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]:
```
def generating_samples(input_data, target_data):
  lnth = len(input_data)
  selecting_rows = random.sample(range(0,lnth),int(lnth*.6))
  replacing_rows = random.sample(selecting_rows,lnth-len(selecting_rows))
  # print(replacing_rows,'\n',len(replacing_rows))
  col_length = len(input_data[0])
  # print(col_length)
  selecting_columns = sorted(random.sample(range(0,13),np.random.randint(3,12
)))
  #     print("selecting_rows-->",selecting_rows)
  #     print("selecting_columns-->",selecting_columns)
  sample_data = input_data[selecting_rows][:,selecting_columns]
  target_of_sample_data = target_data[selecting_rows]
  #     print("sample_data")
  #     print(sample_data)
  replicated_sample_data = input_data[replacing_rows][:,selecting_columns]
  #     print("replicated_sample_data")
  #     print(replicated_sample_data)
  target_of_replicated_sample_data = target_data[replacing_rows]
  final_sample_data = np.vstack((sample_data,replicated_sample_data))
  final_target_data = np.vstack((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_column
s
```

Grader function - 1 </fongt>

In [6]:
```
import random
```

In [8]:
```
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[8]:
```
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 [9]:
```
# 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)
```

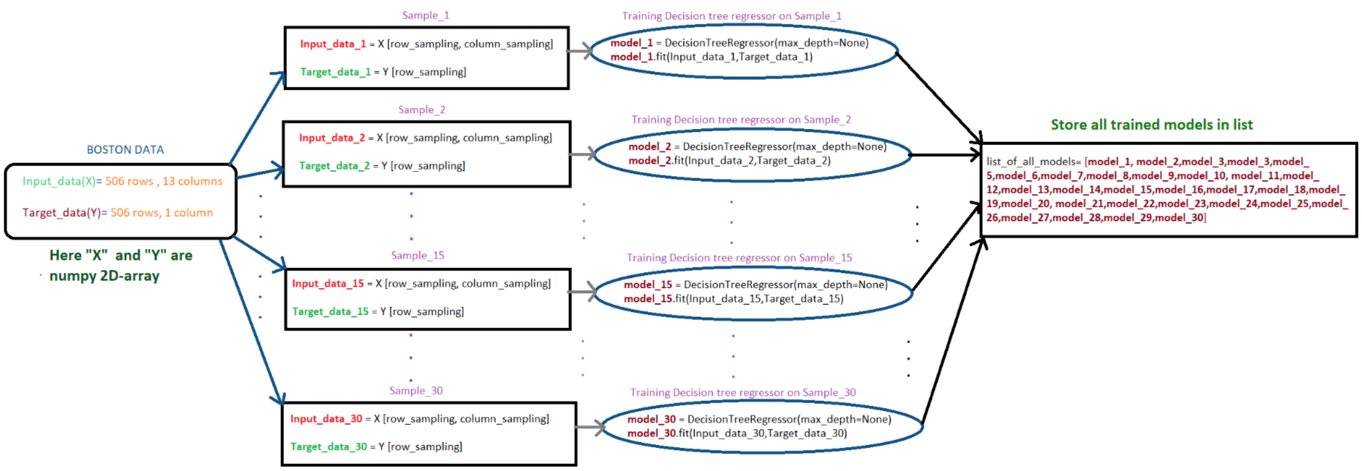
Grader function - 2

In [10]:
```
def grader_30(a):
    assert(len(a)==30 and len(a[0])==506)
    return True
grader_30(list_input_data)
```

