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 [113]:

```
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 [114]:

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

In [115]:

```
x.shape
```

Out[115]:

```
(506, 13)
```

In [116]:

```
x[:5]
```

Out[116]:

```
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
```

In [117]:

```
import random
from tqdm import tqdm
from sklearn.tree import DecisionTreeRegressor
```

- **Write code for generating samples**

In [118]:

```
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/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

    Selecting_rows = np.random.choice(len(input_data),size=303,replace=False)
    Replacing_rows = np.random.choice(Selecting_rows,size=203,replace=True)
    no_of_columns = random.randint(3,13)
    Selecting_columns = np.random.choice(len(input_data[0]),size=no_of_columns,replace=Fals
    sample_data = input_data[Selecting_rows[:,None],Selecting_columns]
    target_of_sample_data = target_data[Selecting_rows]

    #total_rows = np.concatenate((Selecting_rows,Replacing_rows))

    replicated_sample_data = input_data[Replacing_rows[:,None],Selecting_columns]
    target_of_replicated_sample_data = target_data[Replacing_rows]

    final_sampled_data = np.vstack((sample_data,replicated_sample_data))
    final_target_data = np.vstack((target_of_sample_data.reshape(-1,1),target_of_replicatec
    return final_sampled_data, final_target_data, Selecting_rows, Selecting_columns
```

In [ ]:

```

```

Grader function - 1

In [119]:

```
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[119]:

```
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 [120]:

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

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

```
100%|██████████████████████████████████████████████████
█████████| 30/30 [00:00<00:00, 2725.64it/s]
```

In [121]:

```
print("List Input data has",len(list_input_data)," samples")
print("Each sample has", len(list_input_data[23])," rows")
```

```
List Input data has 30  samples
Each sample has 506  rows
```

In [122]:

```
list_input_data[23]
```

Out[122]:

```
array([[  0.    ,   1.7455,  14.7   , ...,   6.152 ,   1.    ,  88.01  ],
       [ 22.    ,   7.3967,  19.1   , ...,   6.487 ,   0.    , 396.28  ],
       [  0.    ,   4.3549,  18.6   , ...,   6.326 ,   0.    , 394.87  ],
       ...,
       [  0.    ,   2.4259,  14.7   , ...,   5.877 ,   0.    , 227.61  ],
       [ 22.    ,   7.9549,  19.1   , ...,   5.593 ,   0.    , 372.49  ],
       [ 20.    ,   1.801 ,  13.    , ...,   8.704 ,   0.    , 389.7   ]])
```

In [123]:

```
print("List output data has",len(list_output_data)," samples")
print("Each sample has", len(list_output_data[23])," rows")
```

```
List output data has 30  samples
Each sample has 506  rows
```

In [124]:

```
print("List selected rows has",len(list_selected_rows)," samples")
print("Each sample has", len(list_selected_rows[23])," indices of rows")
```

```
List selected rows has 30  samples
Each sample has 303  indices of rows
```

In [125]:

```
print("List selected columns has",len(list_selected_columns)," samples")
```

```
List selected columns has 30  samples
```

In [126]:

```
print(len(list_selected_columns[23]))
print(len(list_selected_columns[21]))
#of all 30 samples, each sample has different column indices
```

