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 random
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
from statistics import median
```

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

```
    /usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function load_boston is deprecated; `loa

        The Boston housing prices dataset has an ethical problem. You can refer to
        the documentation of this function for further details.

        The scikit-learn maintainers therefore strongly discourage the use of this
        dataset unless the purpose of the code is to study and educate about
        ethical issues in data science and machine learning.

        In this special case, you can fetch the dataset from the original
        source::
```

```
        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

```
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]])
```

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
- **Note - While selecting the random 60% datapoints from the whole data, make sure that the selected datapoints are all exclusive, repetition is not allowed.**

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.

A few key points

- Remember that the datapoints used for calculating MSE score contain some datapoints that were initially used while training the base learners (the 60% sampling). This makes these datapoints partially seen (i.e. the datapoints used for calculating the MSE score are a mixture of seen and unseen data). Whereas, the datapoints used for calculating OOB score have only the unseen data. This makes these datapoints completely unseen and therefore appropriate for testing the model's performance on unseen data.
- Given the information above, if your logic is correct, the calculated MSE score should be less than the OOB score.
- The MSE score must lie between 0 and 10.
- The OOB score must lie between 10 and 35.
- The difference between the left nad right confidence-interval values must not be more than 10. Make sure this is true for both MSE and OOB confidence-interval values.

Task - 1

Step - 1

- **Creating samples**

Algorithm

Pseudo code for generating sampes

- **Write code for generating samples**

```
# refer : above suedo code


def generating_samples(input_data, target_data):


  # random choice for generating the index without any replacement
  # selecting 303 random row indices from the input_data, without replacement
  rows_selected = np.random.choice(len(input_data), 303, replace=False)

  # so now will replicate 203 index from above selected rows
  # Replacing Rows => Extracting 206 reandom row indices from the abvoe rows_selected
  rows_203_extracted_from_rows_selected = np.random.choice(rows_selected, 203, replace=False)

  # Now get 3 to 13 random column indices from input_data
  number_of_columns_to_select = random.randint(3, 13)
  columns_selected = np.array(random.sample(range(0, 13), number_of_columns_to_select ))

  sample_data = input_data[rows_selected[:, None], columns_selected]

  target_of_sample_data = target_data[rows_selected]

  # Now Replication of Data for 203 data points out of 303 selected points

  replicated_203_sample_data_points = input_data[rows_203_extracted_from_rows_selected[:, None], columns_selected ]

  target_203_replicated_sample_data = target_data[rows_203_extracted_from_rows_selected]

  # now we will concating the selected row and datapoints

  final_sample_data = np.vstack((sample_data, replicated_203_sample_data_points ))

  final_target_data = np.vstack((target_of_sample_data.reshape(-1, 1), target_203_replicated_sample_data.reshape(-1, 1) ))
```

```
  final_target_data = np.vstack((target_of_sample_data.reshape(-1, 1), target_203_replicated_sample_data.reshape(-1, 1) ))

  return final_sample_data, final_target_data, rows_selected, columns_selected
```

Grader function - 1

```
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)
```

```
a,b,c,d = generating_samples(x, y)
print("shape of a" ,a.shape)
print("shape of b" ,b.shape)
print("shape of c " ,c.shape)
print("shape of d " ,d.shape)
grader_samples(a,b,c,d)
```

```
shape of a (506, 10)
shape of b (506, 1)
shape of c  (303,)
shape of d  (10,)
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)
```

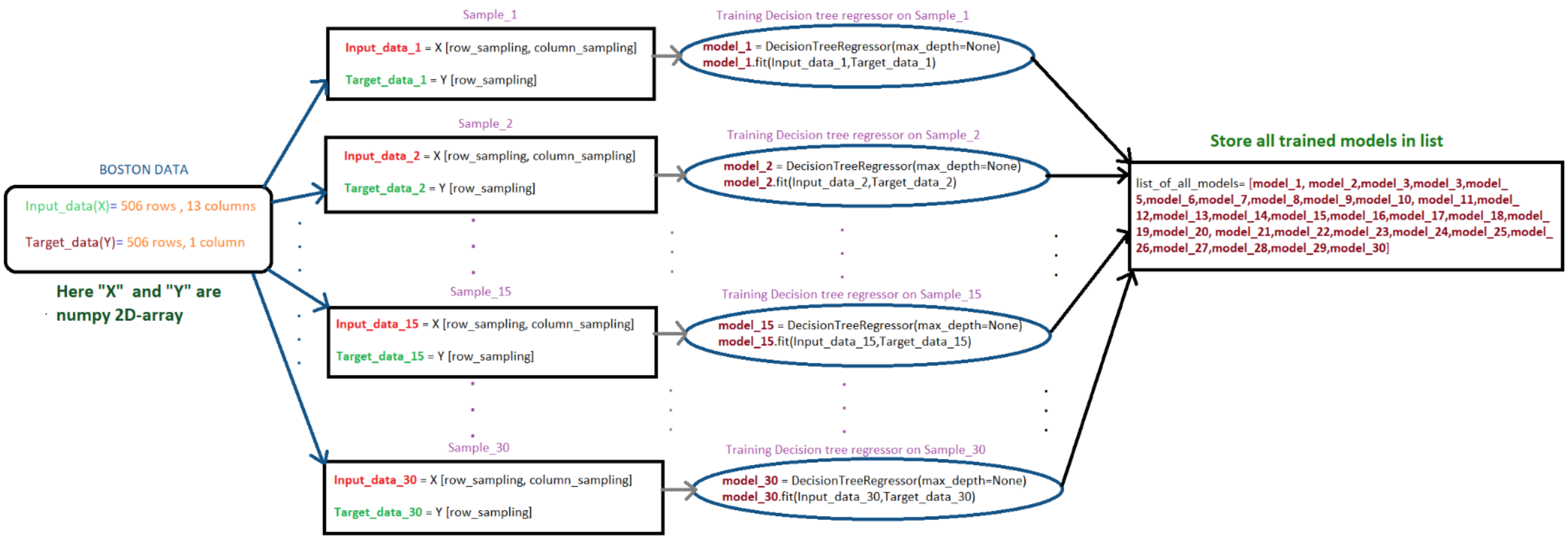
Grader function - 2

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

```
True
```

Step - 2

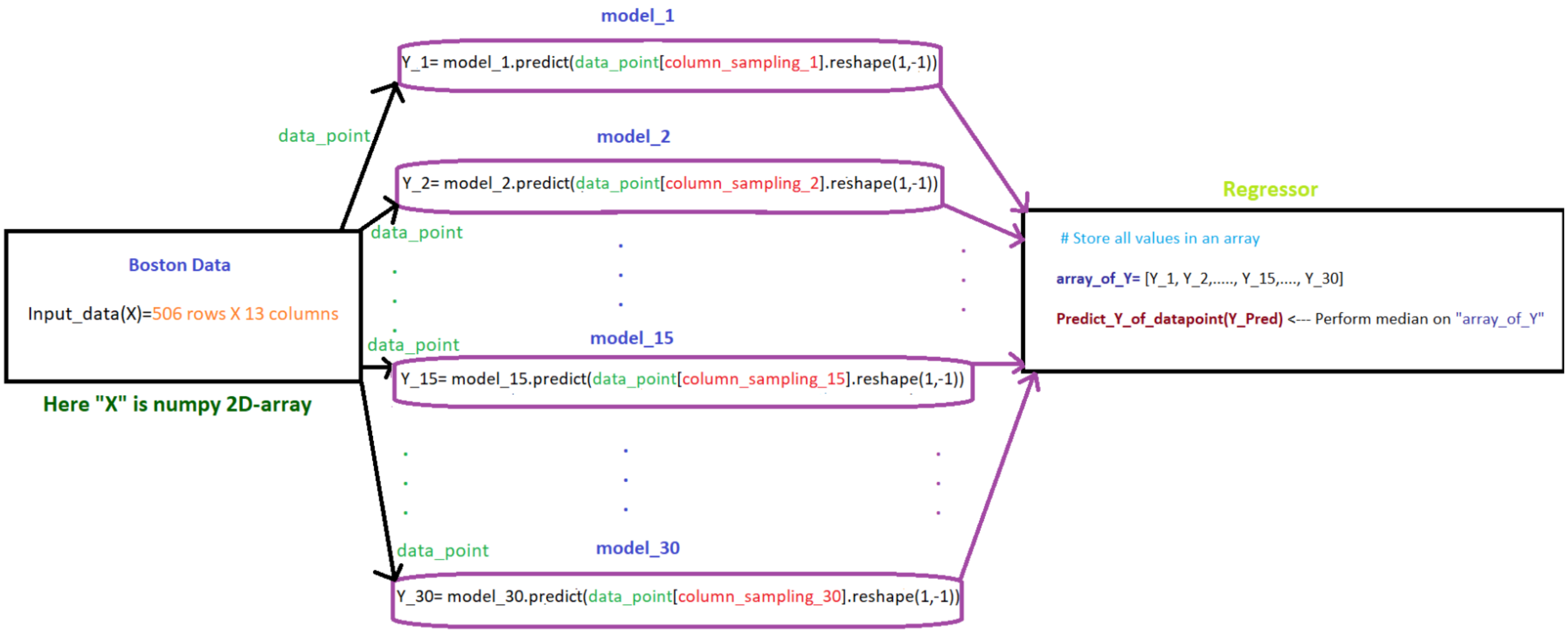
Flowchart for building tree



- **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**

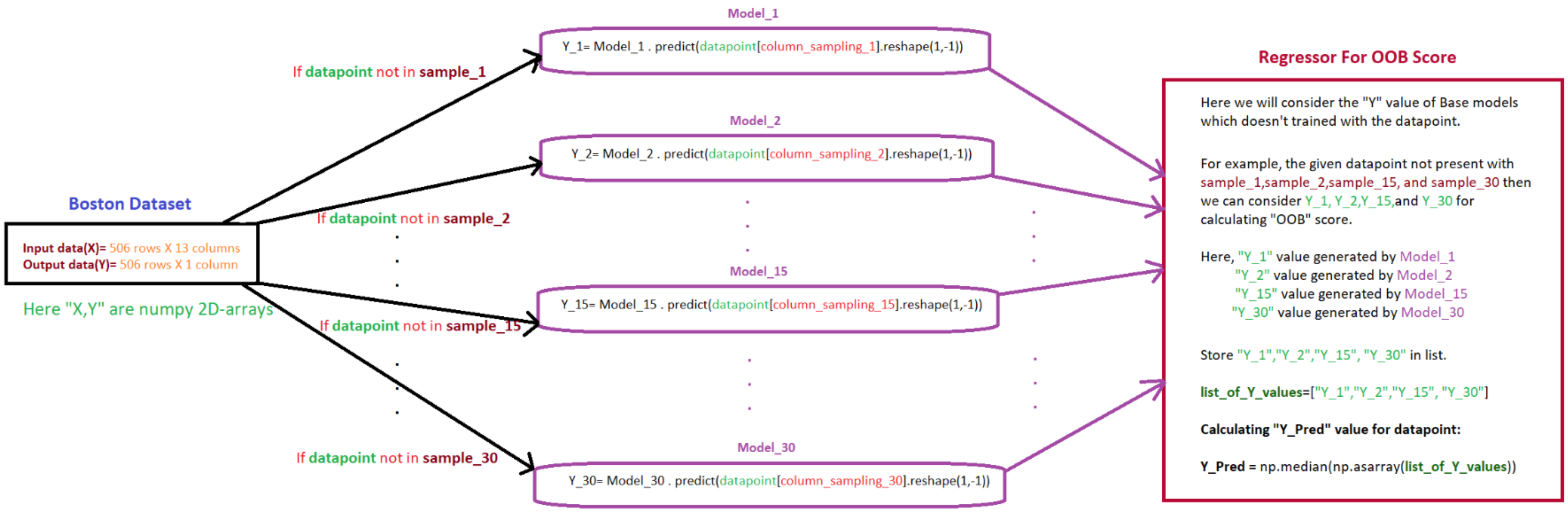
```
# ref : https://www.educative.io/answers/calculating-mean-squared-error-in-python
# ref : https://stackoverflow.com/questions/39064684/mean-squared-error-in-python
```

```
array_of_Y = []  # created the empty list

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_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)
median_predicted_y.shape

print("MSE : ", mean_squared_error(y, median_predicted_y ))
```

```
MSE :  0.39690914031620556
```

Step - 3

Flowchart for calculating OOB score



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

- **Write code for calculating OOB score**

```
y_predicted_oob_median_list = []

for i in range(0, 506):  # declared the range of oob from 0 to 506
  indices_for_oob_models = []  # created the empty variable
```

```
    # For each of i-th row I shall build a list, of sample size 30
    # ONLY condition being that this 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 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)

      # Now extract ONLY those specific columns/featues that were selected during the bootstrapping
      x_oob_data_point = [row_oob[columns] for columns 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)



  # here we are culculating the oob score from number of rows
  def calculate_out_of_bag_score(num_rows):

    oob_score = 0  # here we are intiating frm=om zero

    for i in range(0, num_rows): # taking range from zero to number of rows
```

```
      oob_score += ((y[i] - y_predicted_oob_median_list[i] ) ** 2)  # here we are calculating the oob score by incrementing the value


    final_oob_score = oob_score/506
    return final_oob_score

print("final_oob_score is ", calculate_out_of_bag_score(506))
```

```
final_oob_score is  14.462789767461278
```

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])
```

```
  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 = 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)

# And now the final MSE
MSE = mean_squared_error(y, median_predicted_y )

# here we will calculate the oob median score
y_predicted_oob_median_list = []

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

  # for each row we need to make sample sixe of 30 as per instruction

  # 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 = []
```

```
    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):
    #  refer :  oob_score = (oob_score + (y[i] - y_predicted_oob_median_list[i] ) ** 2)

    oob_score += (y[i] - y_predicted_oob_median_list[i] ) ** 2

  final_oob_score = oob_score/506

  return MSE, final_oob_score

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

```
(0.06095355731225299, 16.54137969367589)
```

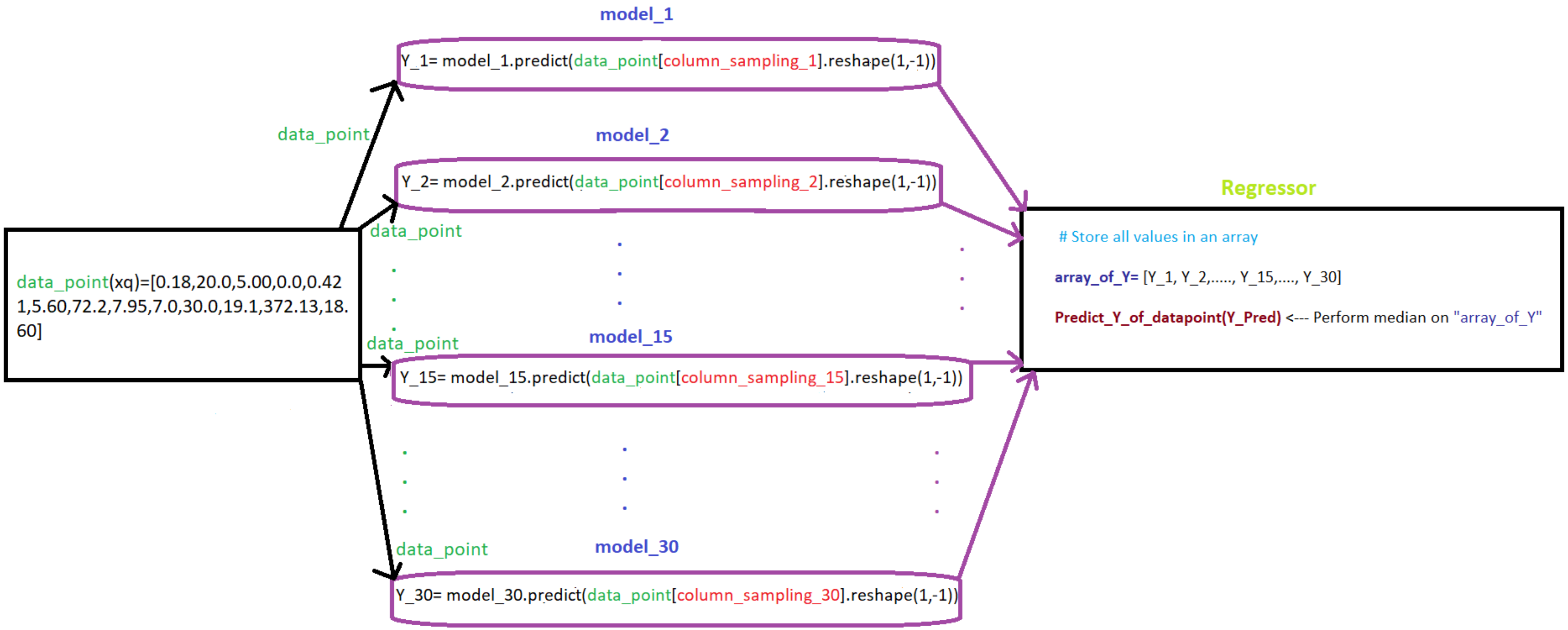
```
import scipy

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)


# Per document - https://www.kite.com/python/answers/how-to-compute-the-confidence-interval-of-a-sample-statistic-in-python
# confidence_interval = scipy.stats.t.interval(confidence_level, degrees_freedom, sample_mean, sample_standard_error)
confidence_interval_mse_35 = scipy.stats.t.interval(confidence_level, degrees_of_freedom, mean_of_sample_mse_35, standard_error_of_sa
print("confidence_interval_mse_35 ", 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_freedom, mean_of_sample_oob_score_35, standard
print("confidence_interval_oob_score_35 ", confidence_interval_oob_score_35)
```

```
confidence_interval_mse_35  (0.09020789646549911, 0.15168791935432868)
confidence_interval_oob_score_35  (13.039539604393621, 14.451392285356814)
```

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_columns
    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)

    # here we are predicting the y for quesry point xq from all base_learners
    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

# here from the given set of sample of query print xq
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]

# here as per instruction predicting the y for given xq
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

TASK 1

1) first we sampled the datapoint and features a per instruction randomly .

2) after concating of datapoint we calculated the mean sqaured error and MSE we got .04% between actual error and predicting error .

3) means our model performing really good not a much big difference between actual and predicted error

TASK 2

A): observation

1: the oob score we got 14%

2: that really god because that's mean 14% dataset we got as validation data .

3: from these dataset we can conclude that how are model will perform on test data

(B) : observation

1) final oob score we got 13% with .03% mse score not a big dfference model will perform will good on test data

(c) : observation

1) mse score on confidence interval of 35 is 0.06 , .16% difference is less than 10 so output is as expected

2) both the oob score got 13% and 14% dataset we got for validation perpose to make_sure the performence of test data

TASK 3

1) the predictedY we got for query point xq is 18.7

2) 18.7 value we have got for given sample of array of 30 xq datapoints