Out[10]: True

Step - 2

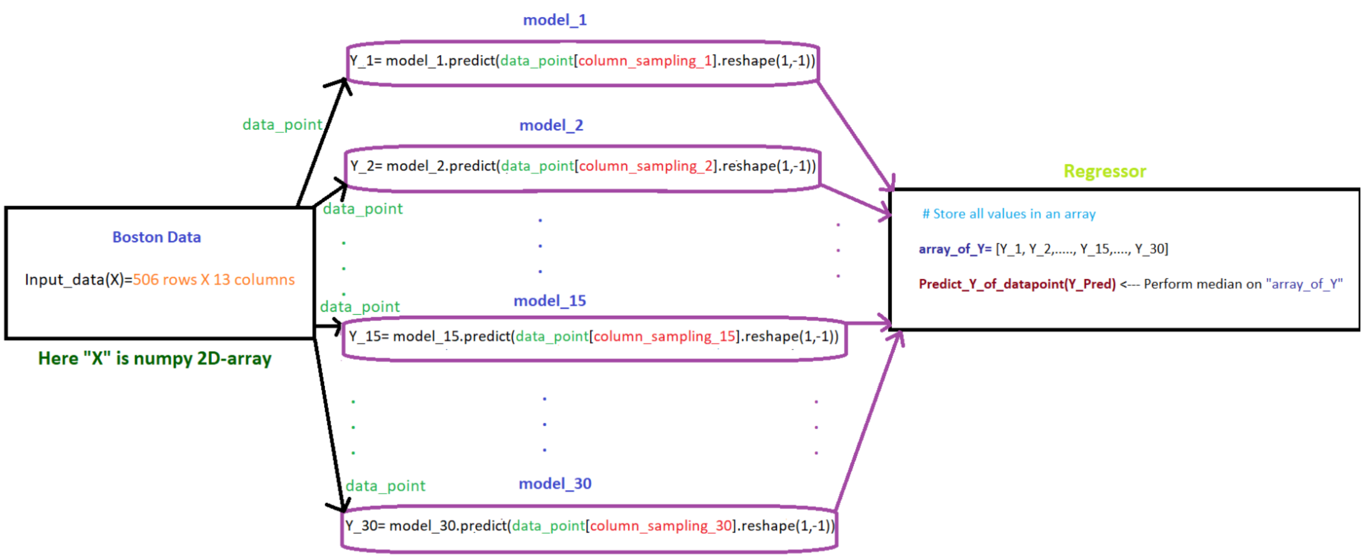
Flowchart for building tree



- **Write code for building regression trees**

In [11]:
```
# https://stackoverflow.com/questions/4010840/generating-variable-names-on-fly-in-python
from sklearn.tree import DecisionTreeRegressor
list_of_all_models=[]
for i in range(1,31):
  globals()['reg_tree_{}'.format(i)]=DecisionTreeRegressor(max_depth=None)
  globals()['reg_tree_{}'.format(i)].fit(list_input_data[i-1],list_output_data[i-1])
  list_of_all_models.append(globals()['reg_tree_{}'.format(i)])
```

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 [12]:
```
array_of_Y=[]
for i in range(1,31):
  globals()['Y_{}'.format(i)]=globals()['reg_tree_{}'.format(i)].predict(list_
input_data[i-1])
  array_of_Y.append(globals()['Y_{}'.format(i)])
```

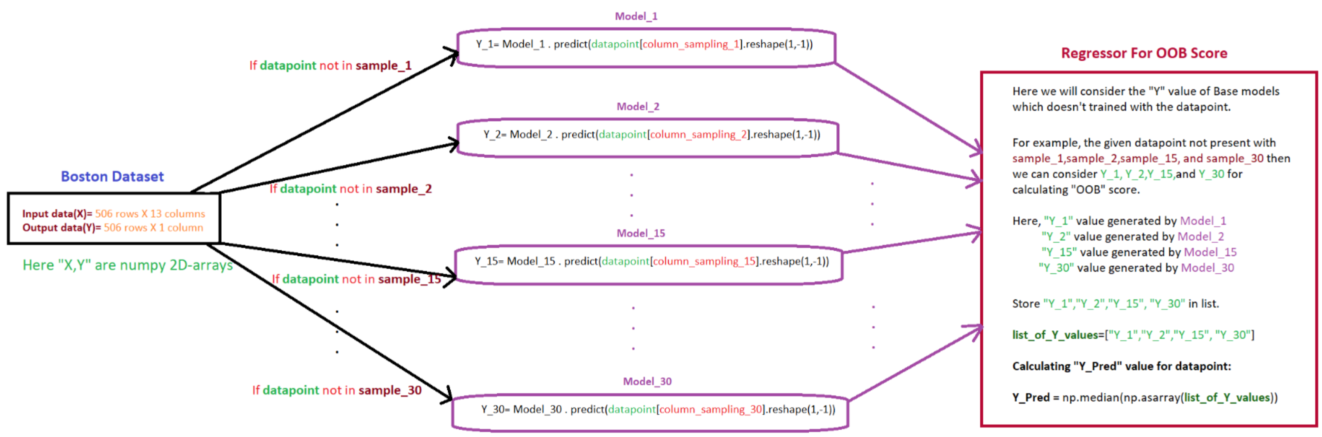
In [13]:
```
y_pred=np.median(array_of_Y,axis=0)
```

In [14]:
```
from sklearn.metrics import mean_squared_error
mean_squared_error(y,y_pred)
```

Out[14]:
```
87.37856284188904
```

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 [15]:
```
def mean(array):
  return sum(array)/max(len(array),1)
```

