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.tree import DecisionTreeRegressor
from sklearn.datasets import load_boston # here we are using sklearn's boston da
from sklearn.metrics import mean_squared_error # importing mean_squared_error me
import random
```

In [2]:
```
boston = load_boston()
x=boston.data #independent variables
y=boston.target #target variable
```

In [3]:
```
x.shape
```

Out[3]:
```
(506, 13)
```

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**
  $y^{i}_{pred} = \frac{1}{k}\sum_{\text{k= model which was buit on samples not included } x^i}(\text{predicted value of } x^i \text{ with } k^{th} \text{ mode}$
  .
- **Now calculate the** $OOBScore = \frac{1}{506}\sum_{i=1}^{506}(y^i - y^i_{pred})^2$**.**

Task 2

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 [4]:

```
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/re
    # Please follow above pseudo code for generating samples


    # return sampled_input_data , sampled_target_data,selected_rows,selected_col
    #note please return as lists

    row_303 = random.sample(range(0, input_data.shape[0]), 303)
    row_203 = random.sample(row_303, 203)
    n_cols = random.randint(3,13)
    cols = random.sample(range(13), n_cols)

    data_303 = input_data[np.ix_(row_303, cols)]
    data_203 = input_data[np.ix_(row_203, cols)]

    sample_data = np.vstack((data_303, data_203))
    sample_target = np.vstack((target_data[row_303].reshape(-1,1), target_data[r

    # row_303.extend(row_203)

    return sample_data, sample_target, row_303, cols
```

Grader function - 1 </fongt>

In [5]:

```
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[5]: 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 [6]:

```
# 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

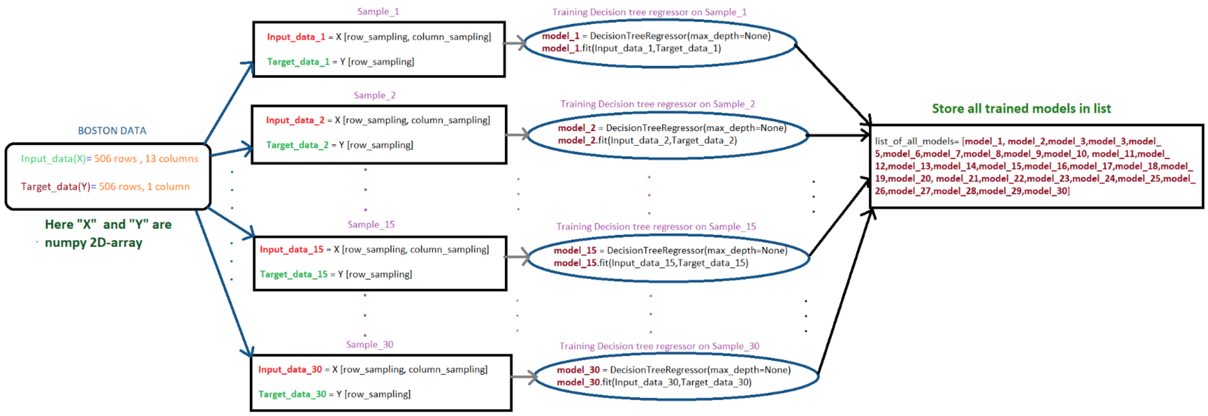
In [7]:

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

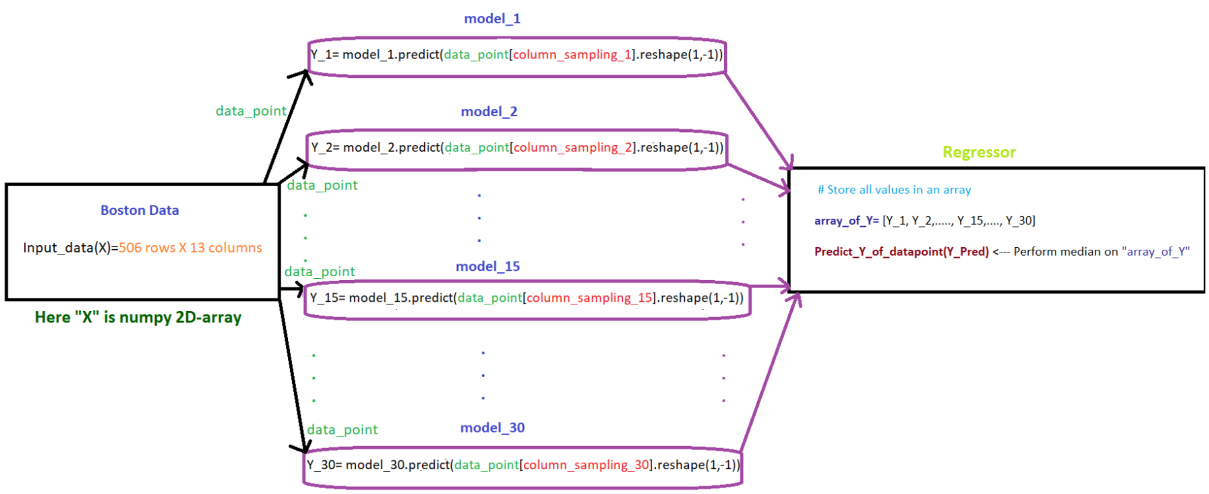
Out[7]: True

Step - 2

Flowchart for building tree



Flowchart for calculating MSE

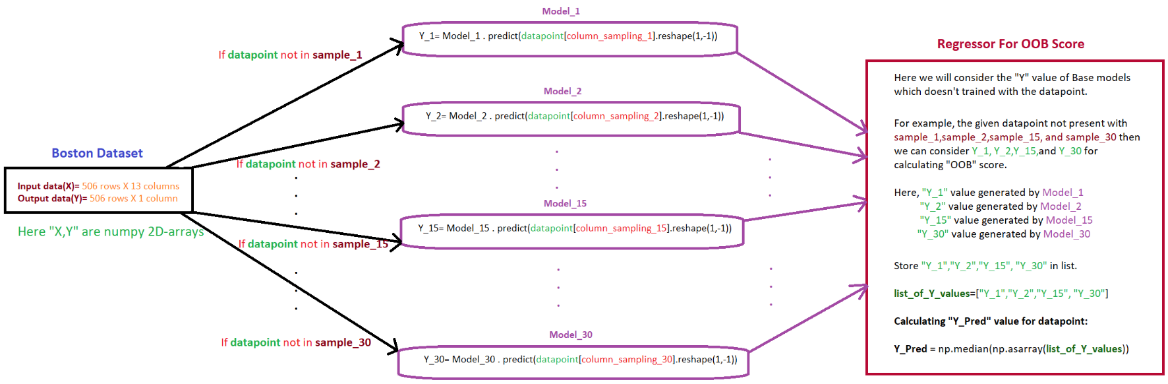


After getting predicted_y for each data point, we can use sklearns mean_squared_error to calculate the MSE between predicted_y and actual_y.

- **Write code for calculating MSE**

Step - 3

Flowchart for calculating OOB score



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

In [8]:

```
def get_base_models(X, y, o_data, row_i, col_i):
    """
    This function returns the base Decision Trees that are fit to the sampled da
    """
    model = DecisionTreeRegressor(max_depth=None)
    model.fit(X, y)

    return model
```

In [9]:

```
def get_mse_score(y, yi):
    """
    This function returns the MSE score between the predicted and original value
    """
    sum = 0

    for i in range(len(y)):
        sum += (y[i] - yi[i])**2

    return sum/len(y)
```

In [10]:

```
def get_models(list_input_data, list_output_data, x, list_selected_row, list_sel
    """
    This function returns a list of models that are fit to the data
    """
    models = []

    for i in range(30):
        model = get_base_models(list_input_data[i], list_output_data[i], x, list
        models.append(model)

    return models
```

In [12]:

```
def pred_and_mse(models, list_selected_columns):
    """
    This function takes models and columns as inputs and returns the MSE score a
    """
```

```
    yi = np.zeros((506,1))

    for i in range(506):
        temp = []
        for j in range(30):
            temp.append(models[j].predict(x[i][list_selected_columns[j]].reshape
        yi[i] = np.median(temp)

    mse = get_mse_score(y, yi)

    return yi, mse
```

In [14]:

```
def get_oob(list_selected_row, list_selected_columns, models):
    """
    This function returns the OOB score using the models obtained
    """
    y_pred = np.zeros((506,1))

    for i in range(506):
        temp = []
        y_temp = []
        for j in range(30):
            if i not in list_selected_row[j]:
                temp.append(j)
        for j in temp:
            y_temp.append(models[j].predict(x[i][list_selected_columns[j]].resha

        y_pred[i] = np.median(y_temp)

    oob_score = 0

    for i in range(506):
        oob_score += (y[i] - y_pred[i])**2

    oob_score /= 506

    return oob_score
```

In [15]:

```
#  Calling all functions and getting the OOB and MSE scores

models = get_models(list_input_data, list_output_data, x, list_selected_row, lis
yi, mse = pred_and_mse(models, list_selected_columns)
oob = get_oob(list_selected_row, list_selected_columns, models)
```

In [16]:

```
print(f"The MSE Score is {mse} and the OOB score is {oob}")
```

```
The MSE Score is [0.1359574] and the OOB score is [12.95518061]
```

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 [17]:

```
mses, oobs = [], []

for i in range(35):

    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)

    models = get_models(list_input_data, list_output_data, x, list_selected_row,
    yi, mse = pred_and_mse(models, list_selected_columns)
    oob = get_oob(list_selected_row, list_selected_columns, models)
    mses.append(mse)
    oobs.append(oob)
```

In [18]:

```
n = 35

mses_mean = np.mean(mses)
oobs_mean = np.mean(oobs)

mses_std = np.std(mses)
oobs_std = np.std(oobs)

mses_se = mses_std/np.sqrt(n)
oobs_se = oobs_std/np.sqrt(n)

mses_lci =  mses_mean - 2 * (mses_se)
mses_rci =  mses_mean + 2 * (mses_se)

oobs_lci =  oobs_mean - 2 * (oobs_se)
oobs_rci =  oobs_mean + 2 * (oobs_se)
```

In [20]:

```
from prettytable import PrettyTable

x = PrettyTable()
x = PrettyTable(["Value", "Sample Size", "Sample Mean", "Sample Std", "Sample St
x.add_row(["MSE", "35", mses_mean, mses_std, mses_se, mses_lci, mses_rci])
x.add_row(["OOB", "35", oobs_mean, oobs_std, oobs_se, oobs_lci, oobs_rci])
```

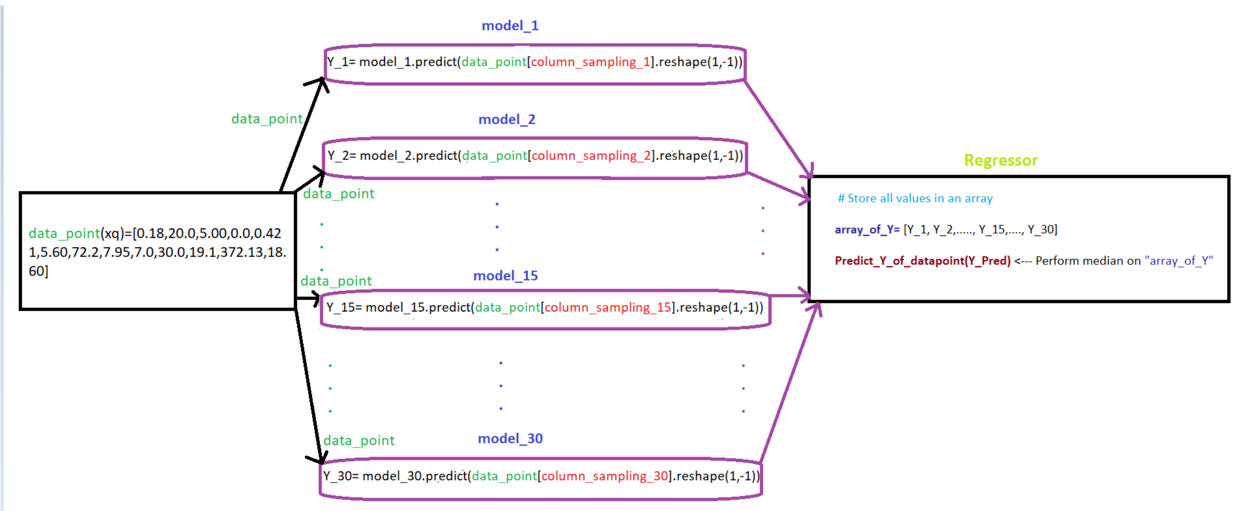
```
print(x)
```

```
+-------+------------+--------------------+--------------------+------------
---------+--------------------+--------------------+
| Value | Sample Size |     Sample Mean     |      Sample Std      | Sample Stand
ard Error |       Left C.I        |       Right C.I        |
+-------+------------+--------------------+--------------------+------------
---------+--------------------+--------------------+
|  MSE  |      35     | 0.09412189233683693 | 0.08042700339370845 |  0.013594644
822651507 | 0.06693260269153392 | 0.12131118198213994 |
|  OOB  |      35     |  13.568222592063591 |  1.3726688895940893 |  0.232023390
47478257  |  13.104175811114027 |  14.032269373013156 |
+-------+------------+--------------------+--------------------+------------
---------+--------------------+--------------------+
```

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 [21]:
```
xq = np.array([0.18,20.0,5.00,0.0,0.421,5.60,72.2,7.95,7.0,30.0,19.1,372.13,18.6
yq = []

for i, model in enumerate(models):
    yq.append(model.predict(xq[list_selected_columns[i]].reshape(1,-1)))

y_pred = np.median(yq)

y_pred
```

Out[21]: 18.9

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

The MSE Score of the Bootstrap is 0.1359574

The OOB Score of the Boostrap is 12.95518061

The prediction of the Query Point is 18.9

The Confidence Interval of MSE Sample is [0.06693260269153392 , 0.12131118198213994]

The Confidence Interval of OOB Sample is [13.104175811114027 | 14.032269373013156]