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
```

Out[3]:
```
(506, 13)
```

In [4]:
```
x[:5]
```

Out[4]:
```
array([[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]])
```

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
  - 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.

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 [5]:

```python
def generating_samples(input_data, target_data):

    '''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/generated/numpy.random.cl
    # Please follow above pseudo code for generating samples

    #input_data represents the independent_feature and target_data represents the output feature/ dependent_featu

    #select the row indices for the input data randomly without replacement
    select_row_indices = np.random.choice(input_data.shape[0],303,replace = False)
    #print(select_row_indices)
    #repeat the selected indices to to construct the remaing datapoint
    repeating_rows = np.random.randint(0,len(select_row_indices),203)

    #select the column incices without replacement #np.random.randint generate the random number between 3 to 13
    select_column_indices = np.random.choice(input_data.shape[1],np.random.randint(3,13,1)[0],replace = False)

    #https://github.com/numpy/numpy/issues/13255 broadcasting error for multi dimentional mask
    sample_data = input_data[select_row_indices][:,select_column_indices]

    #slicing the output feature rows
    target_sample_data = target_data[select_row_indices]


    #Replicating the data

    #https://www.geeksforgeeks.org/numpy-vstack-in-python/
    replicated_sample_data = sample_data[repeating_rows]
    #print(replicated_sample_data.shape)

    target_of_replicated_sample_data = target_data[repeating_rows]

    #merge the data sample
    final_sample_data = np.vstack((sample_data,replicated_sample_data))

    #merge the output feature
    final_target_data = np.vstack((target_sample_data.reshape(-1,1),target_of_replicated_sample_data.reshape(-1,1

    return final_sample_data,final_target_data,select_row_indices,select_column_indices
    # return sampled_input_data , sampled_target_data,selected_rows,selected_columns
    #note please return as lists
```

Grader function - 1 </fongt>

In [6]:

```python
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

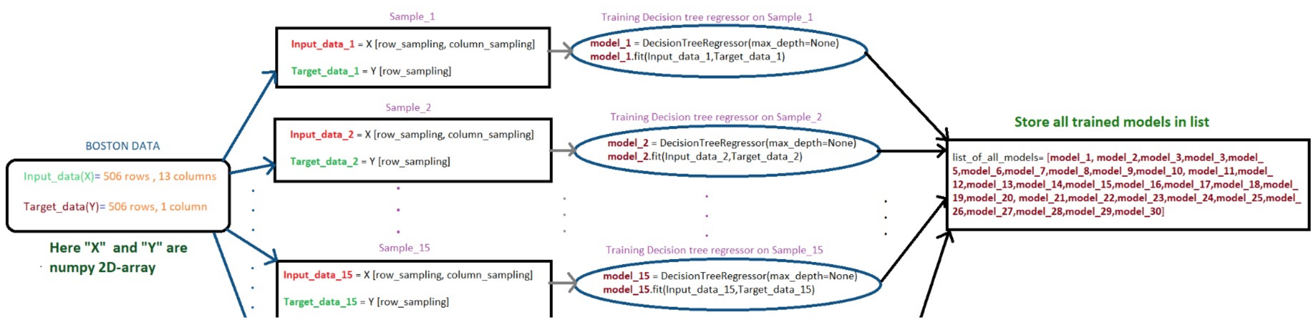
In [8]:

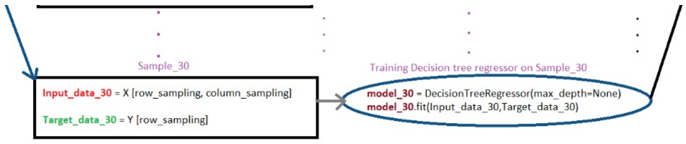
```
def grader_30(a):
    assert(len(a)==30 and len(a[0])==506)
    return True
grader_30(list_input_data)
```