```
8
11
```

List input data has variable columns/features for each sample, whereas list output data will have only one feature

Grader function - 2

In [127]:

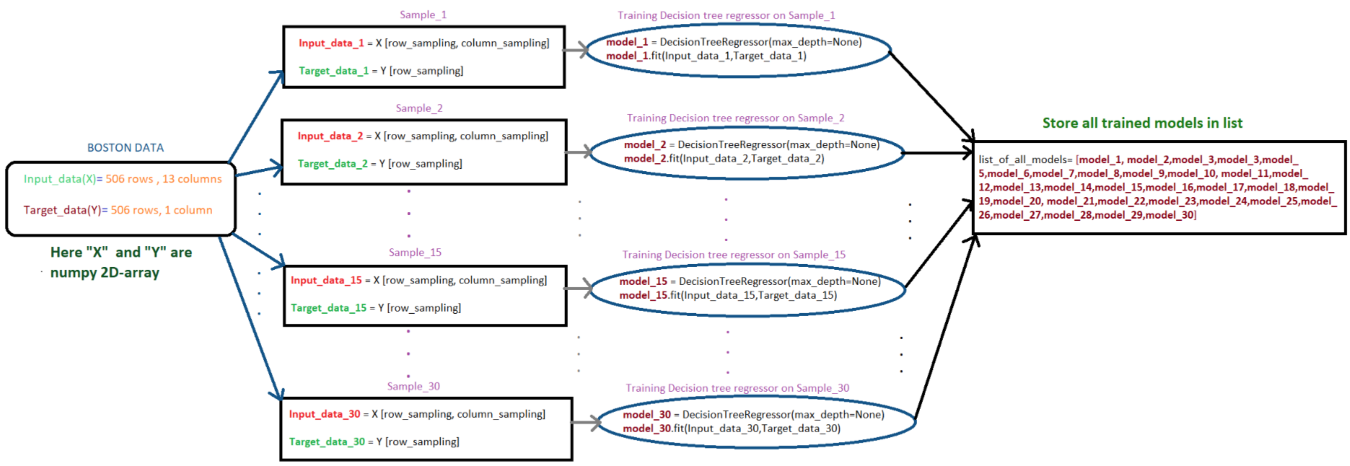
```
def grader_30(a):
    assert(len(a)==30 and len(a[0])==506)
    return True
grader_30(list_input_data)
```

Out[127]:

```
True
```

Step - 2

Flowchart for building tree



- **Write code for building regression trees**

In [128]:

```
models=[]
for i in range(0,30):
    model=DecisionTreeRegressor()
    model.fit(list_input_data[i],list_output_data[i])
    models.append(model)
```

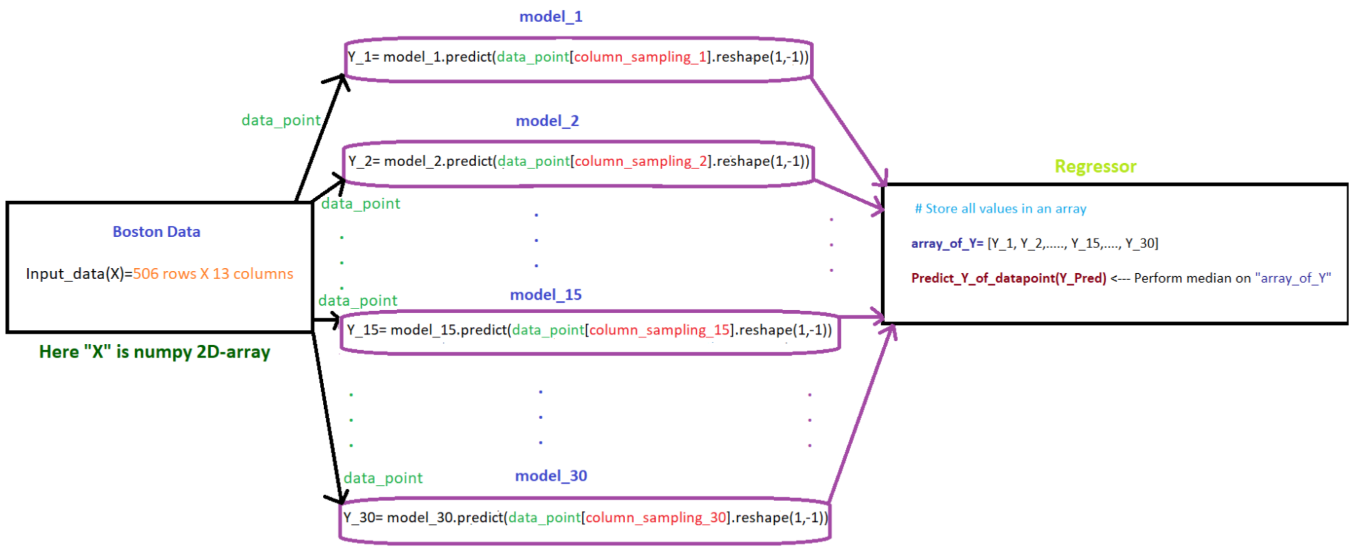
In [129]:

```
models[0]
```

Out[129]:

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=None, splitter='best')
```

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.

For calculating MSE,We have to calculate y_pred values

- 1) Y_pred: We have to calculate for each datapoint from 30 samples and then taking median of it
- 2) We will be having 506 medians of y_pred
- 3) Calculating MSE with 506 y_true values and 506 y_pred medians

In [130]:

```
list_selected_columns[0]
```

Out[130]:

```
array([ 4,  8,  0,  7,  3,  5, 11, 12,  2,  9, 10,  6,  1])
```

In [131]:

```
x[0,list_selected_columns[0]].reshape(-1,1)
```

Out[131]:

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

In [132]:

```
from tqdm import tqdm
y_pred=[]
for i in tqdm(range(0,len(x))):
    y_pred_each_datapoint=[]
    for j in range(0,len(list_selected_columns)):
        y_pred_value = models[j].predict(x[i,list_selected_columns[j]].reshape(1,-1))
        y_pred_each_datapoint.append(y_pred_value)
    y_pred.append(np.median(y_pred_each_datapoint))
```

```
100%|██████████████████████████████████████████████████
█████| 506/506 [00:00<00:00, 555.08it/s]
```

In [133]:

```
MSE = mean_squared_error(y,y_pred)
```

In [134]:

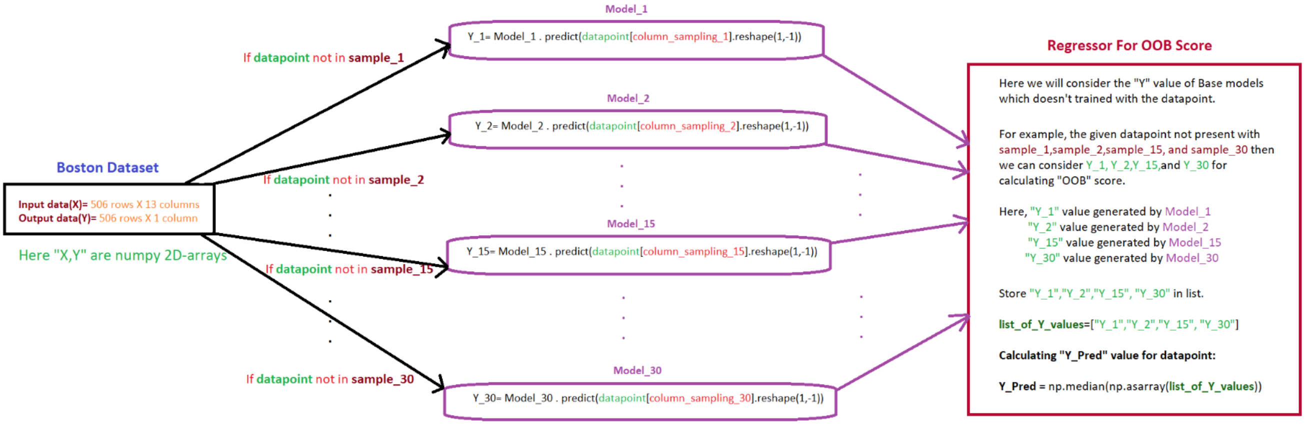
```
MSE
```

Out[134]:

```
0.024174901185770727
```

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 [135]:

```
y_pred=[]
for i in tqdm(range(0,len(x))):
    y_pred_each_datapoint=[]
    for j in range(0,len(list_selected_columns)):
        if i not in list_selected_rows[j]:
            y_pred_value = models[j].predict(x[i,list_selected_columns[j]].reshape(1,-1))
            y_pred_each_datapoint.append(y_pred_value)
    y_pred.append(np.median(y_pred_each_datapoint))
```

```
100%|████████████████████████████████████████████████████
████| 506/506 [00:00<00:00, 964.96it/s]
```

In [136]:

```
y_pred
```

Out[136]:

```
[29.4,
 23.4,
 34.05,
 36.2,
 32.95,
 24.2,
 20.200000000000003,
 18.9,
 15.0,
 19.9,
 19.55,
 20.4,
 21.2,
 19.9,
 19.799999999999997,
 20.4,
 20.7,
 18.65,
```

In [137]:

```
s=0
for i in range(0,len(y_pred)):
    s = s + (y[i] - y_pred[i])**2
OOB_Score = s/len(y)
```

In [138]:

```
OOB_Score
```

Out[138]:

```
15.516999615200303
```

Task 2

In [139]:

```
mean_sqr_error_array=[]
oob_score_array=[]
for k in tqdm(range(0,35)):
    list_input_data =[]
    list_output_data =[]
    list_selected_rows= []
    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_rows.append(c)
        list_selected_columns.append(d)

    models=[]
    for i in range(0,30):
        model=DecisionTreeRegressor()
        model.fit(list_input_data[i],list_output_data[i])
        models.append(model)

    '''y_pred_mse=[]
    y_pred_oob=[]
    for i in range(0,len(x)):
        y_pred_each_datapoint_mse=[]
        y_pred_each_datapoint_oob=[]
        for j in range(0,len(list_selected_columns)):
            if i not in list_selected_rows[j]:
                y_pred_value_0 = models[j].predict(x[i,list_selected_columns[j]].reshape(1,
                y_pred_each_datapoint_oob.append(y_pred_value_0)
            y_pred_value_1 = models[j].predict(x[i,list_selected_columns[j]].reshape(1,-1))
            y_pred_each_datapoint_mse.append(y_pred_value_1)
        y_pred_mse.append(np.median(y_pred_each_datapoint_mse))
        y_pred_oob.append(np.median(y_pred_each_datapoint_oob))'''

    y_pred=[]
    for i in range(0,len(x)):
        y_pred_each_datapoint=[]
        for j in range(0,len(list_selected_columns)):
            y_pred_value = models[j].predict(x[i,list_selected_columns[j]].reshape(1,-1))
            y_pred_each_datapoint.append(y_pred_value)
        y_pred.append(np.median(y_pred_each_datapoint))
    mean_sqr_error_array.append(mean_squared_error(y,y_pred))

    y_pred=[]
    for i in range(0,len(x)):
        y_pred_each_datapoint=[]
        for j in range(0,len(list_selected_columns)):
            if i not in list_selected_rows[j]:
                y_pred_value = models[j].predict(x[i,list_selected_columns[j]].reshape(1,-1
                y_pred_each_datapoint.append(y_pred_value)
        y_pred.append(np.median(y_pred_each_datapoint))

    s=0
    for i in range(0,len(y_pred)):
        s = s + (y[i] - y_pred[i])**2
    OOB_Score = s/len(y)
    oob_score_array.append(OOB_Score)
```

```
100%|██████████████████████████████████████████████████████████████
██████████| 35/35 [00:49<00:00,  1.40s/it]
```

In [140]:

```python
print(mean_sqr_error_array)
print(len(mean_sqr_error_array))
```

```
[0.09864943232233607, 0.29553133324660275, 0.11534906382920525, 0.1158992094
8616601, 0.14368577075098812, 0.050444664031620554, 0.01942687747035572, 0.0
7606719367588931, 0.051749011857707534, 0.029421936758893256, 0.043324648671
943175, 0.0194122529644268 8, 0.025803736506232498, 0.04726615886786048, 0.02
9279891304347813, 0.18725723181371726, 0.06531126482213429, 0.10931324110671
942, 0.012712450592885397, 0.06657785770750997, 0.011403162055335989, 0.0021
64580588493633, 0.1866156126482214, 0.0901698727514722, 0.010587944664031 61
8, 0.09653656126482199, 0.023641304347826082, 0.048325098814229285, 0.251706
25318675244, 0.19949229249011857, 0.0420778779644269, 0.22357718059437456,
1.196205533596838, 0.09748414394663178, 0.018948863636363642]
35
```

In [141]:

```python
print(oob_score_array)
print(len(oob_score_array))
```

```
[15.74372104480223, 14.839352365292784, 15.807805362906365, 11.9611511857707
47, 14.989397233201576, 14.982267786561263, 16.127849692577946, 15.988674276
03453, 13.934240387571375, 11.89580533596838, 15.456272052858129, 15.0887955
259424, 13.070513108397279, 15.005340929753334, 11.495507728124194, 16.56999
8305459006, 12.01405935742989, 12.58307312252964, 11.740578063241108, 12.706
145042074972, 15.657580423803244, 13.709472716293375, 12.773932697637962, 1
6.204163102323115, 14.19397727272728, 13.788469166293222, 14.18857332015809
9, 15.29028107158543, 14.641134995654516, 15.21194387439613, 14.836384157980
273, 14.538160707767537, 15.861294865700312, 13.55331489260839, 11.850980491
05182]
35
```

Calculating CI

In [142]:

```python
from prettytable import PrettyTable
mean_sqr_error_array = np.array(mean_sqr_error_array)
oob_score_array = np.array(oob_score_array)
```

In [143]:

```
X = PrettyTable()
X = PrettyTable(["#samples", "Sample Size", "Sample mean", "Pop Std","Left C.I","Right C.I"
population_std = mean_sqr_error_array.std()
population_mean= np.round(mean_sqr_error_array.mean(), 3)
for i in range(10):
    sample=mean_sqr_error_array[random.sample(range(0, mean_sqr_error_array.shape[0]), 15)]
    sample_mean = sample.mean()
    sample_size = len(sample)
    left_limit  = np.round(sample_mean - 2*(population_std/np.sqrt(sample_size)), 3)
    right_limit = np.round(sample_mean + 2*(population_std/np.sqrt(sample_size)), 3)
    row = []
    row.append(i+1)
    row.append(sample_size)
    row.append(sample_mean)
    row.append(population_std)
    row.append(left_limit)
    row.append(right_limit)
    row.append(population_mean)
    row.append((population_mean <= right_limit) and (population_mean >= left_limit))
    X.add_row(row)
print(X)
```

```
+----------+-------------+---------------------+--------------------+-------
---+-----------+---------+-------+
| #samples | Sample Size |     Sample mean     |      Pop Std       | Left
C.I | Right C.I | Pop mean | Catch |
+----------+-------------+---------------------+--------------------+-------
---+-----------+---------+-------+
|    1     |      15     | 0.15009912803692496 | 0.1992177529479918 |  0.047
|   0.253   |  0.117   |  True |
|    2     |      15     | 0.16358029687661027 | 0.1992177529479918 |  0.061
|   0.266   |  0.117   |  True |
|    3     |      15     | 0.11805821161502773 | 0.1992177529479918 |  0.015
|   0.221   |  0.117   |  True |
|    4     |      15     | 0.08467550555207556 | 0.1992177529479918 |  -0.01
8  |   0.188   |  0.117   |  True |
|    5     |      15     | 0.15058315943313746 | 0.1992177529479918 |  0.048
|   0.253   |  0.117   |  True |
|    6     |      15     | 0.15047279467713223 | 0.1992177529479918 |  0.048
|   0.253   |  0.117   |  True |
|    7     |      15     |  0.167502014234919  | 0.1992177529479918 |  0.065
|    0.27   |  0.117   |  True |
|    8     |      15     | 0.07032487652897056 | 0.1992177529479918 |  -0.03
3  |   0.173   |  0.117   |  True |
|    9     |      15     |  0.1763242335189746 | 0.1992177529479918 |  0.073
|   0.279   |  0.117   |  True |
|    10    |      15     | 0.17310183226602824 | 0.1992177529479918 |   0.07
|   0.276   |  0.117   |  True |
+----------+-------------+---------------------+--------------------+-------
---+-----------+---------+-------+
```

