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 [145]:

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

In [146]:

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

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 [147]:

```
import numpy as np
import random
```

In [148]:

```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.choice.html for details

  selected_rows = np.random.choice(len(input_data), 303, replace = False)
  selected_rows = np.sort(selected_rows)


  replaced_rows = np.random.choice(selected_rows, 203, replace = False)
  replaced_rows = np.sort(replaced_rows)

  selected_columns = np.sort(np.random.choice(input_data.shape[1], size=random.randint(3, input_data.shape[1]), replace =False))

  sample_data = input_data[selected_rows[:,None],selected_columns]
  target_sameple_data = target_data[selected_rows]

  replicating_data = input_data[replaced_rows[:,None],selected_columns]
  target_replicating_data = target_data[replaced_rows]

  final_sample_data = np.vstack((sample_data, replicating_data))
  final_target_data = np.vstack((target_sameple_data.reshape(-1,1),target_replicating_data.reshape(-1,1)))
  return final_sample_data, final_target_data, selected_rows, selected_columns
```

- **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 [150]:

```
# 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):
  final_sample_data, final_target_data, selected_rows, selected_columns = genera
ting_samples(x,y)
  #grader_samples(final_sample_data , final_target_data,selected_rows,selected_c
olumns)
  # Append the info in the respective list.
  list_input_data.append(final_sample_data)
  list_output_data.append(final_target_data)
  list_selected_row.append(selected_rows)
  list_selected_columns.append(selected_columns)
```

Grader function - 1 </fongt>

In [151]:

```
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)
 #print (length, sampled, rows_length, column_length)
 assert(length and sampled and rows_length and column_length)

 return True
```

In [152]:

```
a,b,c,d = generating_samples(x, y)
grader_samples(a,b,c,d)
```

Out[152]:

```
True
```

Grader function - 2

In [153]:

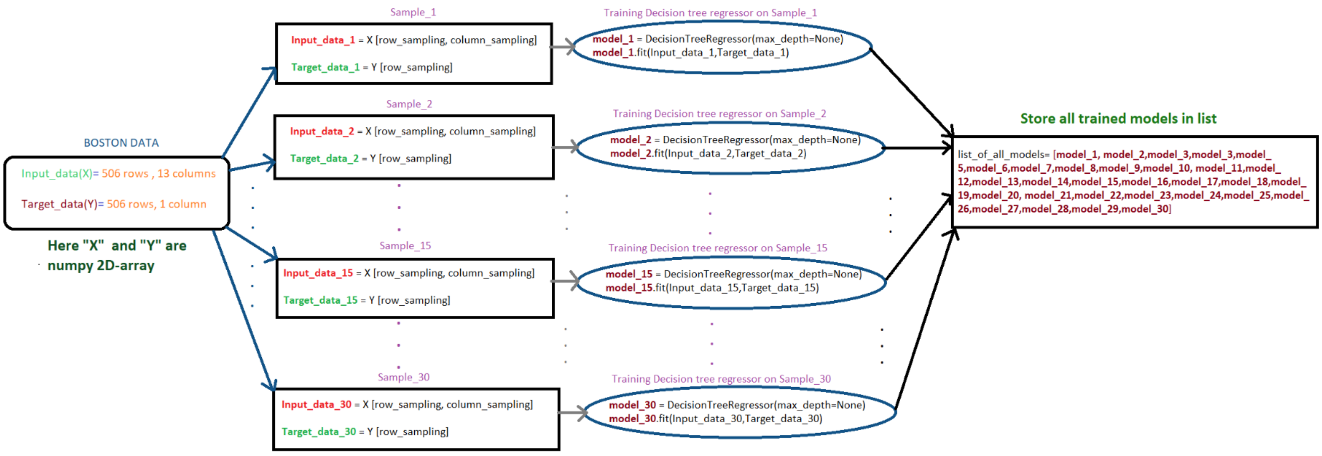
```
def grader_30(a):
  assert(len(a)==30 and len(a[0])==506)
  return True
grader_30(list_input_data)
```

