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

```
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
import pandas as pd
import numpy as np
import random
from sklearn.tree import DecisionTreeRegressor
import scipy
```

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

```
x.shape
```

```
(506, 13)
```

```
x[:5]
```

```
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]])
```

```
y[:5]
```

```
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
  $y^i_{pred} = \frac{1}{k}\sum_{\text{k= model which was buit on samples not included } x^i}(\text{predicted value of } x^i \text{ with } k$
  .

- 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
```

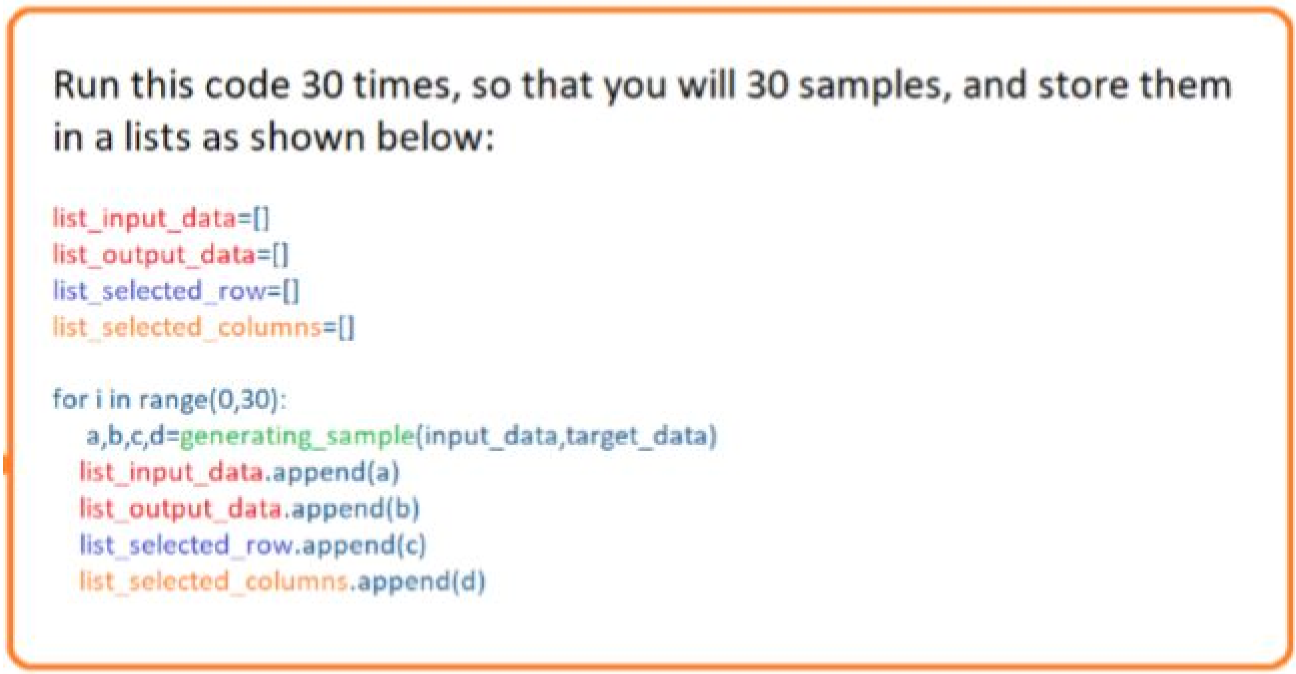
- **Write code for generating samples**

```
    '''In 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/ge
    # Please follow above pseudo code for generating samples
    # note please return as lists

def generating_samples(input_data, target_data):
  selecting_rows = np.random.choice(len(input_data), 303, replace=False) #selection of 303
  replacing_rows = np.random.choice(selecting_rows, 203, replace=False) #extracting 206 ra
  selecting_columns = random.randint(3, 13) # selecting columns
  columns_selected = np.array(random.sample(range(0, 13), selecting_columns)) #select rand
  sample_data = input_data[selecting_rows[:, None], columns_selected]
  target_of_sample_data = target_data[selecting_rows]

  #Replicating data
  replicate_sample_data = input_data[replacing_rows[:, None], columns_selected ]
  target_of_replicate_sample_data = target_data[replacing_rows]

  #Concatinating data
  final_sample_data =  np.vstack((sample_data, replicate_sample_data))
  final_target_data =  np.vstack((target_of_sample_data.reshape(-1, 1), target_of_replicat

  return final_sample_data, final_target_data, selecting_rows, columns_selected



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)
    print("Final sample data")
    print(a.shape)
    print("-"*50)
    print("Final Target Data")
    print(b.shape)
    print("-"*50)
    print("Rows Selected")
    print(c.shape)
    print("-"*50)
    print("Columns Selected")
    print(d.shape)
    print("-"*50)
    grader_samples(a,b,c,d)
```

```
Final sample data
(506, 3)
--------------------------------------------------
Final Target Data
(506, 1)
--------------------------------------------------
Rows Selected
(303,)
--------------------------------------------------
Columns Selected
(3,)
--------------------------------------------------
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)
```

```
# 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)
```

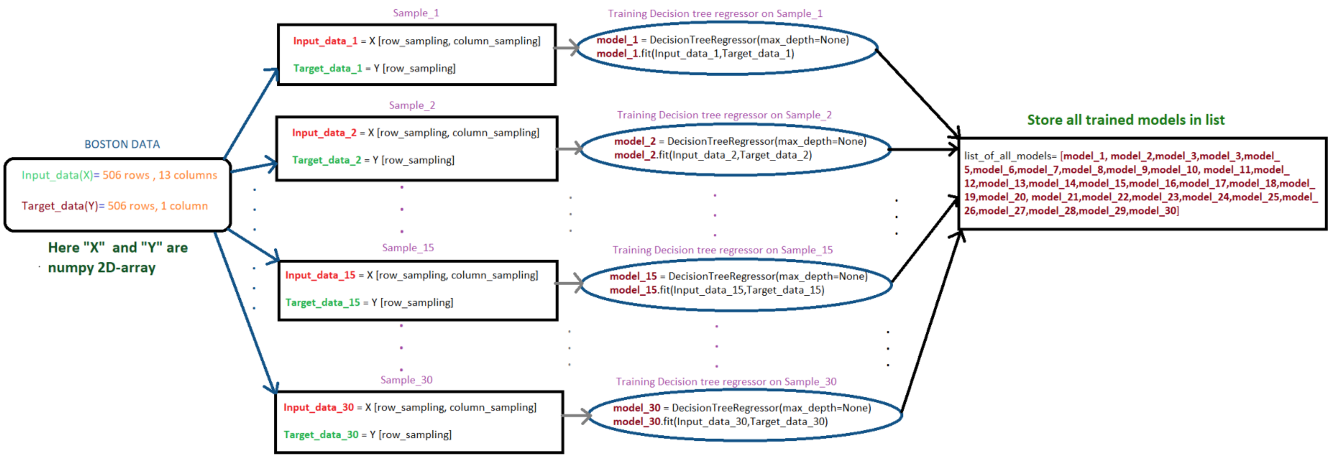
Grader function - 2

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

```
True
```

Step - 2

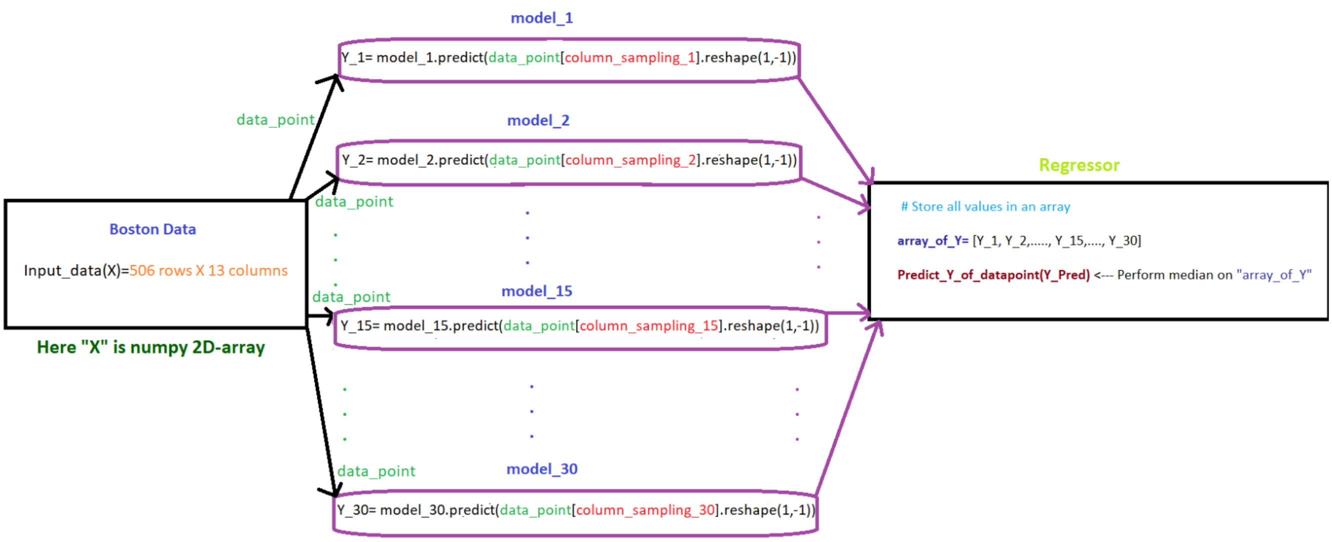
Flowchart for building tree



- **Write code for building regression trees**

```
list_of_all_models_decision_tree = []
for i in range(0, 30):
  model_i = DecisionTreeRegressor(max_depth=None)
  model_i.fit(list_input_data[i], list_output_data[i])
  list_of_all_models_decision_tree.append(model_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**

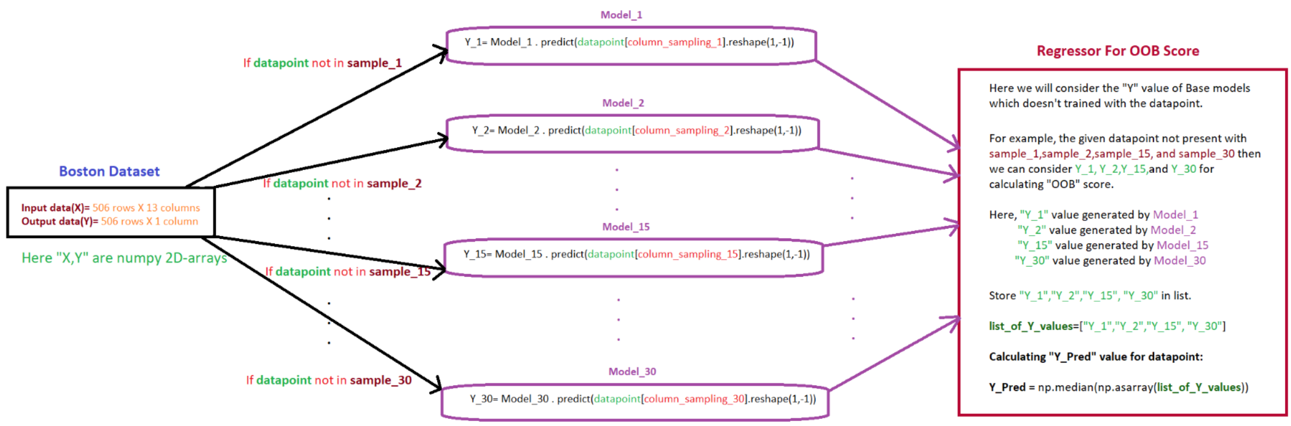
```
from sklearn.metrics import mean_squared_error
from statistics import median
from statistics import mean

array_of_y =[]

for i in range (0,30):
  data_point_i = x[:, list_selected_columns[i]]
  target_y_i = list_of_all_models_decision_tree[i].predict(data_point_i)
  array_of_y.append(target_y_i)

predicted_array_of_target_y = np.array(array_of_y)
predicted_array_of_target_y = predicted_array_of_target_y.transpose()

print("Predicted values using Decision Tree:")
print(predicted_array_of_target_y[:5,])
print("-"*50)
print("Dimensions of Predicted Array of Target y : ", predicted_array_of_target_y.shape) #
print("-"*50)
```

```
# Now to calculate MSE, first calculate the Median of Predicted Y
# passing axis=1 will make sure the medians are computed along axis=1

# Now to calculate MSE, first calculate the Median of Predicted Y
# passing axis=1 will make sure the medians are computed along axis=1

median_predicted_y = np.median(predicted_array_of_target_y, axis=1)
median_predicted_y.shape
print("MSE (Using Median): ", mean_squared_error(y, median_predicted_y ))

print("-"*50)

mean_predicted_y = np.mean(predicted_array_of_target_y, axis=1)
mean_predicted_y.shape
print("MSE (Using Mean): ", mean_squared_error(y, mean_predicted_y ))
```

```
Predicted values using Decision Tree:
[[24.         32.2        18.9        23.1        22.         32.7
  24.         19.6        24.         32.         24.         23.8
  24.         27.9        24.         28.         35.1        24.
  24.         37.2        24.         24.         24.         35.1
  24.         23.6        24.         31.6        23.8        24.        ]
 [21.6        21.6        21.6        21.7        21.6        20.47142857
  21.6        34.7        21.6        21.6        21.6        21.6
  23.6        22.         22.9        21.6        24.6        20.7
  21.6        21.6        21.6        21.2        24.4        11.9
  21.6        21.6        21.6        21.6        21.6        24.4       ]
 [21.6        21.6        34.7        34.7        22.5        20.47142857
  34.7        34.7        34.7        34.7        36.1        35.4
  33.         34.7        37.9        34.7        34.7        33.3
  34.7        34.7        44.8        34.7        33.4        34.7
  34.7        33.1        34.7        34.7        34.7        34.7       ]
 [33.4        33.4        28.7        33.4        33.4        32.76666667
  33.4        35.2        33.4        33.4        29.6        33.4
  33.         33.4        26.6        33.4        35.4        33.4
  50.         33.4        33.1        50.         33.4        33.4
  33.4        33.4        33.4        33.4        33.4        33.4       ]
 [36.2        36.2        28.7        36.2        33.4        32.76666667
  36.2        35.2        36.2        33.         36.2        37.9
  28.5        33.         36.2        33.4        33.         36.2
  36.2        26.6        36.2        36.2        36.2        34.7
  36.2        36.2        33.4        32.7        36.2        36.2       ]]
--------------------------------------------------
Dimensions of Predicted Array of Target y :  (506, 30)
--------------------------------------------------
MSE (Using Median):  0.09576127288860214
--------------------------------------------------
MSE (Using Mean):  2.68640227672939
```

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**

```
y_predicted_oob_median_list = []

for i in range(0, 506):
  indices_for_oob_models = []

  # For each of i-th row I have to build a list, of sample size 30
  # ONLY condition is i-th row should not be part of the list_selected_row[i-th]
  # e.g. say for i = 469 and index_oob in below loop is 10 then
  # list_selected_row[10] (which is an array of row-numbers) should not contain the 469-th

  for index_oob in range(0, 30):
    if i not in list_selected_row[index_oob]:
      indices_for_oob_models.append(index_oob)


  y_predicted_oob_list = []

  for oob_model_index in indices_for_oob_models:
    model_oob = list_of_all_models_decision_tree[oob_model_index]

    row_oob = x[i]

    #Now extract ONLY those specific columns/featues that were selected during the bootstr

    x_oob_data_point = [row_oob[columns] for columns in list_selected_columns[oob_model_in

    x_oob_data_point = np.array(x_oob_data_point).reshape(1, -1) #print('np.array(x_oob_da

    y_predicted_oob_data_point = model_oob.predict(x_oob_data_point)
    y_predicted_oob_list.append(y_predicted_oob_data_point)
```

```
  y_predicted_oob_list = np.array(y_predicted_oob_list)
  y_predicted_median = np.median(y_predicted_oob_list)
  y_predicted_oob_median_list.append(y_predicted_median)

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

print("Final OOB Score is ", calculate_oob_score(506))
```

```
Final OOB Score is  15.350442599179564
```

Task 2

```
# Function to build the entire bootstrapping steps that we did above and
# Reurning from the function the MSE and oob score
def bootstrapping_and_oob(x, y):

  # 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)

   #Building Regression Trees:
  list_of_all_models_decision_tree = []
  for i in range(0, 30):
    model_i = DecisionTreeRegressor(max_depth=None)
    model_i.fit(list_input_data[i], list_output_data[i])
    list_of_all_models_decision_tree.append(model_i)

   # calculating MSE
  array_of_Y = []

  for i in range(0, 30):
    data_point_i = x[:, list_selected_columns[i]]
    target_y_i = list_of_all_models_decision_tree[i].predict(data_point_i)
    array_of_Y.append(target_y_i)


  predicted_array_of_target_y = np.array(array_of_Y)
```

```
    predicted_array_of_target_y = np.array(array_of_Y)
    predicted_array_of_target_y = predicted_array_of_target_y.transpose()

      # print(predicted_array_of_target_y.shape)
      # Now to calculate MSE, first calculate the Median of Predicted Y
      # passing axis=1 will make sure the medians are computed along axis=1

    median_predicted_y = np.median(predicted_array_of_target_y, axis=1)

    MSE = mean_squared_error(y, median_predicted_y ) #Final MSE

      # Calculating OOB Score
    y_predicted_oob_median_list = []
    for i in range(0, 506):
      indices_for_oob_models = []

      # For each of i-th row I shall build a list of sample size 30
      # ONLY condition being that this ith row should not be part of
      # the list_selected_row


      for index_oob in range(0, 30):
        if i not in list_selected_row[index_oob]:
          indices_for_oob_models.append(index_oob)

      y_predicted_oob_list = []
      for oob_model_index in indices_for_oob_models:
        model_oob = list_of_all_models_decision_tree[oob_model_index]
        row_oob = x[i]
          # print('oob_model_index ', oob_model_index)
        x_oob_data_point = [row_oob[col] for col in list_selected_columns[oob_model_index] ]
          # print('np.array(x_oob_data_point) ', np.array(x_oob_data_point))
        x_oob_data_point = np.array(x_oob_data_point).reshape(1, -1)

        y_predicted_oob_data_point = model_oob.predict(x_oob_data_point)
        y_predicted_oob_list.append(y_predicted_oob_data_point)

      y_predicted_oob_list = np.array(y_predicted_oob_list)
      y_predicted_median = np.median(y_predicted_oob_list)
      y_predicted_oob_median_list.append(y_predicted_median)



    oob_score = 0

    for i in range(0, 506):
      oob_score += (y[i] - y_predicted_oob_median_list[i] ) ** 2  # oob_score = (oob_score +

    final_oob_score = oob_score/506
    return MSE, final_oob_score

  print(bootstrapping_and_oob(x, y))
```

    (0.063801004697107, 15.731620219554825)

```
x=boston.data #independent variables
```

```
y=boston.target #target variable

mse_boston_35_times_arr = []
oob_score_boston_35_times_arr = []

# Repeat Task 1 for 35 times, and for each iteration store the Train MSE and OOB score
for i in range(0, 35):
  mse, oob_score = bootstrapping_and_oob(x, y)
  mse_boston_35_times_arr.append(mse)
  oob_score_boston_35_times_arr.append(oob_score)


mse_boston_35_times_arr = np.array(mse_boston_35_times_arr)
oob_score_boston_35_times_arr = np.array(oob_score_boston_35_times_arr)

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

mean_of_sample_mse_35 = np.mean(mse_boston_35_times_arr)
standard_error_of_sample_mse_35 = scipy.stats.sem(mse_boston_35_times_arr)

# confidence_interval = scipy.stats.t.interval(confidence_level, degrees_freedom, sample_m
confidence_interval_mse_35 = scipy.stats.t.interval(confidence_level, degrees_of_freedom,
print("Confidence Interval of MSE 35 Times ", confidence_interval_mse_35)

# Now calculate confidence inter for oob score
mean_of_sample_oob_score_35 = np.mean(oob_score_boston_35_times_arr)
standard_error_of_sample_oob_score_35 = scipy.stats.sem(oob_score_boston_35_times_arr)

confidence_interval_oob_score_35 = scipy.stats.t.interval(confidence_level, degrees_of_fre
print("Confidence Interval of OOB Score 35 Times ", confidence_interval_oob_score_35)
```

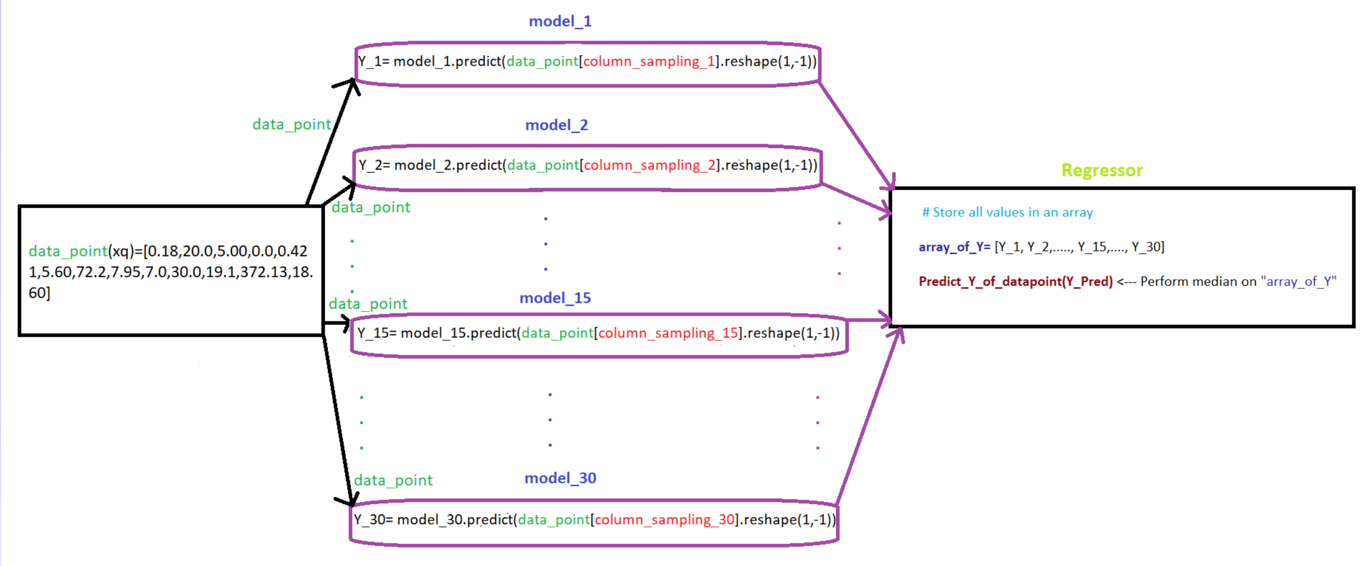
```
Confidence Interval of MSE 35 Times  (0.06969089322660793, 0.1382185426569405)
Confidence Interval of OOB Score 35 Times  (13.441286506283376, 14.703192818207302)
```

- MSE - There is a 95% chance that the confidence interval of (0.06969089322660793, 0.1382185426569405) contains the true population mean of MSE.
- OOB Score - There is a 95% chance that the confidence interval of (13.441286506283376, 14.703192818207302) contains the true population mean of OOB Score.

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

```
def predict_y_given_x_bootstrap(x_query):
  y_predicted_array_30_sample = []

  for i in range(0, 30):
    model_i = list_of_all_models_decision_tree[i]

    # Extract x for ith data point with specific number of featues from list_selected_colu

    x_data_point_i = [x_query[column] for column in list_selected_columns[i]]
    x_data_point_i = np.array(x_data_point_i).reshape(1, -1)
    y_predicted_i = model_i.predict(x_data_point_i)
    y_predicted_array_30_sample.append(y_predicted_i)

  y_predicted_array_30_sample = np.array(y_predicted_array_30_sample)
  y_predicted_median = np.median(y_predicted_array_30_sample)
  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_predicted_for_xq = predict_y_given_x_bootstrap(xq)
y_predicted_for_xq
```

```
18.7
```

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

Observation for Task 1:

- Successfuly create Random 30 samples from the whole boston data points using functions like generating_samples(x, y) & grader_30(a)
- Successfully built decision tree regressor on each sample

- After getting predicted_y for each data point, we useD sklearns mean_squared_error to calculate the MSE & OOB Score between predicted_y and actual_y.

Observation for Task 2:

- Identified Confidence Interval for OOB Score as well as for MSE.
- MSE - There is a 95% chance that the confidence interval of (0.06969089322660793, 0.1382185426569405) contains the true population mean of MSE.
- OOB Score - There is a 95% chance that the confidence interval of (13.441286506283376, 14.703192818207302) contains the true population mean of OOB Score.

Observation for Task 3:

- Given query point "xq" to 30 models and performed the regression on the output generated by 30 models
- Since we choose to take median on predict_y_given_x_bootstrap, the output generated by 30 models is 18.7

0s completed at 8:58 PM