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
from sklearn.tree import DecisionTreeRegressor
```

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,

- we will replicate 4 points from [4, 5, 7, 8, 9, 3], consider 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 [21]:

```
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 more details
    # 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(input_data.shape[0], round(0.6 * input_data.shape[0]) - 1, replace = False)
    replacing_rows = np.random.choice(selecting_rows, round(0.4 * input_data.shape[0]) + 1, replace = False)
    selecting_columns = np.random.choice(13, 3, replace = False)

    sample_data = input_data[selecting_rows[:, None], selecting_columns]
    target_of_sample_data = target_data[selecting_rows]

    replicated_sample_data = input_data[replacing_rows[:, None], selecting_columns]
    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_columns
```

Grader function - 1 </fongt>

In [22]:

```
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[22]:

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 [23]:

```
# 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(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 [24]:

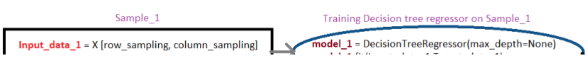
```
def grader_30(a):
    assert(len(a)==30 and len(a[0])==506)
    return True
grader_30(list_input_data)
```

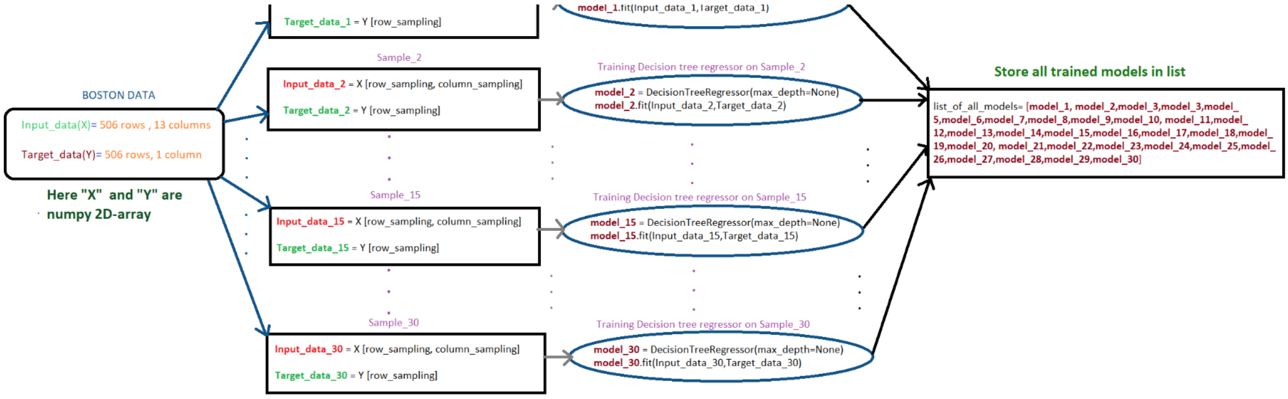
Out[24]:

True

Step - 2

Flowchart for building tree





- **Write code for building regression trees**

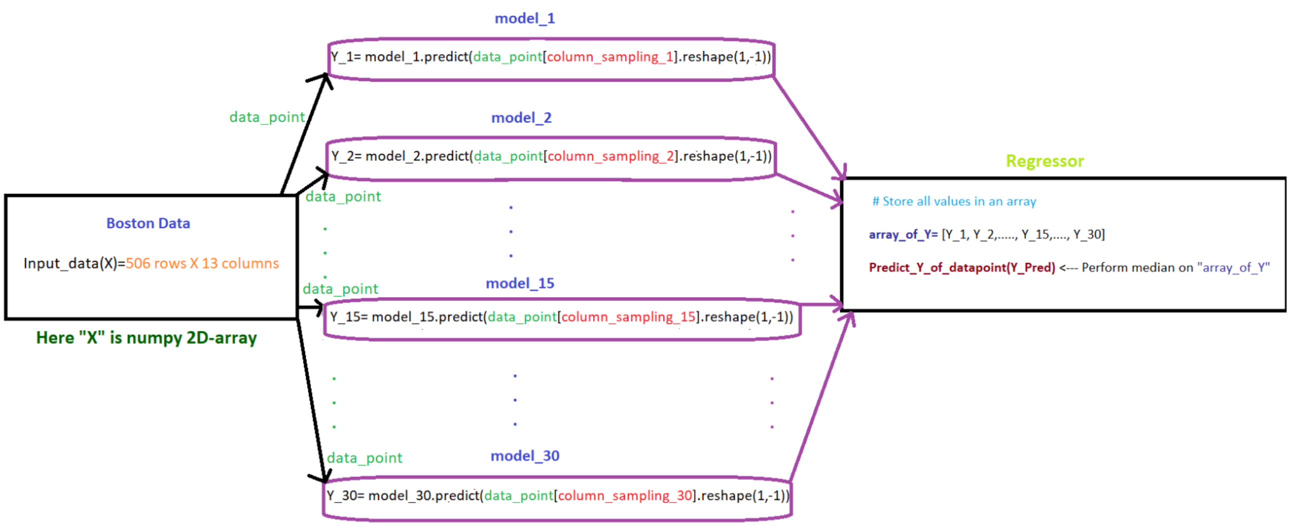
In [25]:

```
def buildDecisionTree(inputX, inputY):

    """This function creates and fits decision tree regressor with input train X and Y"""
    tree = DecisionTreeRegressor(max_depth = None)
    tree.fit(inputX, inputY)

    return tree
```

In [27]:

```
# List 30 decision trees
decisionTrees = []

for i in range(len(list_input_data)):
    decisionTrees.append(buildDecisionTree(list_input_data[i], list_output_data[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 [28]:

```
def predict_Y_of_datapoint(y_pred):

    """This function gives the median of all predicted Y values for each point"""

    # Take transpose of y_pred to make array of shape (506, 30) and sort the array
    # For each data point, sort all 30 y_pred values to compute median
    y_pred = np.transpose(np.array(y_pred))
    y_pred = np.sort(y_pred)
    predicted_y_values = []

    # Compute median on each y_pred value and make a list
    for i in range(len(y_pred)):
        predicted_y_values.append(np.median(y_pred[i]))

    return np.array(predicted_y_values)
```

In [59]:

```
# Predict Y values with all 30 decision trees and compute MSE
def computeMSE(input_x, input_y, decisionTrees, list_selected_columns):
    """This function takes input X and predicts Y pred using random forest and returns MSE"""

    array_of_y = []

    for i in range(len(list_selected_columns)):
        array_of_y.append(decisionTrees[i].predict(input_x[:, list_selected_columns[i]]))

    predict_y_datapoints = predict_Y_of_datapoint(array_of_y)

    return mean_squared_error(input_y, predict_y_datapoints)

computeMSE(x, y, decisionTrees, list_selected_columns)
```

Out[59]:

```
1.2294410439074737
```

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 [60]:

```python
# get oobs predicted y value and compute obb score
def getOob_Y_pred(input_x, input_y, decisionTrees, list_input_data, list_selected_columns):

    """This function checks each point in input dataset with each input sample and
    predicts y value with respective decision tree if the sample doesnt contain the data point"""

    y_preds = []

    # Check for each point
    for i in range(len(input_x)):
        y_preds_datapoint = []

        # Check datapoint in each sample
        for j in range(len(list_selected_columns)):
            datapoint = input_x[i, list_selected_columns[j]].reshape(-1, 3)
            sample_array = list_input_data[j]

            #https://stackoverflow.com/questions/33217660/checking-if-a-numpy-array-contains-anothe
r-array
            if (not (sample_array == datapoint).all(1).any()):
                # Use the model as the model has not been trained with this data point
                y_preds_datapoint.append(decisionTrees[j].predict(datapoint))

        y_preds.append(y_preds_datapoint)

    oob_y_pred = []
    for y_pred_val in y_preds:
        oob_y_pred.append(np.median(np.sort(np.array(y_pred_val))))

    oob_score = 0
    for i in range(x.shape[0]):
        oob_score += ((input_y[i] - oob_y_pred[i]) ** 2)
    oob_score /=  x.shape[0]

    return oob_score

getOob_Y_pred(x,y, decisionTrees, list_input_data, list_selected_columns)
```

Out[60]:

```
28.451222705657134
```

Task 2

In [62]:

```python
MSE_list = []
Oob_scores = []
for itr in range(35):

    # Build sample lists
    list_input_data =[]
    list_output_data =[]
    list_selected_row= []
    list_selected_columns=[]

    for i in range(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)

    # Build decision trees with samples
    decisionTrees = []

    for i in range(len(list_input_data)):
        decisionTrees.append(buildDecisionTree(list_input_data[i], list_output_data[i]))

    # Copute MSE
    MSE_list.append(computeMSE(x, y, decisionTrees, list_selected_columns))
    Oob_scores.append(getOob_Y_pred(x,y, decisionTrees, list_input_data, list_selected_columns))
```

In [72]:

```
from prettytable import PrettyTable
import math
```

In [79]:

```
# Get 10 samples out of MSE_list and Oob_scores values and compute CI

def getCI(population):
    x = PrettyTable(["#samples", "Sample Size", "Sample mean", "Left C.I","Right C.I","Pop mean","C
atch"])
    population = np.array(population)
    population_mean = np.mean(population)

    # Make 10 samples with size 5 and compute CI for all of them
    for i in range(10):
        sample=population[np.random.choice(population.shape[0], 10)]
        sample_mean = sample.mean()
        sample_std =  sample.std()
        sample_size = len(sample)
        # here we are using sample standard deviation instead of population standard deviation
        # Assume we dont know the std-dev of population
        left_limit  = np.round(sample_mean - 2*(sample_std/np.sqrt(sample_size)), 3)
        right_limit = np.round(sample_mean + 2*(sample_std/np.sqrt(sample_size)), 3)
        catch = (population_mean <= right_limit) and (population_mean >= left_limit)

        row = []
        row.append(i+1)
        row.append(sample_size)
        row.append(sample_mean)
        row.append(left_limit)
        row.append(right_limit)
        row.append(population_mean)
        row.append(catch)
        x.add_row(row)
    print(x)
```

In [80]:

```
getCI(MSE_list)
```

#samples	Sample Size	Sample mean	Left C.I	Right C.I	Pop mean	Catch
1	10	1.3302084874400006	0.705	1.955	1.3294221171017317	True
2	10	1.4456273671130953	0.614	2.277	1.3294221171017317	True
3	10	1.6415383065557136	0.793	2.49	1.3294221171017317	True
4	10	1.0305529240176503	0.652	1.409	1.3294221171017317	True
5	10	1.0939774262592388	0.421	1.767	1.3294221171017317	True
6	10	2.157554853385859	1.217	3.098	1.3294221171017317	True
7	10	0.9620705174654619	0.359	1.565	1.3294221171017317	True
8	10	1.443417902445675	0.757	2.129	1.3294221171017317	True
9	10	1.3334081512318146	0.678	1.988	1.3294221171017317	True
10	10	1.0779784292085186	0.558	1.598	1.3294221171017317	True

In [81]:

```
getCI(Oob_scores)
```

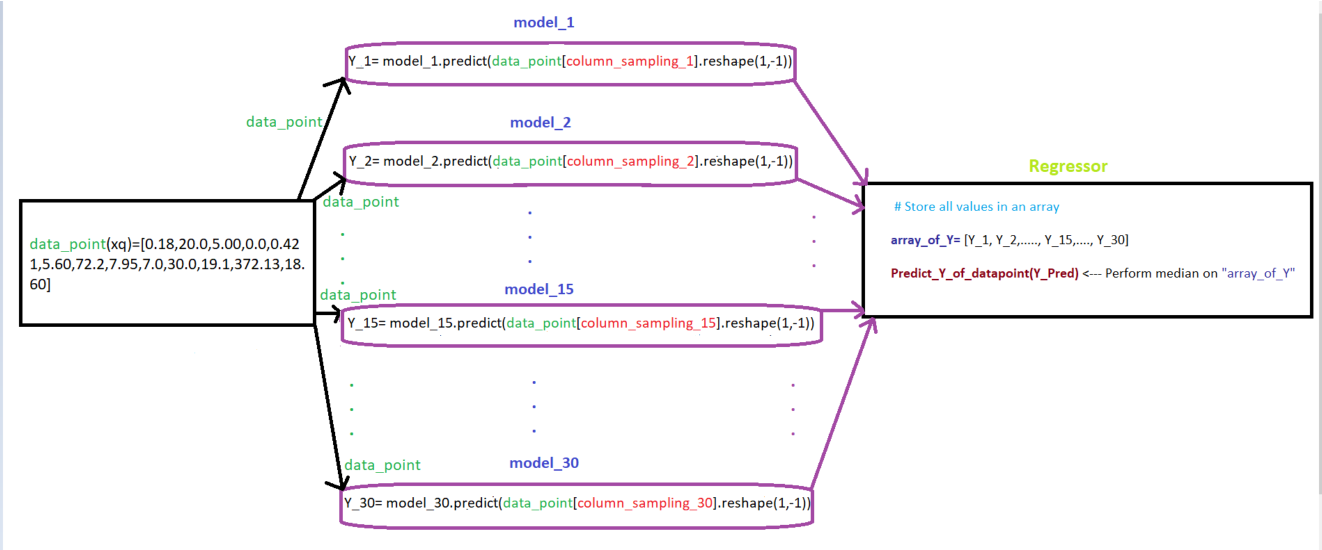
#samples	Sample Size	Sample mean	Left C.I	Right C.I	Pop mean	Catch
1	10	26.88152480584434	23.953	29.81	26.29626551462988 7	True
2	10	25.542205284596214	22.421	28.664	26.296265514629887	True
3	10	26.161440749986134	24.59	27.733	26.296265514629887	True
4	10	26.999307758550593	24.94	29.059	26.296265514629887	True
5	10	25.99267288116721	23.308	28.678	26.296265514629887	True
6	10	27.96396922553409	25.371	30.557	26.296265514629887	True
7	10	27.16238928563642	24.969	29.356	26.296265514629887	True

```
|     8     |      10      | 26.700793818013928 |  25.068   |   28.334   | 26.296265514629887 |  True |
|     9     |      10      | 26.772193668894243 |  24.184   |   29.36    | 26.296265514629887 |  True |
|     10    |      10      | 27.039670106796414 |  24.733   |   29.346   | 26.296265514629887 |  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 [66]:

```
# Compute Yq for input Xq with implemented random forest
yq_preds_30 = []
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])

for i in range(len(list_selected_columns)):
    yq_preds_30.append(decisionTrees[i].predict(xq[list_selected_columns[i]].reshape(-1, 3)))

yq_pred = predict_Y_of_datapoint(yq_preds_30)
print(yq_pred[0])
```

```
18.95
```

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

Task 1:

With bagging, we are getting very less MSE error as we are using high variance base models resulting very less error in training data. The CV error/ Oob score is 28.451. We can try with different number of decision trees and try to reduce the error. But the training MSE is very low 1.229 as we are training and predicting with same dataset.

Task 2:

Here the population and sample mean similar and with Std-dev of samples, we are calculating confidence interval of population mean. Means within the left and right interval, mean can exist in 95% of the points in that interval. We are using the 2nd std-dev value from mean. We have seen, mean actually lies within that range.

the 2nd std-dev value from mean. We have seen, mean actually lies within that range.

Task 3:

We are successfully predicting value for query point Xq.

Here while aggregating, we are using median instead of mean.