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 [26]:
```
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
import scipy
```

In [27]:
```
boston = load_boston()
x = boston.data #independent variables
y = boston.target #target variable
```

In [28]:
```
type(x)
```

Out[28]:
```
numpy.ndarray
```

In [29]:
```
x[:5]
y[:5]
```

Out[29]:
```
array([24. , 21.6, 34.7, 33.4, 36.2])
```

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 $y^i_{pred} = \frac{1}{k}\sum_{\text{k= model which was buit on samples not included } x^i}$(predicted value of x^i with k^{th} 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 [30]:
```
def generating_samples(input_data, target_data):
```

```
    selecting_rows = np.random.choice(len(input_data), 303, replace = False)
    replacing_rows = np.random.choice(selecting_rows, 203, replace = False)
    oob_indices = [i for i in range(len(input_data)) if i not in selecting_rows]
    oob_indices = np.array(oob_indices)
#     print(oob_indices.shape)
#        print(type(selecting_rows), type(oob_indices))


    cols = np.random.randint(4, 13)
    selecting_cols = np.random.choice(input_data.shape[1], cols, replace = False)
    oob_target_indices = [i for i in range(len(target_data)) if i not in selecting_cols]

    sample_data = input_data[selecting_rows[:,None], selecting_cols]
    oob_x_data = input_data[oob_indices[:,None], selecting_cols]
#     print(type(oob_x_data))
    target_of_sample_data = target_data[selecting_rows]
    oob_y_data = target_data[oob_indices]
#     oob_x = input_data[oob_indices[:,None], selecting_cols]
#     oob_y = target_data[oob_target_indices]
    #print(selecting_rows[0:10])
    #print(replacing_rows[0:10])
    #print(len(sample_data))

    #replicating data
    rep_sam_data = input_data[replacing_rows[: , None], selecting_cols]
    target_of_rep_sam_data = target_data[replacing_rows]
    #print(len(rep_sam_data))
    #print(len(target_of_rep_sam_data))

    #concatenating data
    final_sample_data = np.vstack((sample_data, rep_sam_data))
    final_target_data = np.vstack((target_of_sample_data.reshape(-1, 1), target_of_rep_sam_data.reshape(-1,1)))

    return final_sample_data, final_target_data, selecting_rows, selecting_cols, oob_x_data, oob_y_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.ch
    # Please follow above pseudo code for generating samples


    # return sampled_input_data , sampled_target_data,selected_rows,selected_columns
    #note please return as lists
```

Grader function - 1 </fongt>

In [31]:
```
def grader_samples(a,b,c,d,e,f):
    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, e, f = generating_samples(x, y)
print(a.shape)
print(b.shape)
print(c.shape)
print(d.shape)
print(e.shape)
f = f.reshape(-1,1)
print(f.shape)
grader_samples(a,b,c,d,e,f)
```

```
(506, 6)
(506, 1)
(303,)
(6,)
(203, 6)
(203, 1)
```

Out[31]: 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 [32]:
```python
# 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=[]
oob_input_data = []
oob_output_data = []
i = 1
while(i <= 30):
    a, b, c, d, e, f = generating_samples(x, y)
    list_input_data.append(a)
    list_output_data.append(b)
    list_selected_row.append(c)
    list_selected_columns.append(d)
    oob_input_data.append(e)
    oob_output_data.append(f)
    i  += 1
```

In [33]:
```python
print(len(list_input_data[0]))
print(len(list_output_data[0]))
print(len(list_selected_row))
print(len(list_selected_columns))
print(len(oob_input_data[0]))
print(len(oob_output_data[0]))
```

```
506
506
30
30
203
203
```

Grader function - 2

In [34]:
```python
def grader_30(a):
    assert(len(a)==30 and len(a[0])==506)
    return True
grader_30(list_input_data)
```

Out[34]: True

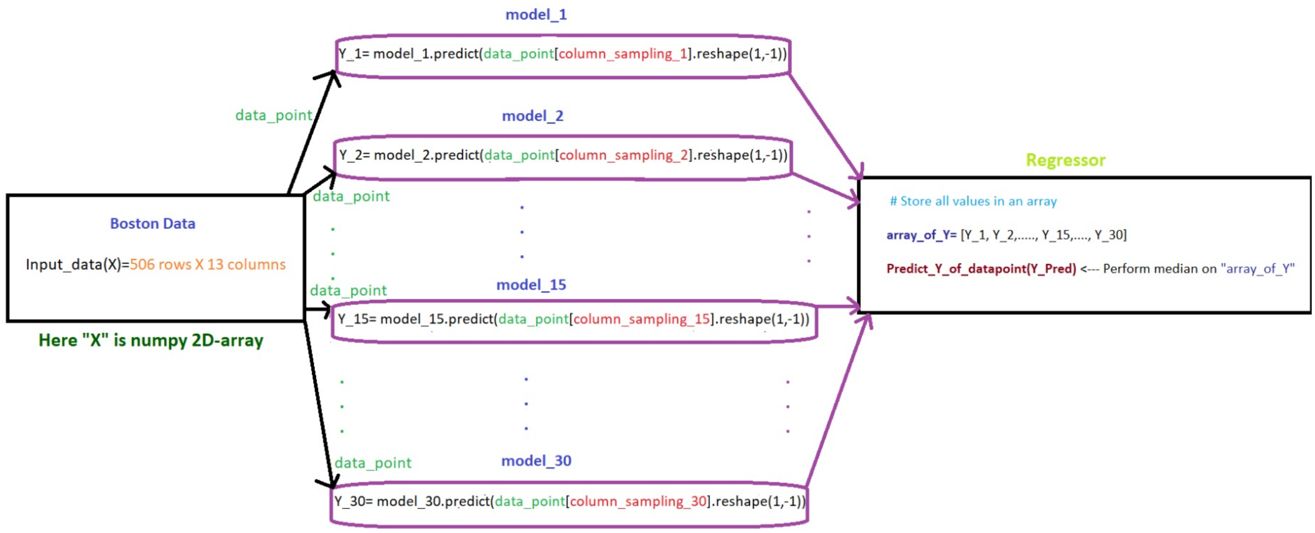
Step - 2

Flowchart for building tree

- **Write code for building regression trees**

```
In [35]: list_of_all_models = []
         for i in range(30):
             model_i = DecisionTreeRegressor(max_depth = None)
             model_i.fit(list_input_data[i], list_output_data[i])
             list_of_all_models.append(model_i)
         # print(list_of_all_models)
```

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 [36]: def Mean_Sqr_Error():
             list_of_all_models = []
             for i in range(30):
                 model_i = DecisionTreeRegressor(max_depth = None)
                 model_i.fit(list_input_data[i], list_output_data[i])
                 list_of_all_models.append(model_i)
             array_of_Y = []
             for i in range(30):
                 data_point_i = x[:, list_selected_columns[i]]
             #     print(type(data_point_i))
             #     print(data_point_i)
                 Y_i = list_of_all_models[i].predict(data_point_i)
                 array_of_Y.append(Y_i)

             array_of_Y = np.array(array_of_Y).transpose()
             # pred_array_of_Y = array_of_Y.t

         #     print(len(array_of_Y[0]))
             median_y = np.median(array_of_Y, axis = 1)
             # print(median_y)
             MSE = mean_squared_error(y, median_y)
             return MSE
```

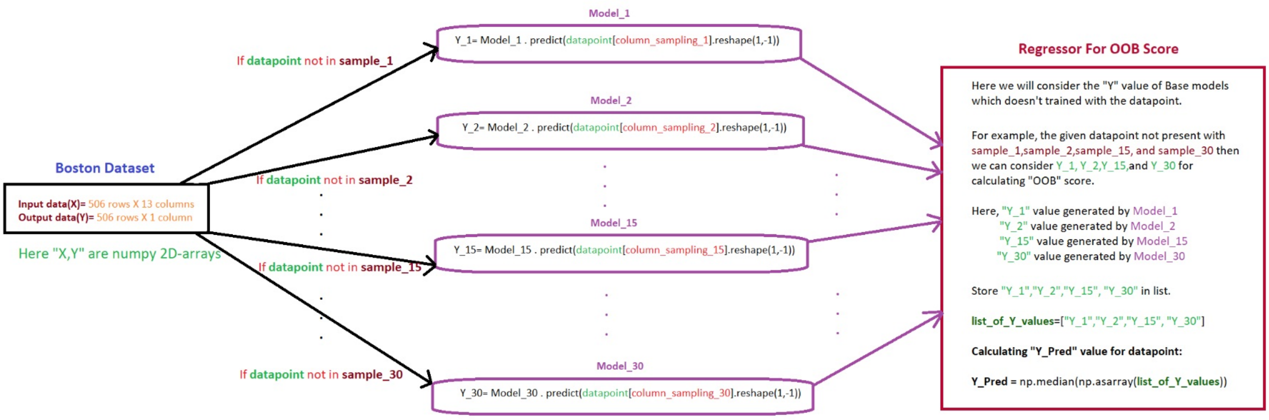
```
In [37]: Mean_Sqr_Error()
```

```
Out[37]: 0.03411067193675891
```

```
In [ ]:
```

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 [38]:
```
def calculate_oob_score(num_rows):
    list_of_all_models_oob = []
    for i in range(30):
        model_i = DecisionTreeRegressor(max_depth = None)
        model_i.fit(oob_input_data[i], oob_output_data[i])
        list_of_all_models_oob.append(model_i)
    array_of_Y_oob = []
    for i in range(30):
        data_point_i = x[:, list_selected_columns[i]]
#     print(type(data_point_i))
#     print(data_point_i)
        Y_i = list_of_all_models_oob[i].predict(data_point_i)
        array_of_Y_oob.append(Y_i)
    array_of_Y_oob = np.array(array_of_Y_oob).transpose()
# pred_array_of_Y = array_of_Y.t

# print(len(array_of_Y[0]))
    median_y_oob = np.median(array_of_Y_oob, axis = 1)
# print(median_y)

    oob_score = 0
    for i in range(0, num_rows):
        oob_score += ((y[i] - median_y_oob[i] ) ** 2)
        final_oob_score = oob_score/506
    return final_oob_score
```

In [39]:
```
print("final_oob_score is ", calculate_oob_score(506))
```

```
final_oob_score is  2.61698740118577
```

Task 2

In [40]:
```
# https://medium.com/analytics-vidhya/why-bootstrap-is-useful-and-implementation-of-bootstrap-sampling-in-random-
# Repeat Task 1 for 35 times, and for each iteration store the Train MSE and OOB score

MSE_35_arr = []
oob_35_arr = []
for i in range(35):
    mse = Mean_Sqr_Error()
    oob = calculate_oob_score(506)
    MSE_35_arr.append(mse)
    oob_35_arr.append(oob)

MSE_35_arr = np.array(MSE_35_arr)
oob_35_arr = np.array(oob_35_arr)
```

```python
confidence_level = 0.95
degrees_of_freedom = 34 # sample.size - 1

# mean and std err value for the lists MSE_35 and oob_35
mean_MSE_35 = np.mean(MSE_35_arr)
std_err_MSE_35 = scipy.stats.sem(MSE_35_arr)

mean_oob_35 = np.mean(oob_35_arr)
std_err_oob_35 = scipy.stats.sem(oob_35_arr)

CI_MSE_35 = scipy.stats.t.interval(confidence_level, degrees_of_freedom, mean_MSE_35, std_err_MSE_35)
CI_oob_35 = scipy.stats.t.interval(confidence_level, degrees_of_freedom, mean_oob_35, std_err_oob_35)

print("Confidence Interval of MSE:", CI_MSE_35)
print("Confidence Interval of OOB:", CI_oob_35)
```

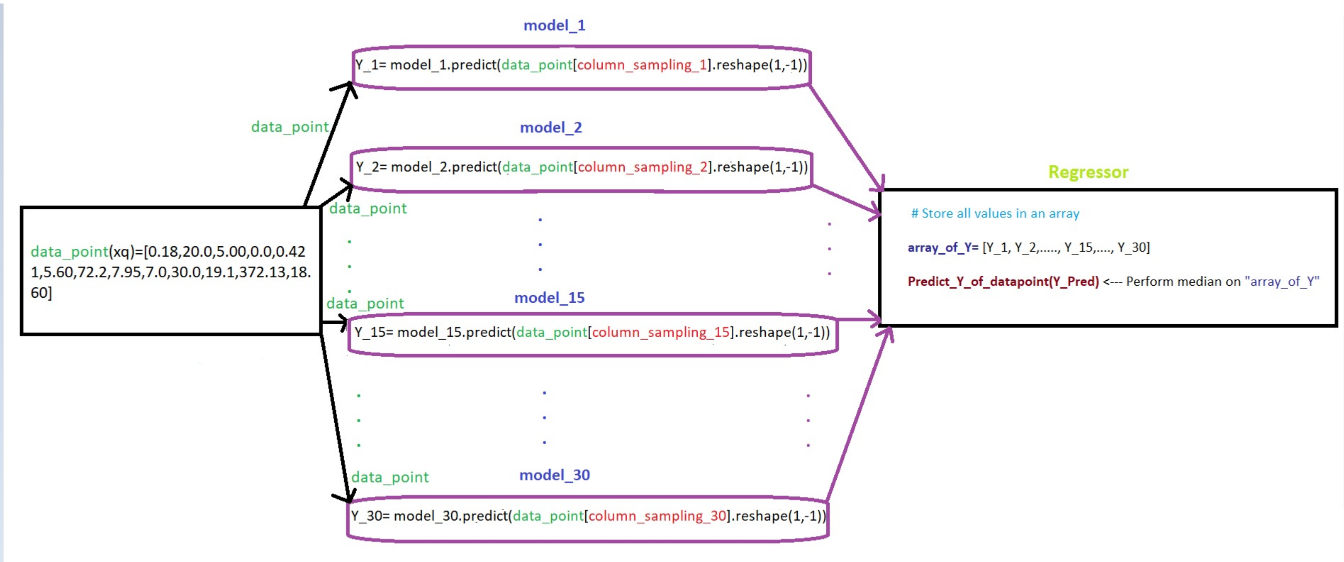
```
Confidence Interval of MSE: (0.03831510028462849, 0.041337920607522866)
Confidence Interval of OOB: (3.0241602439657935, 3.218929818146006)
```

In [ ]:

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 [41]:

```python
#https://medium.com/analytics-vidhya/why-bootstrap-is-useful-and-implementation-of-bootstrap-sampling-in-random-
#Predicting house value for given query
def predict_y_given_x(x_query):
    y_predict = []
    for i in range(0, 30):
    # Extract x for ith data point with specific number of featues from list_selected_columns
        data_point_i = [x_query[column] for column in list_selected_columns[i]]
        data_point_i = np.array(data_point_i).reshape(1, -1)
        predicted_i = list_of_all_models[i].predict(data_point_i)
        y_predict.append(predicted_i)
    y_predict = np.array(y_predict)
    y_predicted_median = np.median(y_predict)
    return y_predicted_median


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]
y_pred_for_xq = predict_y_given_x(xq)
y_pred_for_xq
```

Out[41]: 18.799999999999997

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

Observation for task1:

```
1. Selecting 303 points from orginal data and duplicate 203 points from the selected points for
30 times. also out of bagging points also saved. 303(selected points) + 203(oob) = 506(original
points). This oob points used later for oob score calculation.
2. 30 models with the base learner as DecisionTrees created and bootstrap samples are fitted in
the models. then Mean sqaure error is calculated with help of target var y and median of 30
models (y_median_pred).
3. function for OOB score calculation is done and output is recorded.
```

Observation for task2: Confidence Interval for both bootstrap sample and OOB MSE — There is a 95% chance that the confidence interval of (0.03831510028462849, 0.041337920607522866) contains the true population mean of MSE. OOB Score — There is a 95% chance that the confidence interval of (3.0241602439657935, 3.218929818146006) contains the true population mean of OOB Score.

Observation for task 3: Predicted house rate given query xq is 18.79

In [ ]:

Processing math: 100%