In [16]:
```
y_oob=[]
for i in range(len(x)): #looping boston whole dataset
  y_oob_internal=[] #temporary arrays to hold oob value
  for indx,j in enumerate(list_selected_row): # looping for all trees
    if i not in j:   # calculating prediction value if ith point not in select
ed records of the particular model,
      y_oob_internal.append(globals()['reg_tree_{}'.format(indx+1)].predict(x[
i][list_selected_columns[indx]].reshape(1,-1))[0]) # appending predicted value
y_oob_internal array
  y_oob.append(mean(y_oob_internal)) # calculating mean value returned by each
model and appending to main array.
```

In [17]:
```
oob_score = sum((y-y_oob)*(y-y_oob))/len(y)
oob_score
```

Out[17]: 14.517366425703933

In [18]:
```
y_oob=[]
for i in range(len(x)): #looping boston whole dataset
  y_oob_internal=[] #temporary arrays to hold oob value
  for indx,j in enumerate(list_selected_row): # looping for all trees
    if i not in j:   # calculating prediction value if ith point not in select
ed records
      # print("i-->",i,"indx-->",indx,"reg_tree_{}".format(indx),"j-->",sorted
(j))
      y_oob_internal.append(globals()['reg_tree_{}'.format(indx+1)].predict(x[
i][list_selected_columns[indx]].reshape(1,-1))[0])
      # print("cnt-->",cnt,"y_oob_internal-->",y_oob_internal)
  # print(mean(y_oob_internal))
  y_oob.append(mean(y_oob_internal))
      # print(x[list_selected_row[indx+1]][:,list_selected_columns[indx+1]])
# print(y_oob)
```

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 **

In [31]:
```
n=35
```

In [19]:
```
def task1(x,y):
  # 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,lis
t_of_all_models =[],[],[],[],[]

  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)

  # print("length of list_input_Data -->",len(list_input_data))

  for i in range(1,31):
    # print("i value-->",i)
    globals()['reg_tree_task_{}'.format(i)]=DecisionTreeRegressor(max_depth=No
ne)
    globals()['reg_tree_task_{}'.format(i)].fit(list_input_data[i-1],list_outp
ut_data[i-1])
    list_of_all_models.append(globals()['reg_tree_task_{}'.format(i)])

  array_of_Y=[]
  for i in range(1,31):
    globals()['Y_{}'.format(i)]=globals()['reg_tree_task_{}'.format(i)].predic
t(list_input_data[i-1])
    array_of_Y.append(globals()['Y_{}'.format(i)])

  y_pred=np.median(array_of_Y,axis=0)
  mse=mean_squared_error(y,y_pred)


  y_oob=[]
  for i in range(len(x)):
    y_oob_internal=[]
    for indx,j in enumerate(list_selected_row):
      if i not in j:
        y_oob_internal.append(globals()['reg_tree_task_{}'.format(indx+1)].pre
dict(x[i][list_selected_columns[indx]].reshape(1,-1))[0]) # appending predicte
d value y_oob_internal array
    y_oob.append(mean(y_oob_internal)) # calculating mean value returned by ea
ch model and appending to main array.

  oob_score=sum((y-y_oob)*(y-y_oob))/len(y)

  return (mse,oob_score)
```

In [20]:
```
%time
mse_array,oob_score_array=[],[]
for i in range(0,35):
  mse,oob_score=task1(x,y)
  mse_array.append(mse)
  oob_score_array.append(oob_score)
print(mse_array)
print(oob_score_array)
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 5.48 µs
[87.8173666007905, 86.90562345230303, 87.4832108426176, 89.03901274367453, 8
9.18900156791092, 89.52661106170399, 88.10275678771728, 87.96654480898873, 8
8.97034981149785, 87.13909321316135, 88.93247776679841, 88.67744565217392, 8
7.90607213438734, 89.04961778107159, 88.21654327368628, 87.3445185290404, 87.
57602413141872, 88.27480880605326, 88.09342979249011, 88.02053398660519, 87.4
0042574315169, 88.20650120155173, 86.5413014657444, 90.11659972551602, 87.397
80773262214, 88.25711134102532, 87.3556295387605, 88.74949361679872, 89.20465
954068573, 87.67815286012296, 89.07599802371541, 86.66105732343996, 88.316470
97533116, 87.66139449390276, 87.85813090140535]
[20.167591960676784, 15.459772064747193, 14.501112513507993, 15.0500128532791
01, 17.1219712309001 9, 14.831523904088103, 15.587629092090319, 17.19131932210
5397, 15.722496026796163, 17.735577279606975, 17.296723179028277, 12.98968923
4379506, 13.408635147917456, 13.42511107993492, 13.075106242462333, 15.829532
446102302, 14.353272334480803, 13.376274009302264, 14.083359410166432, 13.547
105115093883, 17.700948463264684, 15.268203256784696, 12.794921266468288, 16.
091015576924143, 16.10958849231399, 13.537399385983168, 12.192548943194907, 1
5.496652968989952, 15.355819925714787, 15.248678203530853, 14.58328525768864
2, 14.8127948201059, 13.492705209486502, 13.446766543897292, 13.6942020638383
02]
```