Out[8]: True

Step - 2

Flowchart for building tree

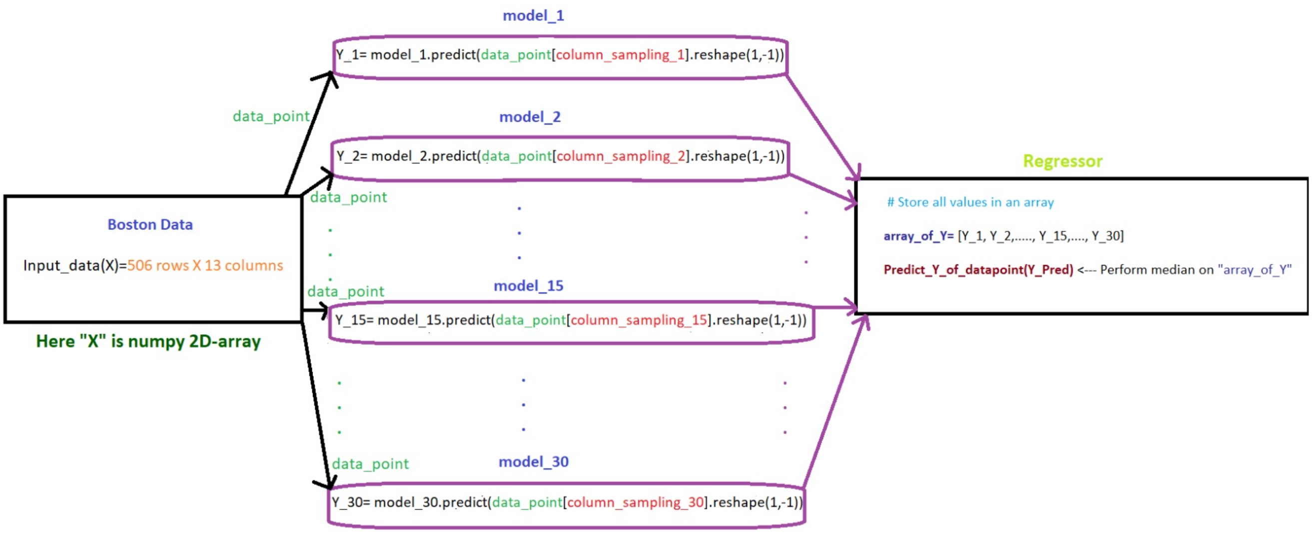




- **Write code for building regression trees**

In [9]:
```python
from sklearn.tree import DecisionTreeRegressor

def regression_tree(list_input_data,list_output_data):

    model_list = []

    for idx in range(len(list_input_data)):
        #print(x_train.shape,y_train.shape)
        tree = DecisionTreeRegressor(max_depth = None)

        model = tree.fit(list_input_data[idx],list_output_data[idx])
        model_list.append(model)
    return model_list
list_model = regression_tree(list_input_data,list_output_data)
```

Flowchart for calculating MSE



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]:
```python
def model_performance(input_data,list_model,list_selected_column):

    y_predict_list = []

    #prediction for the original_data,#Iterate Each model with selected column
    for model,sampled_column in zip(list_model,list_selected_column):

        #print(len(sampled_column))
        y_pred = model.predict(input_data[:,sampled_column])

        #append the each model prediction
        y_predict_list.append(y_pred)

    return np.array(y_predict_list)

#Pediction for all the 30 model
y_predict_list = model_performance(x,list_model,list_selected_columns)
```

```
print("Shape of the predicted Models :{0}".format(y_predict_list.shape))
```

```
Shape of the predicted Models :(30, 506)
```

In [12]:

```
#print(type(y_predict_list))

#Compute the Median of the 30 model prediction
#Aggregate the all model into single model to predict the o/p
#In regression We use aggregation as Median/mean to dececide the predicted class label
mean_of_y_pred = np.mean(y_predict_list.T,axis = 1)
print("Shape of the aggregated Model {0}".format(mean_of_y_pred.reshape(-1,1).shape))
#print(y.reshape(-1,1).shape)
#compute the performance of the model mean_squared_error
```

```
Shape of the aggregated Model (506, 1)
```

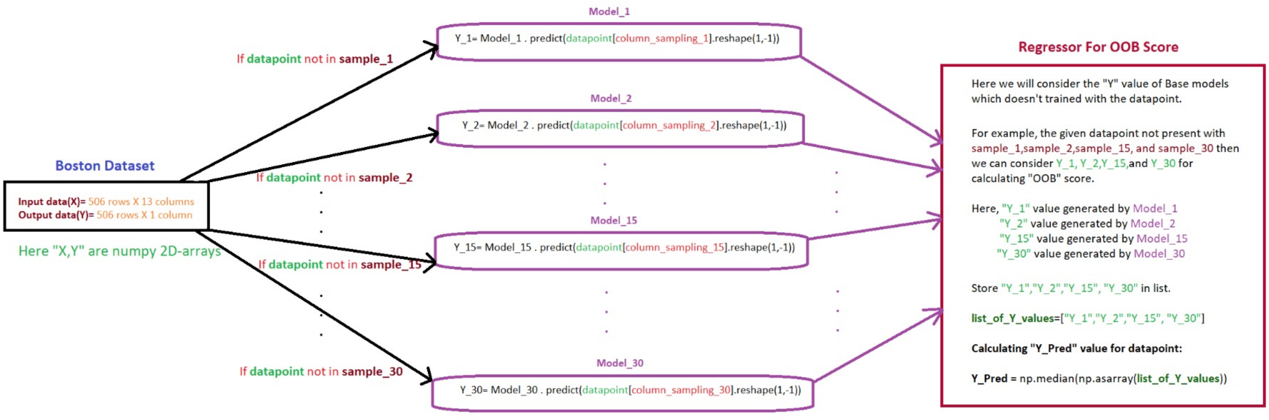
In [13]:

```
def mean_square_error(actual_value,predicted_value):
    mse =  0.0
    #check  the lenght length both the value
    if len(actual_value) == len(predicted_value):

        for i in range(len(actual_value)):

            mse += (actual_value[i] - predicted_value[i]) ** 2


    mse  = mse /  len(actual_value)

    return mse

mean_squared_error  = mean_square_error(y,mean_of_y_pred)
#print('Mean square error : {0}'.format(mean_squared_error(y.reshape(-1,1),median_of_y_pred.reshape(-1,1))))
print('Mean square error : {0}'.format(mean_squared_error))
```

```
Mean square error : 12.778534811209928
```