Out[153]:

```
True
```

Step - 2

Flowchart for building tree

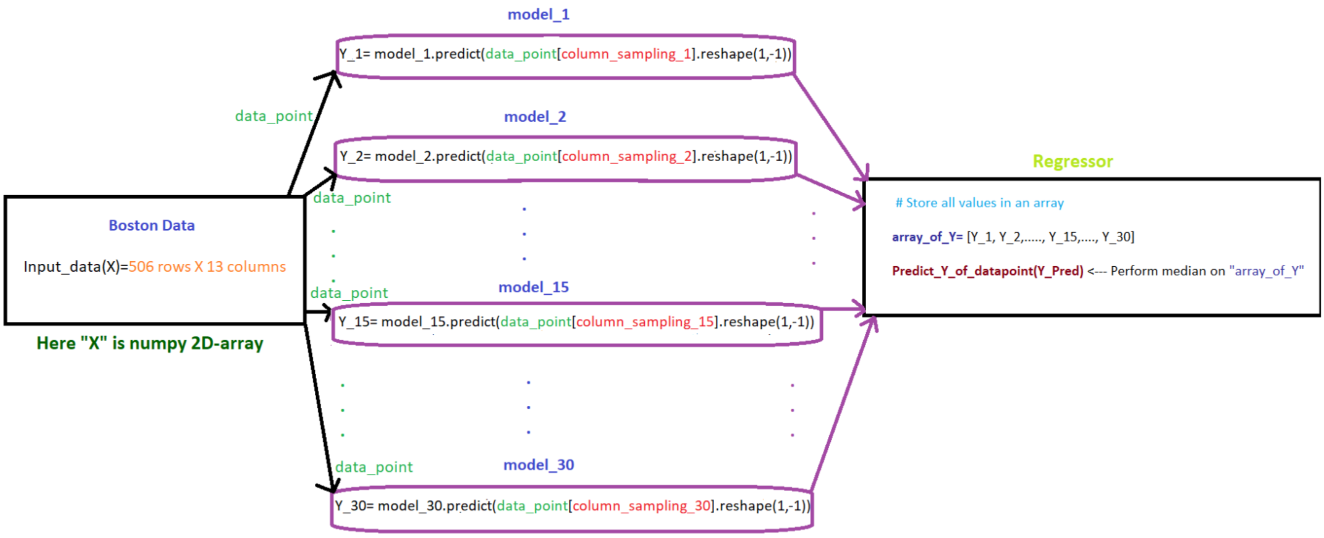


- **Write code for building regression trees**

In [154]:

```
from sklearn.tree import DecisionTreeRegressor
list_of_all_models = []
for i in range(0, 30):
  input_data = list_input_data[i]
  target_data = list_output_data[i]
  classifier = DecisionTreeRegressor(max_depth=None,min_samples_split=2)
  classifier.fit(input_data, target_data)
  list_of_all_models.append(classifier)
```

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 [155]:

```
import statistics

list_predict = []
for i in range(0,30):
  pred_y = list_of_all_models[i].predict(x[:,list_selected_columns[i]])
  list_predict.append(pred_y)

# Calculating median for each data point from predicted y of each model
final_y_predict = []

for i in range(0, 506):
  med_y = []
  for j in range(0, 30):
    med_y.append(list_predict[j][i])
  med = statistics.median(med_y)
  final_y_predict.append(med)
```

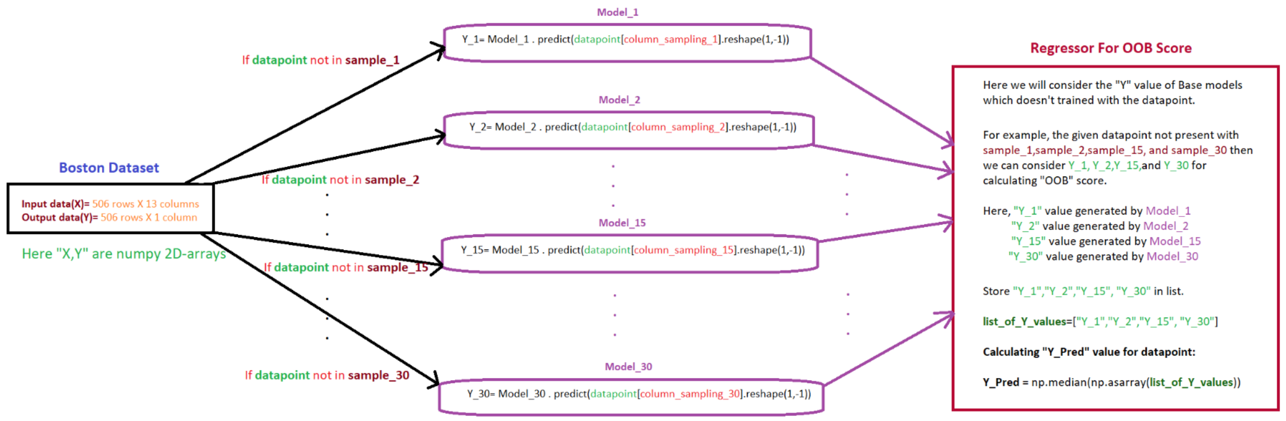
In [156]:

```
# Calculating MSE :
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y, final_y_predict)
print("=" * 30)
print("MSE on train data set : " , mse)
print("=" * 30)
```

```
==============================
MSE on train data set :  0.06123023715415015
==============================
```

