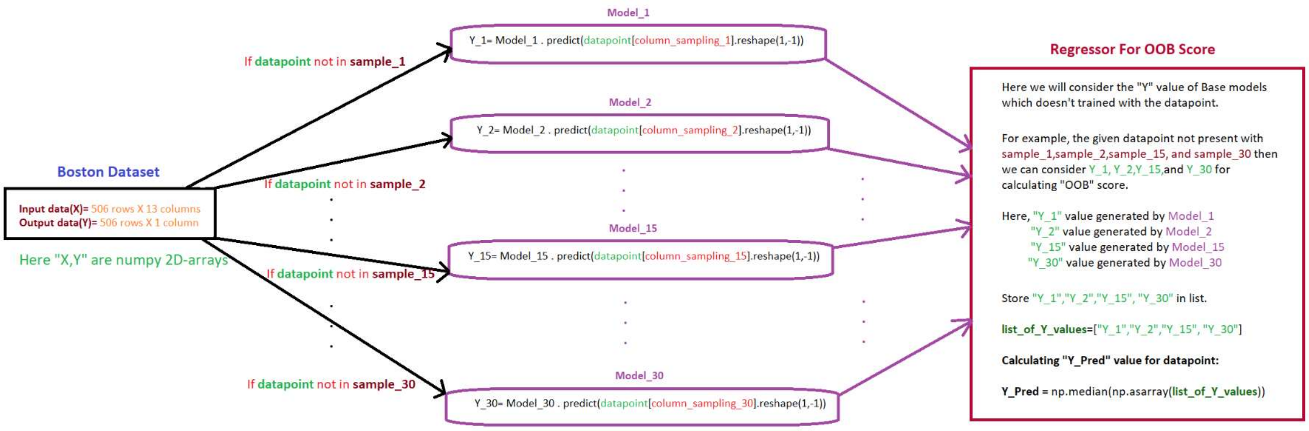
In [10]:

```
data_pt_med_lst = []
for k in range(len(x)):
    y_pred_1 = []
    for i in range(0,30):
        data_pt = x[:,list_selected_columns[i]][k]
        y_pred_2 = rgrsr_modl_lst[i].predict(data_pt.reshape(1,-1))
        sq_err = (y[k]-y_pred_2)**2
        y_pred_1.append(sq_err)
    data_pt_med = np.median(np.array(y_pred_1))
    data_pt_med_lst.append(data_pt_med)

print("Mean square error is:" + str(np.mean(np.array(data_pt_med_lst))))
```

```
Mean square error is:0.09013660274239976
```

Step - 3

Flowchart for calculating OOB score


Model_1
Y_1= Model_1 . predict(datapoint[column_sampling_1].reshape(1,-1))
If datapoint not in sample_1
Model_2
Y_2= Model_2 . predict(datapoint[column_sampling_2].reshape(1,-1))
Boston Dataset
If datapoint not in sample_2
Input data(X)= 506 rows X 13 columns
Output data(Y)= 506 rows X 1 column
Here "X,Y" are numpy 2D-arrays
Model_15
Y_15= Model_15 . predict(datapoint[column_sampling_15].reshape(1,-1))
If datapoint not in sample_15
Model_30
If datapoint not in sample_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 [11]:

```
eachdatapt_notprsnt_samples = []
for i in range(len(x)):
    data_notprsnt_samples = []
    for j in range(0,30):
        data_pt = x[:,list_selected_columns[j]][i]
        if (data_pt!=list_input_data[j][i]).all():
            data_notprsnt_samples.append(j)
    eachdatapt_notprsnt_samples.append(data_notprsnt_samples)
for i in range(len(x)):
    y_pred_1 = []
    lst = eachdatapt_notprsnt_samples[i]
    for k in range(len(lst)):
        data_pt = x[:,list_selected_columns[lst[k]]][i]
        y_pred_2 = rgrsr_modl_lst[lst[k]].predict(data_pt.reshape(1,-1))
        sq_err = (y[i]-y_pred_2)**2
        y_pred_1.append(sq_err)
    data_pt_med = np.median(np.array(y_pred_1))
    data_pt_med_lst.append(data_pt_med)

print("OOB score is:" + str(np.mean(np.array(data_pt_med_lst))))
```

```
OOB score is:0.6813962383496804
```

Task 2

In [12]:

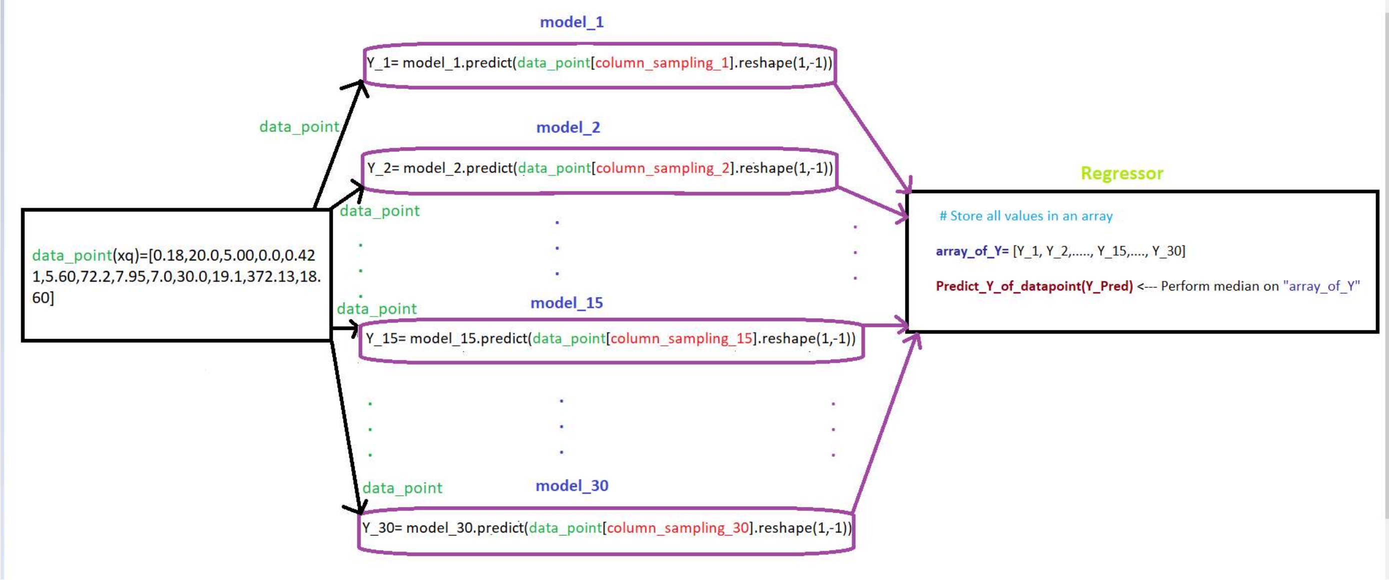
```
MSE_sample_lst = []
OOB_sample_lst = []

for k in range(0,35):
    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)
    rgrsr_modl_lst = []
    for i in range(0,30):
        Model_i = DecisionTreeRegressor(max_depth=None)
        Model_i.fit(list_input_data[i],list_output_data[i])
        rgrsr_modl_lst.append(Model_i)
    data_pt_med_lst = []
    for k in range(len(x)):
        y_pred_1 = []
        for i in range(0,30):
            data_pt = x[:,list_selected_columns[i]][k]
            y_pred_2 = rgrsr_modl_lst[i].predict(data_pt.reshape(1,-1))
            sq_err = (y[k]-y_pred_2)**2
            y_pred_1.append(sq_err)
        data_pt_med = np.median(np.array(y_pred_1))
        data_pt_med_lst.append(data_pt_med)
    MSE_sample = np.mean(np.array(data_pt_med_lst))
    MSE_sample_lst.append(MSE_sample)
    eachdatapt_notprsnt_samples = []
    for i in range(len(x)):
        data_notprsnt_samples = []
        for j in range(0,30):
            data_pt = x[:,list_selected_columns[j]][i]
            if (data_pt!=list_input_data[j][i]).all():
                data_notprsnt_samples.append(j)
        eachdatapt_notprsnt_samples.append(data_notprsnt_samples)
    for i in range(len(x)):
        y_pred_1 = []
        lst = eachdatapt_notprsnt_samples[i]
        for k in range(len(lst)):
            data_pt = x[:,list_selected_columns[lst[k]]][i]
            y_pred_2 = rgrsr_modl_lst[lst[k]].predict(data_pt.reshape(1,-1))
            sq_err = (y[i]-y_pred_2)**2
            y_pred_1.append(sq_err)
        data_pt_med = np.median(np.array(y_pred_1))
        data_pt_med_lst.append(data_pt_med)
    OOB_sample = np.mean(np.array(data_pt_med_lst))
    OOB_sample_lst.append(OOB_sample)

mu_mse = np.mean(np.array(MSE_sample_lst))
sig_mse = np.std(np.array(MSE_sample_lst))
mu_oob = np.mean(np.array(OOB_sample_lst))
sig_oob = np.std(np.array(OOB_sample_lst))
print("95% CI for MSE is: "+"["+str(mu_mse-2*sig_mse)+","+str(mu_mse+2*sig_mse)+"]")
print("95% CI for OOB is: "+"["+str(mu_oob-2*sig_oob)+","+str(mu_oob+2*sig_oob)+"]")
```

```
95% CI for MSE is: [0.01910580691958691,0.6418164375045516]
95% CI for OOB is: [0.13856953150818496,2.3150198344862116]
```

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.


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))
data_point
data_point(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]
data_point
model_15
Y_15= model_15.predict(data_point[column_sampling_15].reshape(1,-1))
data_point
model_30
Y_30= model_30.predict(data_point[column_sampling_30].reshape(1,-1))
Regressor
# 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 [13]:

```
x = np.array([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]).reshape(
data_pt_med_lst = []
y_pred_1 = []
for i in range(0,30):
    data_pt = x[:,list_selected_columns[i]]
    y_pred_2 = rgrsr_modl_lst[i].predict(data_pt.reshape(1,-1))
    y_pred_1.append(y_pred_2)
data_pt_med = np.median(np.array(y_pred_1))

print("Median of Regressor is:" + str(data_pt_med))
```

```
Median of Regressor is:18.6
```

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

In [ ]:

```
1. Mean square error is less than OOB score
```