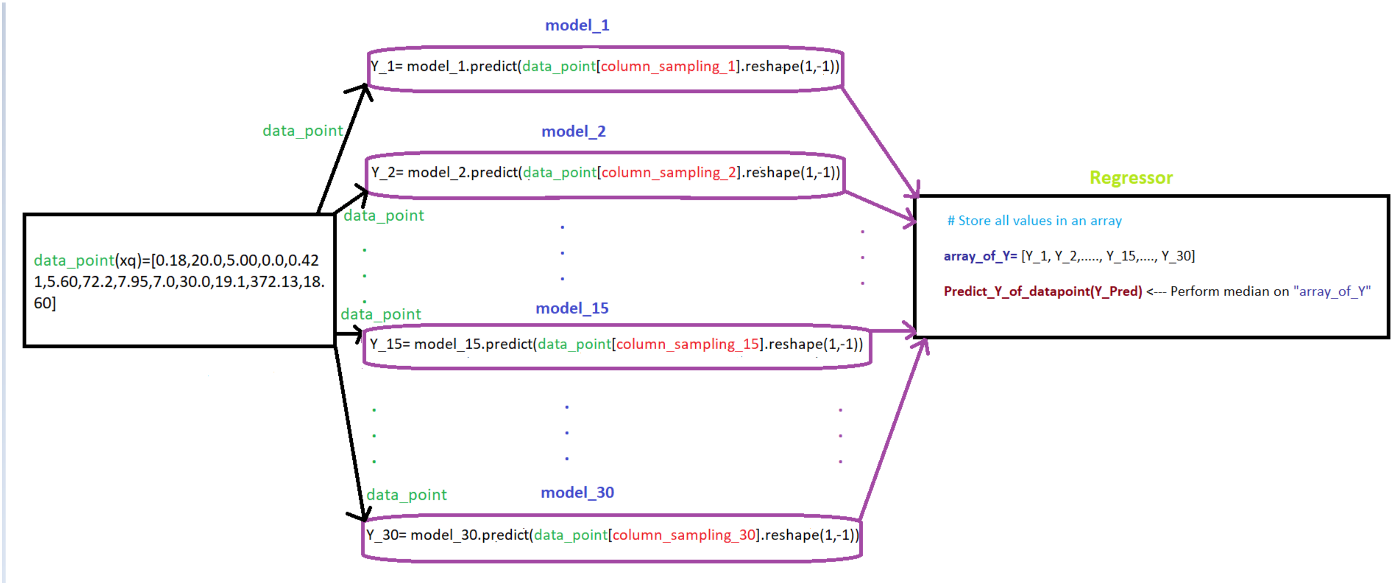
In [26]:
```
from statistics import stdev
mean_mse = mean(mse_array)
stddev_mse = stdev(mse_array)
mean_oob = mean(oob_score_array)
stddev_oob = stdev(oob_score_array)
print("Confidence interval for MSE is:{} {}".format(mean_mse-(2*stddev_mse),me
an_mse+(2*stddev_mse)))
print("Confidence interval for OOB is: {} {} ".format(mean_oob-(2*stddev_oob),
mean_oob+(2*stddev_oob)))
```

```
Confidence interval for MSE is:86.46419305892697 89.80505135409389
Confidence interval for OOB is: 11.53614509803822 18.439817463381928
```

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 [27]:
```
xq=np.array([0.18,20.0,5.00,0.0,0.421,5.60,72.2,7.95,7.0,30.0,19.1,372.13,18.60])
xq.reshape(1,-1)
```

Out[27]:
```
array([[1.8000e-01, 2.0000e+01, 5.0000e+00, 0.0000e+00, 4.2100e-01,
        5.6000e+00, 7.2200e+01, 7.9500e+00, 7.0000e+00, 3.0000e+01,
        1.9100e+01, 3.7213e+02, 1.8600e+01]])
```

In [32]:
```
xq=np.array([0.18,20.0,5.00,0.0,0.421,5.60,72.2,7.95,7.0,30.0,19.1,372.13,18.60])
yq=[]
for i in range(1,31):
  globals()['yq_{}'.format(i)]=globals()['reg_tree_{}'.format(i)].predict(xq[list_selected_columns[i-1]].reshape(1,-1))
  yq.append(globals()['yq_{}'.format(i)])
np.median(yq)
```

Out[32]: 18.5

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

In [ ]:

In [ ]: