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 [60]:
```
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 [61]:
```
boston = load_boston()
x=boston.data #independent variables
y=boston.target #target variable
```

In [62]:
```
print(x.shape)
print(y.shape)
```

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

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 [63]:

```python
# '''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
    # 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(input_data, target_data):

    Selecting_rows = np.random.choice(506 ,303 ,replace=False)
    Selecting_columns = np.random.choice([0,1,2,3,4,5,6,7,8,9,10,11,12], np.random.choi
    sample_data = input_data[ Selecting_rows[:,None] , Selecting_columns ]
    target_of_sample_data = target_data[Selecting_rows]

    #Replicating data
    Replacing_rows = np.random.choice(303, 203,replace=False)
    Replicated_sample_data = sample_data[Replacing_rows]
    target_of_Replicated_sample_data = target_of_sample_data[Replacing_rows]

    #Concatenation
    final_sample_data = np.vstack((sample_data,Replicated_sample_data))
    final_target_data = np.vstack((target_of_sample_data.reshape(-1,1),target_of_Replic

    return final_sample_data,final_target_data,Selecting_rows,Selecting_columns
```

Grader function - 1 </fongt>

In [64]:

```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[64]: 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 [65]:

```
# 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

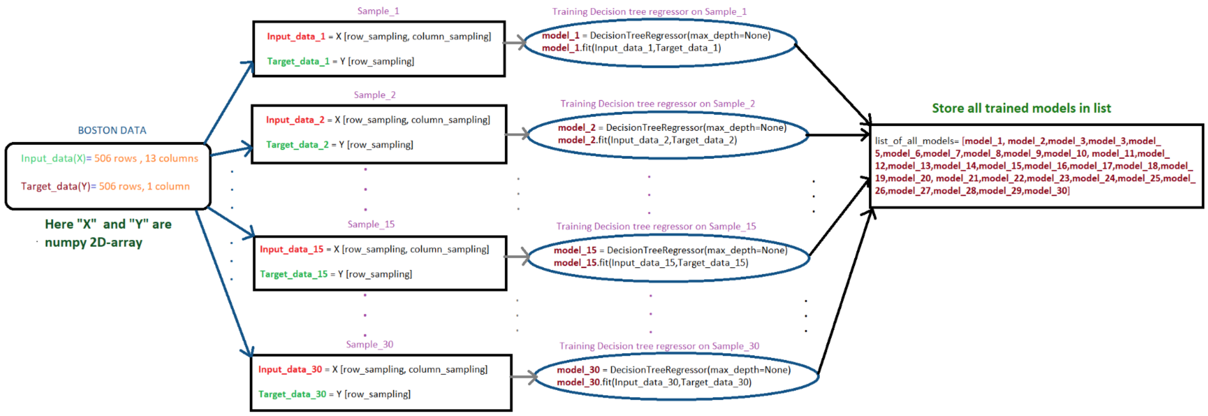
In [66]:

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

Out[66]: True

Step - 2

Flowchart for building tree

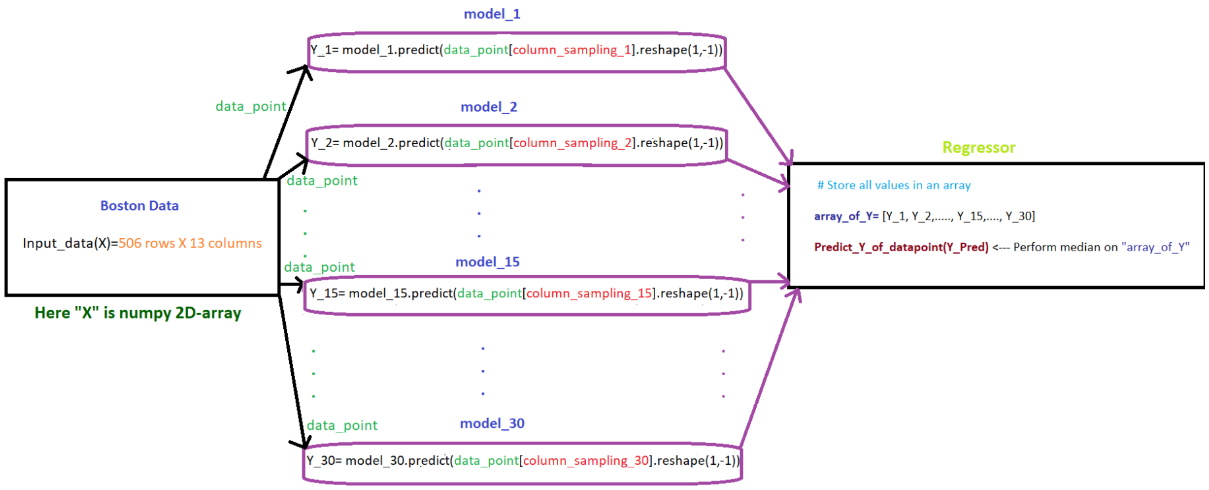


- **Write code for building regression trees**

In [67]:
```
from sklearn.tree import DecisionTreeRegressor

list_of_all_models = []

for i in range(len(list_input_data)):
    dt = DecisionTreeRegressor(max_depth = None)

    trained_dt = dt.fit(list_input_data[i],list_output_data[i])

    list_of_all_models.append(trained_dt)

print(list_of_all_models)    #storing all decision trees for each set of input and outp
```

```
[DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTree
Regressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeRegressor(),
DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeR
egressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), D
ecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeRe
gressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), De
cisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeReg
ressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), DecisionTreeRegressor(), Dec
isionTreeRegressor(), DecisionTreeRegressor()]
```

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 [68]:
```
array_of_Y = []

for k in range(len(list_of_all_models)):

    y_predicted = list_of_all_models[k].predict( x[:,list_selected_columns[k]]  )

    array_of_Y.append(y_predicted)

array_of_Y = np.array(array_of_Y)   #array of predicted Y all data points from x

predicted_y = []

for j in range(506):

    sorted_array_of_Y = np.sort(array_of_Y[:,j])

    median =  np.median(sorted_array_of_Y)

    predicted_y.append(median)   #median of prediction of each data point from all the

MSE = mean_squared_error(y, predicted_y )

print(MSE)
```

```
0.039582509881422916
```

Step - 3

Flowchart for calculating OOB score

alt text

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

- **Write code for calculating OOB score**

In [69]:
```
final_model_list = []    #for ith data point not in which samples/models

for i in range(len(x)):
    temp = []
    for j in range(len(list_selected_row)):
        if (i not in list_selected_row[j]):  #for each data point , storing the models
            temp.append(j)
    final_model_list.append(temp)

print(final_model_list[:5])   #length of 506
```

```
[[9, 11, 13, 14, 17, 22, 24, 26], [5, 7, 8, 9, 10, 11, 12, 14, 15, 16, 18, 19, 21, 23, 2
5, 26, 28, 29], [0, 1, 3, 4, 5, 6, 8, 9, 11, 13, 17, 21, 22, 23, 24, 26, 28], [4, 5, 6,
9, 11, 12, 14, 15, 17, 19, 24], [5, 8, 9, 16, 19, 21, 28]]
```

In [70]:
```
#array_of_Y

predicted_y = []

for j in range(array_of_Y.shape[1]):


    sorted_array_of_Y = np.sort(array_of_Y[final_model_list[j],j])  #from array of all

    median = np.median(sorted_array_of_Y)

    predicted_y.append(median)


OOB = mean_squared_error(y , predicted_y )

print(OOB)
```

```
16.96931710673051
```

Task 2

In [71]:
```
MSE_35 = []      #runinng above code for 35 times and storing MSE of each run
OOB_35 = []      #runinng above code for 35 times and storing OOB of each run

for z in range(35):

    def generating_samples(input_data, target_data):

        Selecting_rows = np.random.choice(506 ,303 ,replace=False)
        Selecting_columns = np.random.choice([0,1,2,3,4,5,6,7,8,9,10,11,12], np.random.
        sample_data = input_data[ Selecting_rows[:,None] , Selecting_columns ]
        target_of_sample_data = target_data[Selecting_rows]

        #Replicating data
        Replacing_rows = np.random.choice(303, 203)
        Replicated_sample_data = sample_data[Replacing_rows]
        target_of_Replicated_sample_data = target_of_sample_data[Replacing_rows]

        #Concatenation
        final_sample_data = np.vstack((sample_data,Replicated_sample_data))
        final_target_data = np.vstack((target_of_sample_data.reshape(-1,1),target_of_Re

        return final_sample_data,final_target_data,Selecting_rows,Selecting_columns
```

```
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)

from sklearn.tree import DecisionTreeRegressor

list_of_all_models = []

for i in range(len(list_input_data)):
    dt = DecisionTreeRegressor(max_depth = None)
    trained_dt = dt.fit(list_input_data[i],list_output_data[i])
    list_of_all_models.append(trained_dt)

from sklearn.metrics import mean_squared_error

array_of_Y = []

for k in range(len(list_of_all_models)):

    y_predicted = list_of_all_models[k].predict(x[:,list_selected_columns[k]])

    array_of_Y.append(y_predicted)

array_of_Y = np.array(array_of_Y)

predicted_y = []

for j in range(array_of_Y.shape[1]):

    sorted_array_of_Y = np.sort(array_of_Y[:,j])

    median = np.median(sorted_array_of_Y)

    predicted_y.append(median)


MSE_35.append(mean_squared_error(y , predicted_y ))

predicted_y = []

final_model_list = []    #for ith data point not in which samples/models

for i in range(len(x)):
    temp = []
    for j in range(len(list_selected_row)):
        if (i not in list_selected_row[j]):  #for each data point , storing the mod
            temp.append(j)
    final_model_list.append(temp)



for j in range(array_of_Y.shape[1]):
```

```
        sorted_array_of_Y = np.sort(array_of_Y[final_model_list[j],j])  #from array of

        median = np.median(sorted_array_of_Y)

        predicted_y.append(median)


    OOB_35.append(mean_squared_error(y , predicted_y ))

print("35 MSE Scores : \n" , MSE_35)
print("\n35 OOB Scores : \n" ,OOB_35)
```

```
35 MSE Scores :
 [0.07703063241106725, 0.18720928853754942, 0.2419960474308301, 0.09516916996047439, 0.0
9120772588442816, 0.11063848338436015, 0.0395009881422925, 0.05561264822134389, 0.126358
69565217392, 0.04510753767292489, 0.029150197628458527, 0.04700716403162053, 0.063843873
51778661, 0.3925351310352865, 0.009240207234540597, 0.10354592116820392, 0.0615025252525
2522, 0.04281620553359686, 0.14819664031620558, 0.05352041126512326, 0.0656867588932806
8, 0.01657608695652172, 0.19062757118657733, 0.03340909090909095, 0.07006916996047435,
0.011002964426877478, 0.09020850351339488, 0.04695369512307116, 0.05483973567193676, 0.2
4840415019762857, 0.056849061264822136, 0.025879446640316214, 0.12128318124749328, 0.185
46566205533577, 0.06975516029863857]

35 OOB Scores :
 [14.53287453337725, 15.017660928831349, 13.436488526570047, 14.107198686622214, 16.0173
3341809545, 12.601590858187995, 14.549772727272726, 10.7008998270751, 13.38334761883424,
15.176447724582784, 14.145968379446641, 15.950983750548968, 11.490838842672263, 13.14057
5440767572, 12.719290854307983, 12.930442509737203, 12.25596643256698, 13.8651185770751,
11.290968379446639, 13.484062796031196, 14.56314723320158, 11.327782855731224, 14.295874
157912767, 15.249782632166097, 11.303486632630653, 14.666441178085197, 12.2039064045291
4, 13.143070423765382, 15.10691854001976, 13.172223274936549, 15.174440009826768, 15.240
0395256917, 13.202148960835446, 14.20138666007905, 12.230044045129581]
```

In [72]:

```
#Confidence Interval using the python notebook on central limit theorem

s_MSE_std = np.std(np.array(MSE_35))
x_MSE_mean = np.round(np.mean(np.array(MSE_35)), 3)
size_MSE = len(MSE_35)

left_limit  = np.round(x_MSE_mean - 2*(s_MSE_std/np.sqrt(size_MSE)), 3)
right_limit  = np.round(x_MSE_mean + 2*(s_MSE_std/np.sqrt(size_MSE)), 3)

print("95% CI of MSE : " , [left_limit,right_limit])


s_OOB_std = np.std(np.array(OOB_35))
x_OOB_mean = np.round(np.mean(np.array(OOB_35)), 3)
size_OOB = len(OOB_35)

left_limit  = np.round(x_OOB_mean - 2*(s_OOB_std/np.sqrt(size_OOB)), 3)
right_limit  = np.round(x_OOB_mean + 2*(s_OOB_std/np.sqrt(size_OOB)), 3)

print("95% CI of OOB : " , [left_limit,right_limit])
```

```
95% CI of MSE :  [0.068, 0.122]
95% CI of OOB :  [13.124, 14.07]
```

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.

alt text

- **Write code for TASK 3**

In [77]:
```
def generating_samples(input_data, target_data):

    Selecting_rows = np.random.choice(506 ,303 ,replace=False)
    Selecting_columns = np.random.choice([0,1,2,3,4,5,6,7,8,9,10,11,12], np.random.choi
    sample_data = input_data[ Selecting_rows[:,None] , Selecting_columns ]
    target_of_sample_data = target_data[Selecting_rows]

    #Replicating data
    Replacing_rows = np.random.choice(303, 203)
    Replicated_sample_data = sample_data[Replacing_rows]
    target_of_Replicated_sample_data = target_of_sample_data[Replacing_rows]

    #Concatenation
    final_sample_data = np.vstack((sample_data,Replicated_sample_data))
    final_target_data = np.vstack((target_of_sample_data.reshape(-1,1),target_of_Replic

    return final_sample_data,final_target_data,Selecting_rows,Selecting_columns

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)

from sklearn.tree import DecisionTreeRegressor

list_of_all_models = []

for i in range(len(list_input_data)):
    dt = DecisionTreeRegressor(max_depth = None)
    trained_dt = dt.fit(list_input_data[i],list_output_data[i])
    list_of_all_models.append(trained_dt)

from sklearn.metrics import mean_squared_error

array_of_Y = []

for k in range(len(list_of_all_models)):

    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],]

    xq = np.array(xq)

    y_predicted = list_of_all_models[k].predict(  xq[:,list_selected_columns[k]] )

    array_of_Y.append(y_predicted)
```

```
array_of_Y = np.array(array_of_Y)

predicted_y = []

for j in range(array_of_Y.shape[1]):

    sorted_array_of_Y = np.sort(array_of_Y[:,j])

    median = np.median(sorted_array_of_Y)

    predicted_y.append(median)

print("Price of the house for given Xq: " , predicted_y)
```

```
Price of the house for given Xq:  [18.5]
```

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

Task 1 Observations :

```
MSE is 0.039582509881422916

OOB is 16.96931710673051
```

Task 2 Observations :

```
35 MSE Scores :
 [0.07703063241106725, 0.18720928853754942, 0.2419960474308301,
0.09516916996047439, 0.09120772588442816, 0.11063848338436015,
0.0395009881422925, 0.05561264822134389, 0.12635869565217392,
0.04510753767292489, 0.029150197628458527, 0.04700716403162053,
0.06384387351778661, 0.3925351310352865, 0.009240207234540597,
0.10354592116820392, 0.06150252525252522, 0.04281620553359686,
0.14819664031620558, 0.05352041126512326, 0.06568675889328068,
0.01657608695652172, 0.19062757118657733, 0.03340909090909095,
0.07006916996047435, 0.011002964426877478, 0.09020850351339488,
0.04695369512307116, 0.05483973567193676, 0.24840415019762857,
0.056849061264822136, 0.025879446640316214, 0.12128318124749328,
0.18546566205533577, 0.06975516029863857]

35 OOB Scores :
 [14.53287453337725, 15.017660928831349, 13.436488526570047,
14.107198686622214, 16.01733341809545, 12.601590858187995,
14.549772727272726, 10.7008998270751, 13.38334761883424,
15.176447724582784, 14.145968379446641, 15.950983750548968,
11.490838842672263, 13.140575440767572, 12.719290854307983,
12.930442509737203, 12.25596643256698, 13.8651185770751,
11.290968379446639, 13.484062796031196, 14.56314723320158,
11.327782855731224, 14.295874157912767, 15.249782632166097,
11.303486632630653, 14.666441178085197, 12.20390640452914,
```

```
13.143070423765382, 15.10691854001976, 13.172223274936549,
15.174440009826768, 15.2400395256917, 13.202148960835446,
14.20138666007905, 12.230044045129581]

95% CI of MSE :  [0.068, 0.122]

95% CI of OOB :  [13.124, 14.07]
```

Task 3 Observations :

```
Price of the house for given Xq:  [18.5]
```