In [144]:

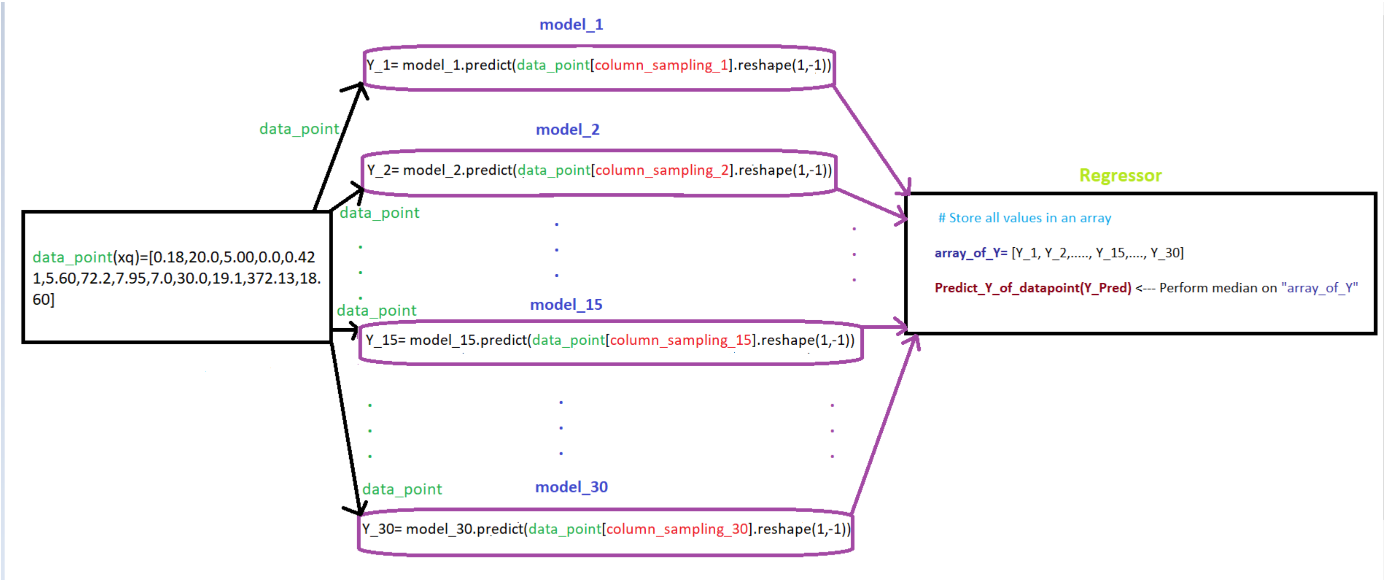
```
X = PrettyTable()
X = PrettyTable(["#samples", "Sample Size", "Sample mean", "Pop Std","Left C.I","Right C.I"
population_std = oob_score_array.std()
population_mean= np.round(oob_score_array.mean(), 3)
for i in range(10):
    sample=oob_score_array[random.sample(range(0, oob_score_array.shape[0]), 15)]
    sample_mean = sample.mean()
    sample_size = len(sample)
    left_limit  = np.round(sample_mean - 2*(population_std/np.sqrt(sample_size)), 3)
    right_limit = np.round(sample_mean + 2*(population_std/np.sqrt(sample_size)), 3)
    row = []
    row.append(i+1)
    row.append(sample_size)
    row.append(sample_mean)
    row.append(population_std)
    row.append(left_limit)
    row.append(right_limit)
    row.append(population_mean)
    row.append((population_mean <= right_limit) and (population_mean >= left_limit))
    X.add_row(row)
print(X)
```

```
+----------+------------+-------------------+--------------------+--------
--+-----------+---------+-------+
| #samples | Sample Size |    Sample mean     |      Pop Std       | Left C.
I | Right C.I | Pop mean | Catch |
+----------+------------+-------------------+--------------------+--------
--+-----------+---------+-------+
|    1     |     15     | 14.287220966821979 | 1.4830135773034538 |  13.521
|  15.053  |  14.237   |  True |
|    2     |     15     | 14.730486737712642 | 1.4830135773034538 |  13.965
|  15.496  |  14.237   |  True |
|    3     |     15     | 13.463618362202974 | 1.4830135773034538 |  12.698
|  14.229  |  14.237   | False |
|    4     |     15     | 14.382455018028311 | 1.4830135773034538 |  13.617
|  15.148  |  14.237   |  True |
|    5     |     15     | 13.740151181190852 | 1.4830135773034538 |  12.974
|  14.506  |  14.237   |  True |
|    6     |     15     | 14.295700627462116 | 1.4830135773034538 |  13.53
|  15.062  |  14.237   |  True |
|    7     |     15     | 13.821035617461783 | 1.4830135773034538 |  13.055
|  14.587  |  14.237   |  True |
|    8     |     15     | 14.143833152918443 | 1.4830135773034538 |  13.378
|  14.91   |  14.237   |  True |
|    9     |     15     | 14.838824043471744 | 1.4830135773034538 |  14.073
|  15.605  |  14.237   |  True |
|   10     |     15     | 14.276190686990237 | 1.4830135773034538 |  13.51
|  15.042  |  14.237   |  True |
+----------+------------+-------------------+--------------------+--------
--+-----------+---------+-------+
```