Step - 3

Flowchart for calculating OOB score



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

- **Write code for calculating OOB score**

In [14]:

```
#https://towardsdatascience.com/what-is-out-of-bag-oob-score-in-random-forest-a7fa23d710
def predict_data_on_oob(x,y,list_of_model,selecting_row_index_list,selecting_column_list):

    oob_predicted_score = []
```

```python
    y_predict_list = []


    # iterate all the data samples
    for idx_row in range(x.shape[0]):

        predict_oob_data_model = []

        predicted_unseen_query_point_list = []

        y_prediction_models = []

        subset_data_model_sampling_column = []

        #select the model have the index is not present in the model indices
        for idx_model in range(len(list_of_model)):

            if idx_row not in selecting_row_index_list[idx_model]:

                #example the corresponding row is not present in the given  subset of the sample and  select the
                predict_oob_data_model.append(list_of_model[idx_model])

                #get the sampling columns corresponding model and convert array to list
                subset_data_model_sampling_column.append(selecting_column_list[idx_model].tolist())

                #print(subset_data_model_sampling_column)

        #Iterate selected model which is the data is present in the sample
        for idx in range(len(predict_oob_data_model)):

            #query
            unseen_data = x[idx_row]

            #sampling the columns -->> corresponding to the model
            unseen_data = unseen_data[subset_data_model_sampling_column[idx]]

            #predict the given query not seen in the training data
            #print(type(predict_oob_query))
            #print(predict_oob_query)

            #predict the  new query point not present in the samples
            y_predict_unseen_data = predict_oob_data_model[idx].predict(np.array(unseen_data).reshape(1,-1))


            #predict_oob_data_model.append(predict_oob_query)

            predicted_unseen_query_point_list.append(y_predict_unseen_data)


        #Predicted house price of  ith  data point  take the average of the query point
        mean_of_the_datapoint = np.median(predicted_unseen_query_point_list)
        #print(predicted_unseen_query_point_list.shape)
        y_predict_list.append(mean_of_the_datapoint)

    return np.array(y_predict_list)


def calculate_oob_score(actual_value,predicted_value):
    #out of bag _score
    oobScore =  0.0
    if len(actual_value) == len(predicted_value):

        for i in range(len(actual_value)):
            size = actual_value[i] - predicted_value[i]
            oobScore += (actual_value[i] - predicted_value[i]) ** 2


    oobScore  = oobScore /  len(actual_value)

    return oobScore


y_prediction_list = predict_data_on_oob(x,y,list_model,list_selected_row,list_selected_columns)

oobScore = calculate_oob_score(y,y_prediction_list)

print("Shape of the y_prediction_list: {0}".format(y_prediction_list.shape))

print("OOB Score : {0}".format(oobScore))
```

```
Shape of the y_prediction_list: (506,)
OOB Score : 22.70470218763723
```

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

In [15]:

```
#Compute CI of OOBSCore and Train MSE

#Genrating 35 Samples to compute the confindence Intereval

def generate_sample_to_compute_confidence_interval(x,y):


    mean_squared_error_sample_list = []

    oob_Score_sample_list = []

    for i 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)

        #generate k regression trees
        list_model = regression_tree(list_input_data,list_output_data)

        #predict the data
        y_predict_list = model_performance(x,list_model,list_selected_columns)

        mean_of_y_pred = np.mean(y_predict_list.T,axis = 1)
        #compute the Mean squaredError for the predicted model
        mean_squared_error  = mean_square_error(y,mean_of_y_pred)

        #predict the unseen data in training time
        y_prediction_list = predict_data_on_oob(x,y,list_model,list_selected_row,list_selected_columns)

        #compute oob_score for the predicted model
        oobScore = calculate_oob_score(y,y_prediction_list)

        #add the mean square error in the list
        mean_squared_error_sample_list.append(mean_squared_error)


        oob_Score_sample_list.append(oobScore)

    return (mean_squared_error_sample_list,oob_Score_sample_list)
```

In [16]:

```
mean_squared_error_sample_list, oob_Score_sample_list= generate_sample_to_compute_confidence_interval(x,y)
```

In [18]:

```
import math
n = 35
#step 1 : caculate its mean  of sample
mean_squared_error_sample_list = np.array(mean_squared_error_sample_list)

oob_Score_sample_list =  np.array(oob_Score_sample_list)
sample_mean_of_mse = np.mean(mean_squared_error_sample_list)

sample_mean_of_oob = np.mean(oob_Score_sample_list)


#Step 2. compute the std deviation of the sample std

std_mse = np.std(mean_squared_error_sample_list)
```

```
std_oob = np.std(oob_Score_sample_list)

#std_error = sample_std / sqroot(n)
std_error_mse  = std_mse / math.sqrt(n)

std_error_oob = std_oob / math.sqrt(n)

#print(std_mse)

print("Confidence Interval for Mean Squared Error : {0}".format([np.round(sample_mean_of_mse - (2 * std_error_mse
                                                                 np.round(sample_mean_of_mse + (2 *std_error_mse)

print("Confidence Interval for OOB Score : {0}".format([np.round(sample_mean_of_oob - (2 *std_error_oob),3), \
                                                                 np.round(sample_mean_of_oob + (2 *std_error_oob)
```

```
Confidence Interval for Mean Squared Error : [12.207, 12.734]
Confidence Interval for OOB Score : [23.305, 24.696]
```

Observation

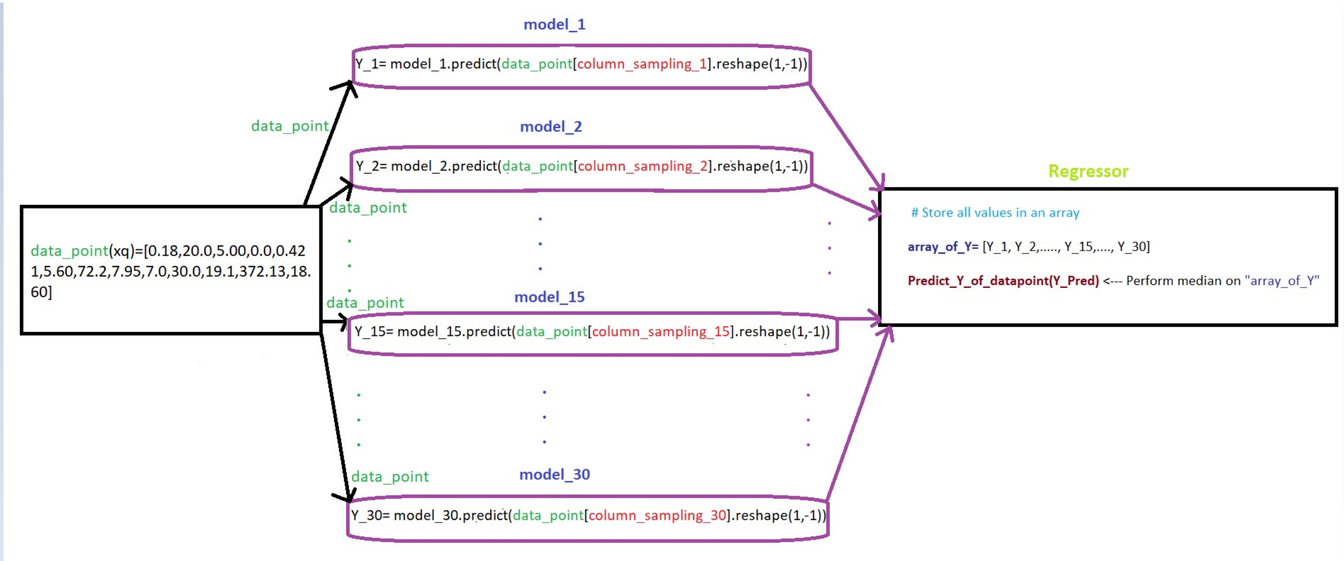
1.we can say 95% confidence to perform error(MES) in the range between [12.207, 12.734] population datasat with 35 samples MSE

2.we can say 95% confidence to OOB Score in the range between [23.305, 24.696] to unseen data based on the sample mean and std population datasat

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.



- **Write code for TASK 3**

In [19]:
```
#Predict the house price - unseen data
def predict_new_data(x_q,model_list,list_selected_columns):

    y_predict_list = []

    for i in range(len(model_list)):

        Y_predict = model_list[i].predict(x_q[list_selected_columns[i]].reshape(1,-1))

        y_predict_list.append(Y_predict)

    return np.mean(np.array(y_predict_list))
```

In [21]:
```
x_q = 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])
predicted_price = predict_new_data(x_q,list_model,list_selected_columns)
```

```
print("Predicted value of House Price : {0}".format(predicted_price))
```

```
Predicted value of House Price : 23.243333333333336
```

Processing math: 100%