Step - 3

Flowchart for calculating 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** $OOBScore = \frac{1}{506} \sum_{i=1}^{506} (y^i - y^i_{pred})^2$.

- **Write code for calculating OOB score**

In [157]:

```
oob_y_predict = []
y_sample_predict = []
# iterating through each data point.
for i in range (0,506):
  data = x[i]
  for j in range(0,30):
    if i not in list_selected_row[j]:
      pred_y = list_of_all_models[j].predict(data[list_selected_columns[j]].resh
ape(1,-1))
      y_sample_predict.append(pred_y[0])
  y_sample_predict_oobs = statistics.median(y_sample_predict)
  y_sample_predict.clear()
  oob_y_predict.append(y_sample_predict_oobs)

#print((oob_y_predict))
#Calculate OObs score now :
total_error = 0
for i in range(0, 506):
  error = y[i] - oob_y_predict[i]
  error = error * error #Square error :
  total_error = total_error + error
oobs_score = total_error/506
print("=" * 30)
print("OOBS Score : " , oobs_score)
print("=" * 30)
```

```
==============================
OOBS Score :  14.860715612648221
==============================
```

Task 2

In [158]:

```
# repeat Task 1 35 times :
train_mse_list = []
oobs_score_list = []
for iteration in range (1 , 36):
  list_input_data =[]
  list_output_data =[]
  list_selected_row= []
  list_selected_columns=[]
  list_of_all_models = []
  list_predict = []
  oob_y_predict = []
  y_sample_predict = []

  for i in range(0,30):
    final_sample_data, final_target_data, selected_rows, selected_columns = gene
rating_samples(x,y)
    # Append the info in the respective list.
    list_input_data.append(final_sample_data)
    list_output_data.append(final_target_data)
    list_selected_row.append(selected_rows)
    list_selected_columns.append(selected_columns)

  #Write code for building regression trees
  for i in range(0, 30):
    input_data = list_input_data[i]
    target_data = list_output_data[i]
    classifier = DecisionTreeRegressor(max_depth=None,min_samples_split=2)
    classifier.fit(input_data,target_data)
    list_of_all_models.append(classifier)

  # write code for Calculating MSE :
  for i in range(0,30):
    pred_y = list_of_all_models[i].predict(x[:,list_selected_columns[i]])
    list_predict.append(pred_y)
    # Calculating median for each data point from predicted y of each model.
  final_y_predict = []
  for i in range(0 , 506):
    med_y = []
    for j in range(0 , 30):
      med_y.append(list_predict[j][i])
    med = statistics.median(med_y)
    final_y_predict.append(med)

  # Calculating MSE :
  mse = mean_squared_error(y, final_y_predict)
  print("=" * 30)
  print("Iteration = : " , iteration)
  print("MSE on train data " , mse)
  train_mse_list.append(mse)
  # Calculating OOBS score
  # iterating through each data point.
  for i in range (0,506):
    data = x[i]
    for j in range(0,30):
      if i not in list_selected_row[j]:
        pred_y = list_of_all_models[j].predict(data[list_selected_columns[j]].re
shape(1,-1))
        y_sample_predict.append(pred_y[0])
    y_sample_predict_oobs = statistics.median(y_sample_predict)
```

```
    y_sample_predict.clear()
    oob_y_predict.append(y_sample_predict_oobs)

  #Calculate OObs score now :
  total_error = 0
  for i in range(0,506):
    error = y[i] - oob_y_predict[i] #Square error :
    error = error * error
    total_error += error
  oobs_score = total_error/506
  print("OOBS Score : " , oobs_score)
  print("=" * 30)
  oobs_score_list.append(oobs_score)
```

```
==============================
Iteration = :  1
MSE on train data  0.026442687747035606
OOBS Score :  11.246935055972319
==============================
==============================
Iteration = :  2
MSE on train data  0.26442687747035565
OOBS Score :  13.73833498023715
==============================
==============================
Iteration = :  3
MSE on train data  0.1078557312252965
OOBS Score :  12.350770750988143
==============================
==============================
Iteration = :  4
MSE on train data  0.020286561264822124
OOBS Score :  18.15286190711461
==============================
==============================
Iteration = :  5
MSE on train data  0.008922924901185781
OOBS Score :  13.91498023715414
==============================
==============================
Iteration = :  6
MSE on train data  0.03548812212632089
OOBS Score :  18.222345231029685
==============================
==============================
Iteration = :  7
MSE on train data  0.02956027667984188
OOBS Score :  12.81495553359683
==============================
==============================
Iteration = :  8
MSE on train data  0.0753471219807576
OOBS Score :  12.660466036279141
==============================
==============================
Iteration = :  9
MSE on train data  0.06647233201581033
OOBS Score :  15.012462597032545
==============================
==============================
Iteration = :  10
MSE on train data  0.10600029386112711
OOBS Score :  11.484972135454287
==============================
==============================
Iteration = :  11
MSE on train data  0.06942826704545457
OOBS Score :  14.426508966746411
==============================
==============================
Iteration = :  12
MSE on train data  0.1593972332015811
OOBS Score :  16.883961379097094
==============================
==============================
```

```
Iteration = :  13
MSE on train data  0.02001482213438734
OOBS Score :  13.755258167578615
==============================
==============================
Iteration = :  14
MSE on train data  0.19947472184222984
OOBS Score :  14.79439018104647
==============================
==============================
Iteration = :  15
MSE on train data  0.09996496135875665
OOBS Score :  15.696652499850611
==============================
==============================
Iteration = :  16
MSE on train data  0.1985129040160731
OOBS Score :  15.255477704411375
==============================
==============================
Iteration = :  17
MSE on train data  0.06558684672815097
OOBS Score :  15.843335651436242
==============================
==============================
Iteration = :  18
MSE on train data  0.0696257411067194
OOBS Score :  13.573347194773818
==============================
==============================
Iteration = :  19
MSE on train data  0.06585474308300392
OOBS Score :  12.965644685990327
==============================
==============================
Iteration = :  20
MSE on train data  0.2344144553093252
OOBS Score :  14.719237355377253
==============================
==============================
Iteration = :  21
MSE on train data  0.036400966183574864
OOBS Score :  15.549275674158613
==============================
==============================
Iteration = :  22
MSE on train data  0.04368083003952572
OOBS Score :  12.955169219367583
==============================
==============================
Iteration = :  23
MSE on train data  0.12095553908651732
OOBS Score :  15.689476154118573
==============================
==============================
Iteration = :  24
MSE on train data  0.025498076630727184
OOBS Score :  13.978845048662876
==============================
==============================
Iteration = :  25
```

```
MSE on train data  0.06297430830039526
OOBS Score :  13.688147233201576
==============================
==============================
Iteration = :  26
MSE on train data  0.057588932806324135
OOBS Score :  14.646519474112514
==============================
==============================
Iteration = :  27
MSE on train data  0.11796048666007895
OOBS Score :  14.528702565902709
==============================
==============================
Iteration = :  28
MSE on train data  0.1276784145805885
OOBS Score :  10.740686896135262
==============================
==============================
Iteration = :  29
MSE on train data  0.011413043478260857
OOBS Score :  13.841020471357599
==============================
==============================
Iteration = :  30
MSE on train data  0.04966403162055339
OOBS Score :  13.043169416996047
==============================
==============================
Iteration = :  31
MSE on train data  0.0885365393061045
OOBS Score :  12.970766020446739
==============================
==============================
Iteration = :  32
MSE on train data  0.027707509881422898
OOBS Score :  14.344813610135162
==============================
==============================
Iteration = :  33
MSE on train data  0.048103315766359345
OOBS Score :  14.544954185275262
==============================
==============================
Iteration = :  34
MSE on train data  0.02906140206411946
OOBS Score :  12.964700160573123
==============================
==============================
Iteration = :  35
MSE on train data  0.2214495443566097
OOBS Score :  15.409598986043923
==============================
```

Caclulate COnfidence Interval

In [160]:

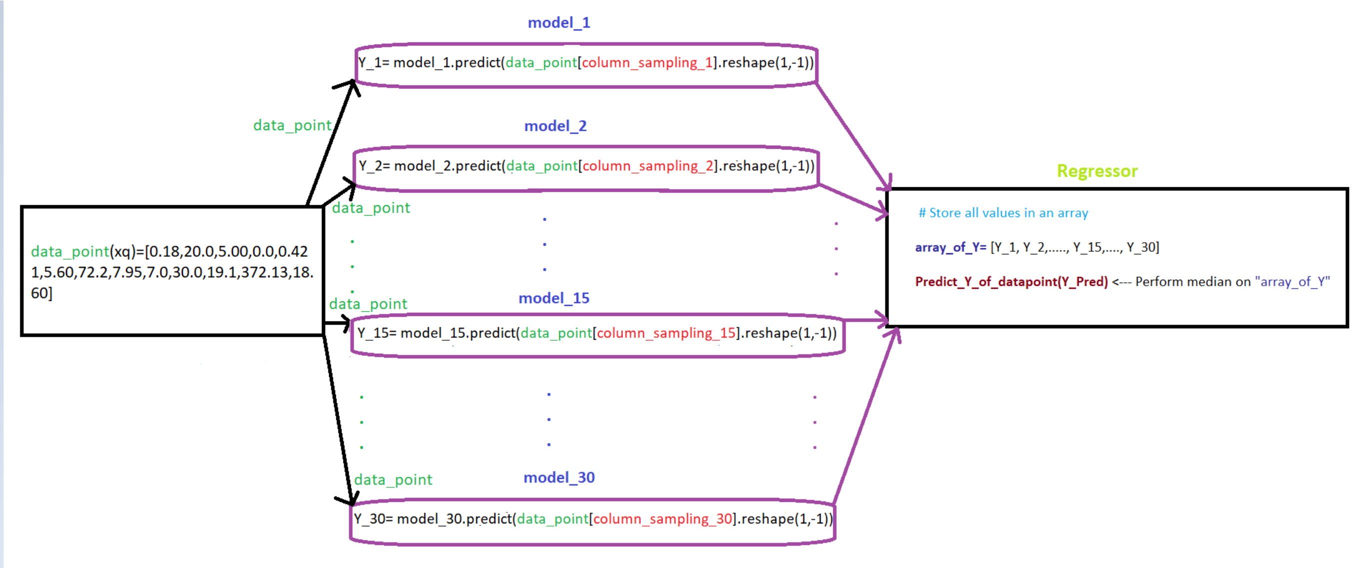
```
#Converting list to array
import math
train_mse = np.asarray(train_mse_list)
oobs_score = np.asarray(oobs_score_list)

# Compute mean
mean_mse = np.mean(train_mse)
mean_oobs = np.mean(oobs_score)

#Compute STD:
std_mse = np.std(train_mse)
std_oobs = np.std(oobs_score)

#Compute Standard error:
sqrt_n = math.sqrt(30)
standard_error_mse = std_mse/sqrt_n
standard_error_obs = std_oobs/sqrt_n

#CI for MSE:
lower_limit_mse = mean_mse - 2 * (standard_error_mse)
upper_limit_mse = mean_mse + 2 * (standard_error_mse)
print("=" * 30)
print("CI for MSE :" , lower_limit_mse , "," , upper_limit_mse)

#CI for MSE :
lower_limit_oobs = mean_oobs - 2 * (standard_error_obs)
upper_limit_oobs = mean_oobs + 2 * (standard_error_obs)
print("CI for oob :" , lower_limit_oobs , "," , upper_limit_oobs)
print("=" * 30)
```

```
==============================
CI for MSE : 0.06093333519577696 , 0.11002383999618859
CI for oob : 13.575609080006858 , 14.790604826716264
==============================
```

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 [161]:

```
xq = np.array([0.18,20.0,5.00,0.0,0.421,5.60,72.2,7.95,7.0,30.0,19.1,372.13,18.6
0])
total_price = 0
for j in range(0,30):
  pred_y = list_of_all_models[j].predict(xq[list_selected_columns[j]].reshape(1,
-1))
  total_price += pred_y

# Not predicted price for the query point will be average of all prices predicte
d.
predicted_price = total_price/30
print("=" * 50)
print("Predicted price for query point xq is :" , predicted_price)
print("=" * 50)
```

```
==================================================
Predicted price for query point xq is : [19.54333333]
==================================================
```