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 [145]:

```
list_input_data =[]
list_output_data =[]
list_selected_rows= []
list_selected_columns=[]

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

```
100%|██████████████████████████████████████████████
█████| 30/30 [00:00<00:00, 3471.82it/s]
```

In [146]:

```
models=[]
for i in range(0,30):
    model=DecisionTreeRegressor()
    model.fit(list_input_data[i],list_output_data[i])
    models.append(model)
```

In [147]:

```
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])
```

In [151]:

```
xq[[0,3]]
```

Out[151]:

```
array([0.18, 0.  ])
```

In [155]:

```
y_pred_each_datapoint=[]
for i in range(0,len(list_selected_columns)):
    y_pred_value = models[i].predict(xq[list_selected_columns[i]].reshape(1,-1))
    y_pred_each_datapoint.append(y_pred_value)
```

In [156]:

```
y_pred_each_datapoint
```

Out[156]:

```
[array([18.5]),
 array([19.4]),
 array([46.]),
 array([22.5]),
 array([16.6]),
 array([17.4]),
 array([19.7]),
 array([22.5]),
 array([18.5]),
 array([27.1]),
 array([19.4]),
 array([18.5]),
 array([17.8]),
 array([19.5]),
 array([50.]),
 array([17.5]),
 array([18.5]),
 array([18.9]),
 array([17.6]),
 array([18.5]),
 array([18.5]),
 array([18.5]),
 array([18.5]),
 array([18.5]),
 array([19.4]),
 array([20.5]),
 array([18.5]),
 array([18.5]),
 array([22.5]),
 array([17.6])]
```

In [158]:

```
print("Predicted House Price for the given data point is {0}".format(np.median(y_pred_each_
```

```
Predicted House Price for the given data point is 18.5
```

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

For Task 1, very low MSE Score, so there is high variance between the features

For Task 2, MSE & OOB mean lie within CI

For Task 3, y value is predicted

In [ ]: