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
from sklearn.metrics import mean_squared_error
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
import matplotlib.pyplot as plt
```

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, 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.**

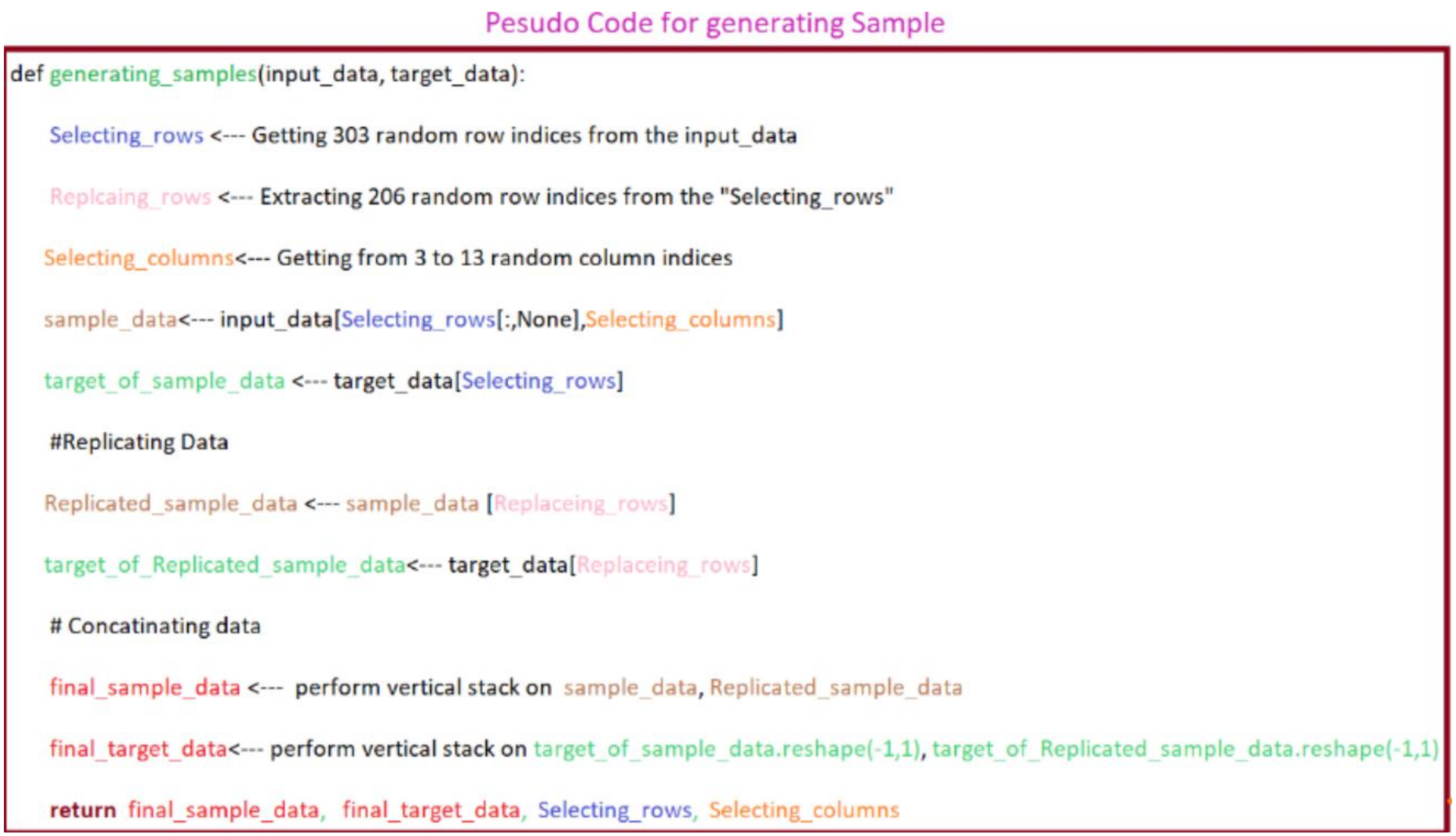
Consider xq= [0.18,20.0,5.00,0.0,0.421,5.60,72.2,7.95,7.0,30.0,19.1,372.13,18.60] Predict the house price for this point as mentioned in the step 2 of Task 1.

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**

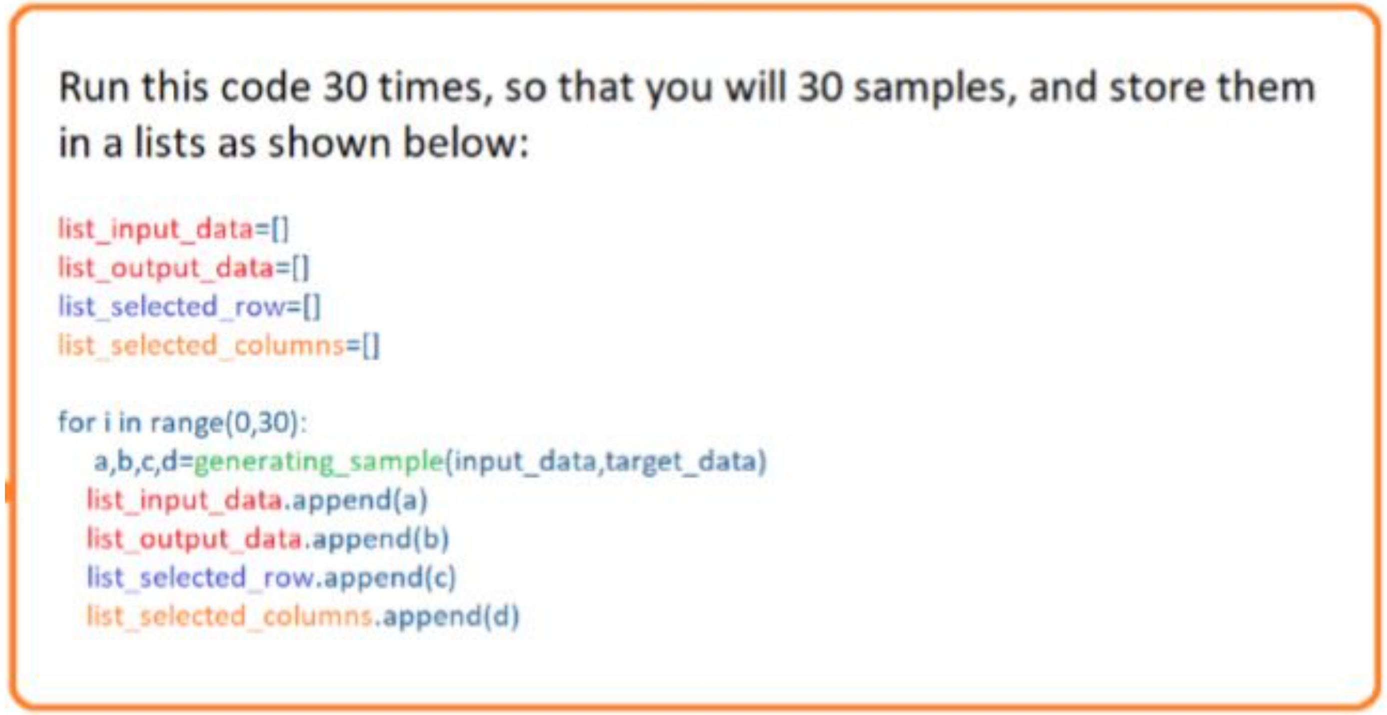
In [5]:
```
def generating_samples(input_data, target_data):

    '''In this function, we will write code for generating 30 samples '''

    sel_rows = np.random.choice(len(input_data), int(0.6*len(input_data)), replace=False)
    rep_rows = np.random.choice(sel_rows, 203, replace=False)
    sel_cols = np.random.choice(13, np.random.randint(3,12), replace=False)

    sampled_rows = np.concatenate((sel_rows, rep_rows))
    input_col_sampling = input_data[:, sel_cols]

    final_sample_data = input_col_sampling[sampled_rows]
    final_target_data = target_data[sampled_rows]

    return list(final_sample_data), list(final_target_data), list(sel_rows), list(sel_cols)

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

Grader function - 1

In [7]:
```
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)  ## subtracts
    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[7]:
```
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 [8]:
```
# Use generating_samples function to create 30 samples
# store these created samples in a list
def generate_30_samples(x,y):
    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)

    return list_input_data, list_output_data, list_selected_row, list_selected_columns
```

In [9]:
```
list_input_data, list_output_data, list_selected_row, list_selected_columns = generate_30_samples(x,y)
print(list_selected_columns)
```

```
[[7, 8, 12, 0, 9], [1, 8, 11, 3, 12, 4, 6, 9, 0], [7, 12, 1, 6, 0, 8, 11, 5, 3, 2], [11, 2, 8, 0, 10, 7
, 3, 4, 1, 12, 9], [6, 1, 5], [3, 7, 12, 0, 10, 1, 5, 4, 9], [6, 2, 4, 1, 9], [2, 9, 0, 11, 6, 3, 4, 5,
10, 7], [5, 10, 7, 8, 12], [7, 12, 10, 1, 6, 0, 2, 9, 3, 5], [11, 3, 7, 9, 8, 4, 1, 2, 12], [1, 0, 8, 1
2], [12, 8, 7, 11, 0, 10, 6, 4, 3, 1], [5, 4, 2, 11], [4, 8, 1, 0, 12], [2, 4, 1, 5, 3, 10, 9], [2, 1,
6, 11, 7, 10, 4, 0, 5, 8, 3], [11, 4, 6, 8, 12, 3], [8, 2, 7, 6, 0, 4], [4, 2, 10, 3, 7, 11], [10, 4, 6
, 8], [1, 10, 9, 3, 12, 0, 8, 5], [4, 11, 12, 1, 8, 2, 3, 6, 7, 5, 10], [6, 0, 5, 11, 1, 12, 4, 2, 10],
[4, 3, 11, 9], [1, 5, 7, 11, 4, 12, 9, 8], [8, 7, 6, 12, 1, 0, 4], [7, 12, 10, 9, 3, 2, 6, 8, 4, 11], [
7, 8, 5, 10, 3, 2], [10, 6, 4, 0, 2, 9, 12, 3, 11, 8]]
```

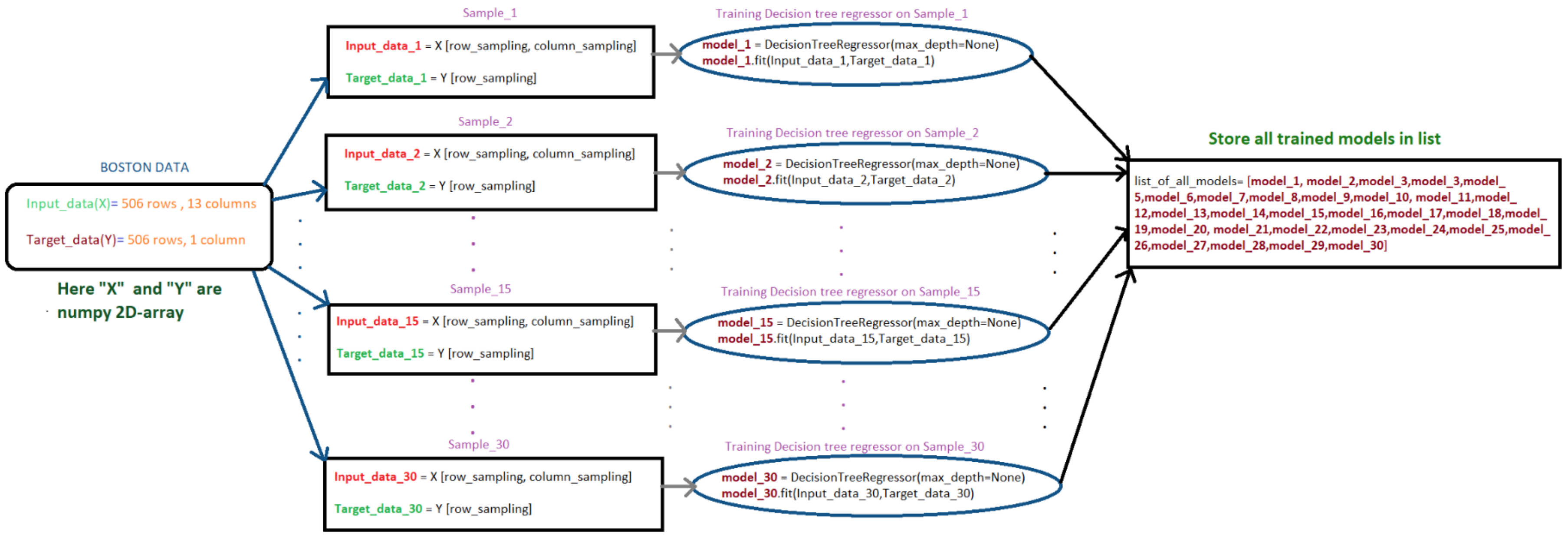
Grader function - 2

In [10]:
```
def grader_30(a):
    assert(len(a)==30 and len(a[0])==506)
    return True
grader_30(list_input_data)
```

Out[10]:
```
True
```

Step - 2

Flowchart for building tree


BOSTON DATA
Input_data(X)= 506 rows , 13 columns
Target_data(Y)= 506 rows, 1 column
Here "X" and "Y" are numpy 2D-array
Sample_1
Input_data_1 = X [row_sampling, column_sampling]
Target_data_1 = Y [row_sampling]
Sample_2
Input_data_2 = X [row_sampling, column_sampling]
Target_data_2 = Y [row_sampling]
Sample_15
Input_data_15 = X [row_sampling, column_sampling]
Target_data_15 = Y [row_sampling]
Sample_30
Input_data_30 = X [row_sampling, column_sampling]
Target_data_30 = Y [row_sampling]
Training Decision tree regressor on Sample_1
model_1 = DecisionTreeRegressor(max_depth=None)
model_1.fit(Input_data_1,Target_data_1)
Training Decision tree regressor on Sample_2
model_2 = DecisionTreeRegressor(max_depth=None)
model_2.fit(Input_data_2,Target_data_2)
Training Decision tree regressor on Sample_15
model_15 = DecisionTreeRegressor(max_depth=None)
model_15.fit(Input_data_15,Target_data_15)
Training Decision tree regressor on Sample_30
model_30 = DecisionTreeRegressor(max_depth=None)
model_30.fit(Input_data_30,Target_data_30)
Store all trained models in list
list_of_all_models= [model_1, model_2,model_3,model_3,model_5,model_6,model_7,model_8,model_9,model_10, model_11,model_12,model_13,model_14,model_15,model_16,model_17,model_18,model_19,model_20, model_21,model_22,model_23,model_24,model_25,model_26,model_27,model_28,model_29,model_30]

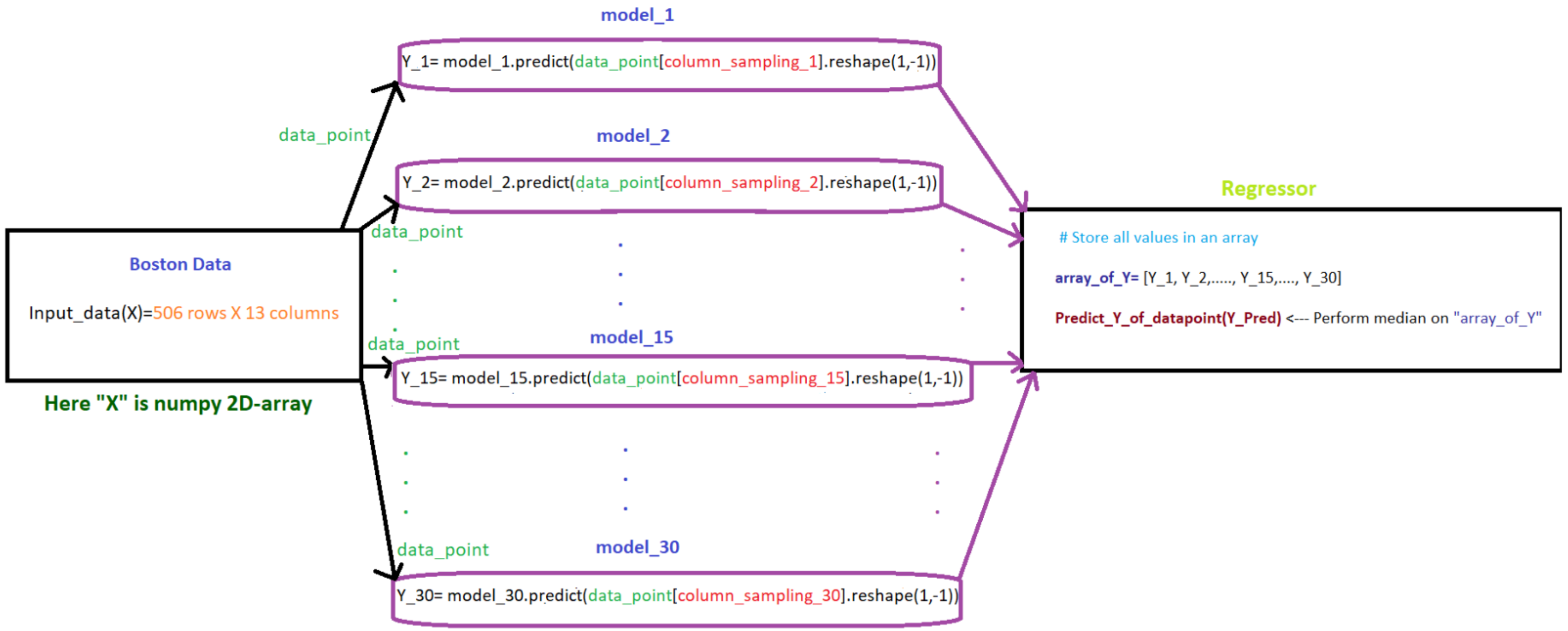
- **Write code for building regression trees**

In [11]:
```
def build_regression_tree(input, output):
    # data_point = x[:, ]
    list_of_all_models = []

    for i in range(0,30):
      model = DecisionTreeRegressor(max_depth=None)
      model.fit(input[i], output[i])
      list_of_all_models.append(model)

    return list_of_all_models
```

In [12]:
```
list_of_all_models = build_regression_tree(list_input_data, list_output_data)
print(len(list_of_all_models))
```

```
30
```

Flowchart for calculating MSE


model_1
Y_1= model_1.predict(data_point[column_sampling_1].reshape(1,-1))
data_point
model_2
Y_2= model_2.predict(data_point[column_sampling_2].reshape(1,-1))
data_point
Boston Data
Input_data(X)=506 rows X 13 columns
Here "X" is numpy 2D-array
data_point
model_15
Y_15= model_15.predict(data_point[column_sampling_15].reshape(1,-1))
data_point
model_30
Y_30= model_30.predict(data_point[column_sampling_30].reshape(1,-1))
Regressor
# Store all values in an array
array_of_Y= [Y_1, Y_2,....., Y_15,...., Y_30]
Predict_Y_of_datapoint(Y_Pred) <--- Perform median on "array_of_Y"

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 [13]:
```
def predict_y(list_selected_columns, list_of_all_models):
    array_of_Y = np.zeros((506,30))
    predicted_y = []
    x_points = []
    for i in range(0, 506):
        for j in range(0,30):
            cols = list_selected_columns[j]
            datapoint = x[i][cols].reshape(1,-1)
            model = list_of_all_models[j]
            y_pred = model.predict(datapoint)
            array_of_Y[i][j] = y_pred
        predicted_y.append(np.average(np.asarray(array_of_Y[i])))

    narr = np.array(array_of_Y)
    return predicted_y, array_of_Y
    #datapoint = x[10, cols].reshape(1,-1)
```

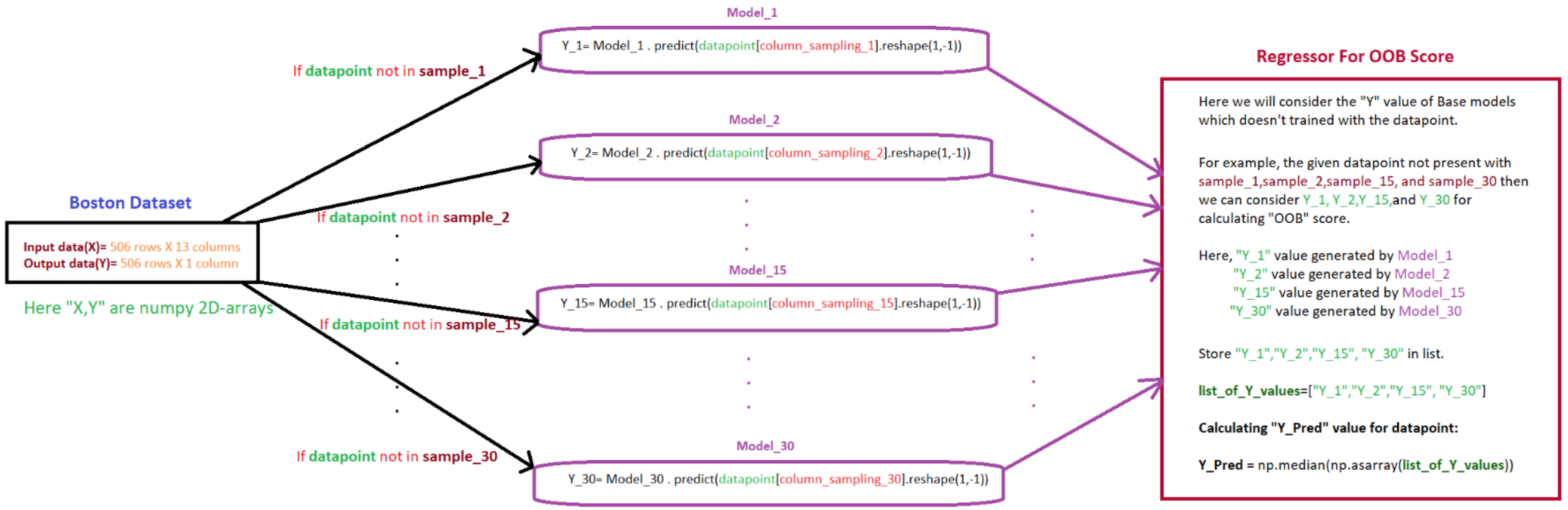
In [14]:
```
predicted_y, array_of_Y = predict_y(list_selected_columns, list_of_all_models)
```

In [15]:
```
mean_squared_error(y, predicted_y)
```

Out[15]:
```
2.199879038392526
```

Step - 3

Flowchart for calculating OOB score


Boston Dataset
Input data(X)= 506 rows X 13 columns
Output data(Y)= 506 rows X 1 column
Here "X,Y" are numpy 2D-arrays
If datapoint not in sample_1
If datapoint not in sample_2
If datapoint not in sample_15
If datapoint not in sample_30
Model_1
Model_2
Model_15
Model_30
Regressor For OOB Score

```
Y_1= Model_1 . predict(datapoint[column_sampling_1].reshape(1,-1))
Y_2= Model_2 . predict(datapoint[column_sampling_2].reshape(1,-1))
Y_15= Model_15 . predict(datapoint[column_sampling_15].reshape(1,-1))
Y_30= Model_30 . predict(datapoint[column_sampling_30].reshape(1,-1))
```

Here we will consider the "Y" value of Base models which doesn't trained with the datapoint.

For example, the given datapoint not present with sample_1,sample_2,sample_15, and sample_30 then we can consider Y_1, Y_2,Y_15,and Y_30 for calculating "OOB" score.

Here, "Y_1" value generated by Model_1
"Y_2" value generated by Model_2
"Y_15" value generated by Model_15
"Y_30" value generated by Model_30

Store "Y_1","Y_2","Y_15", "Y_30" in list.

```
list_of_Y_values=["Y_1","Y_2","Y_15", "Y_30"]
```

Calculating "Y_Pred" value for datapoint:

```
Y_Pred = np.median(np.asarray(list_of_Y_values))
```

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

- **Write code for calculating OOB score**

In [16]:
```
def get_Y_pred(list_selected_columns, list_of_all_models, list_input_data):
    list_of_y_values = []
    x_points = []
    for i in range(0, 506):
        y_pred_list = []
        for j in range(0, 30):
            cols = list_selected_columns[j]
            datapoint = x[i][cols].reshape(1,-1)
            flag = "not present"
            for a in list_input_data[j]:
              comparison = datapoint[0] == a
              equal_arrays = comparison.all()
              if(equal_arrays):
                flag = "present"
            if(flag == "not present"):
              model = list_of_all_models[j]
              y_pred = model.predict(datapoint)
              y_pred_list.append(y_pred)
        if y_pred_list:
          med = np.median(y_pred_list)
          list_of_y_values.append(med)
    return list_of_y_values
```

In [17]:
```
def compute_oob(y, y_pred):
    sum = 0
    for i in range(0, len(y)):
        sum += np.square(y[i] - y_pred[i])

    oobscore = (1/len(y)) * sum
    return oobscore
```

In [18]:
```
list_of_y_values = get_Y_pred(list_selected_columns, list_of_all_models, list_input_data)
oobscore = compute_oob(y, list_of_y_values)
print(oobscore)
```

```
12.61443401405357
```

Task 2

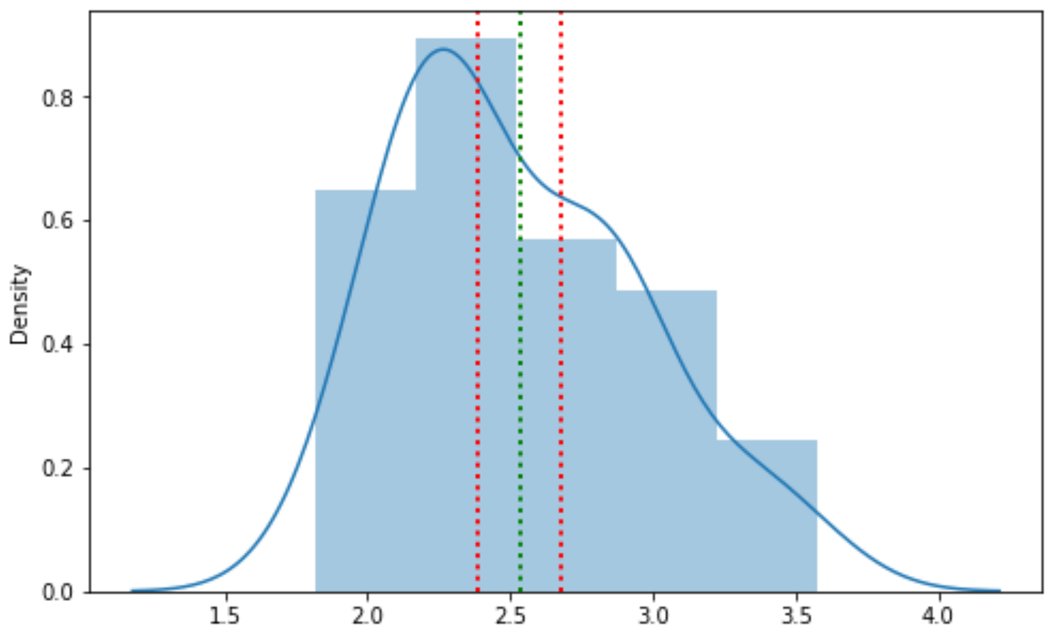
In [19]:
```
mse_35 = []
oob_35 = []

for i in range(0, 35):
    list_input_data_35, list_output_data_35, list_selected_row_35, list_selected_columns_35 = generate_3
    list_of_all_models_35 = build_regression_tree(list_input_data_35, list_output_data_35)
    predicted_y, array_of_Y = predict_y(list_selected_columns_35, list_of_all_models_35)
    mse = mean_squared_error(y, predicted_y)
    list_of_y_values_35 = get_Y_pred(list_selected_columns_35, list_of_all_models_35, list_input_data_35
    oobscore = compute_oob(y, list_of_y_values_35)
    mse_35.append(mse)
    oob_35.append(oobscore)
```

In [20]:
```
print(mse_35)
print(oob_35)
```

```
[2.1547204263417545, 2.711813157148423, 2.90561369427763, 2.8917761550164913, 2.346214526404097, 1.8168
066315327185, 3.375640403014113, 2.09143662515859, 3.5732134316768205, 1.9874128076477335, 2.4347333775
928175, 2.8152016019919324, 2.8946181705532625, 2.308452085485981, 2.011703150466012, 2.957566300308189
3, 2.884860730289338, 2.161006405627202, 2.654602610907457, 2.8092001587351083, 3.1868047010867437, 2.2
65719545698531, 2.2283966403162045, 2.3188868540774865, 2.8014401849813777, 2.321022792523589, 2.324546
415852177, 2.5547741234811885, 2.1773461954634246, 3.3224225169792465, 2.0282587066543343, 2.5446187638
77642, 2.136250916198737, 2.251861335797091, 2.4160083633240688]
[14.457697628458494, 15.622062472551596, 15.372937423144496, 19.416014629995598, 15.890523852656989, 10
.984925889328053, 15.122248572683354, 11.626037549407108, 18.46641414141413, 14.044640563241106, 14.557
964975845408, 15.394581274703556, 16.56861482213439, 14.525815217391296, 12.732129446640311, 13.8183564
18667598, 15.296516798418962, 14.510167984189707, 14.913837834870424, 19.376882411067193, 18.2077673473
86913, 13.45850790513833, 13.812371541501978, 12.31249066754502, 14.827291392182696, 13.51091681077075,
14.817940272288094, 16.1739476284585, 14.151744861660076, 16.53228874264993, 12.28227272727273, 13.1965
0490037912, 12.74126482213438, 13.205454545454542, 12.078468379446635]
```

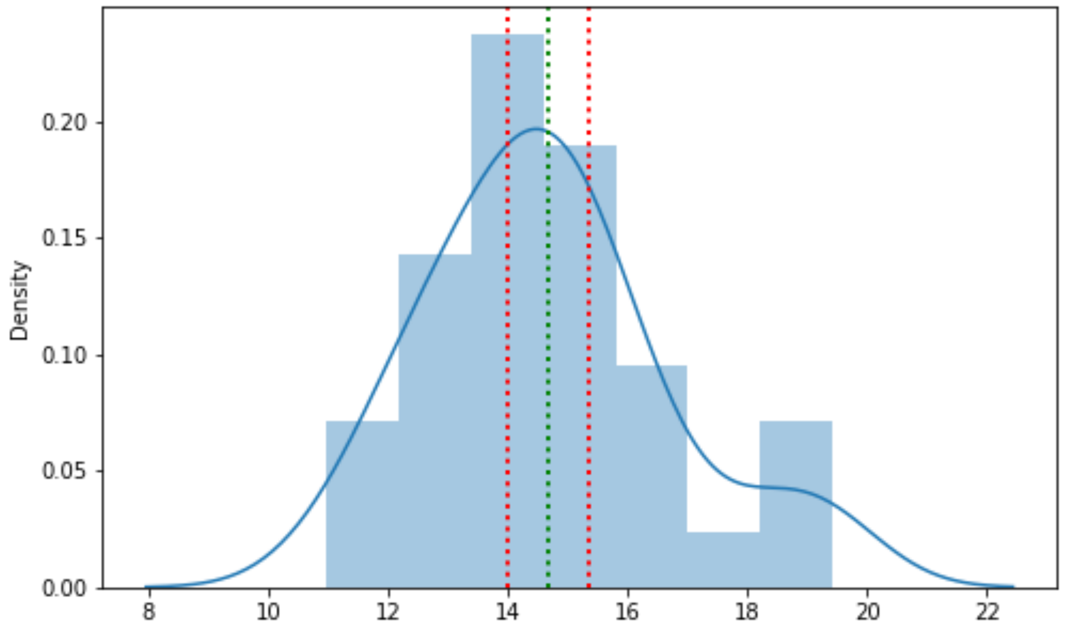
In [21]:
```
mean = np.mean(mse_35)
sample_std = np.std(mse_35)
SE = sample_std/(np.sqrt(len(mse_35)))
upper = mean + (SE*2)
lower = mean - (SE*2)
print(upper)
print(lower)

fig, axs = plt.subplots(figsize=(8, 5))
sns.distplot(mse_35, ax=axs)
axs.axvline(mean, linestyle=":", color='g', label="sample_mean", lw=2)
axs.axvline(upper, linestyle=":", color='r', label="s_mean+2*SE", lw=2)
axs.axvline(lower, linestyle=":", color='r', label="s_mean-2*SE", lw=2)
plt.show()
```

```
2.677891752942009
2.3886768474287052
```



In [22]:
```
print(oob_35)
mean = np.mean(oob_35)
sample_std = np.std(oob_35)
SE = sample_std/(np.sqrt(len(oob_35)))
upper = mean + (SE*2)
lower = mean - (SE*2)
print(upper)
print(lower)

fig1, axs = plt.subplots(figsize=(8, 5))
sns.distplot(oob_35, ax=axs)
axs.axvline(mean, linestyle=":", color='g', label="sample_mean", lw=2)
axs.axvline(upper, linestyle=":", color='r', label="s_mean+2*SE", lw=2)
axs.axvline(lower, linestyle=":", color='r', label="s_mean-2*SE", lw=2)
plt.show()
```

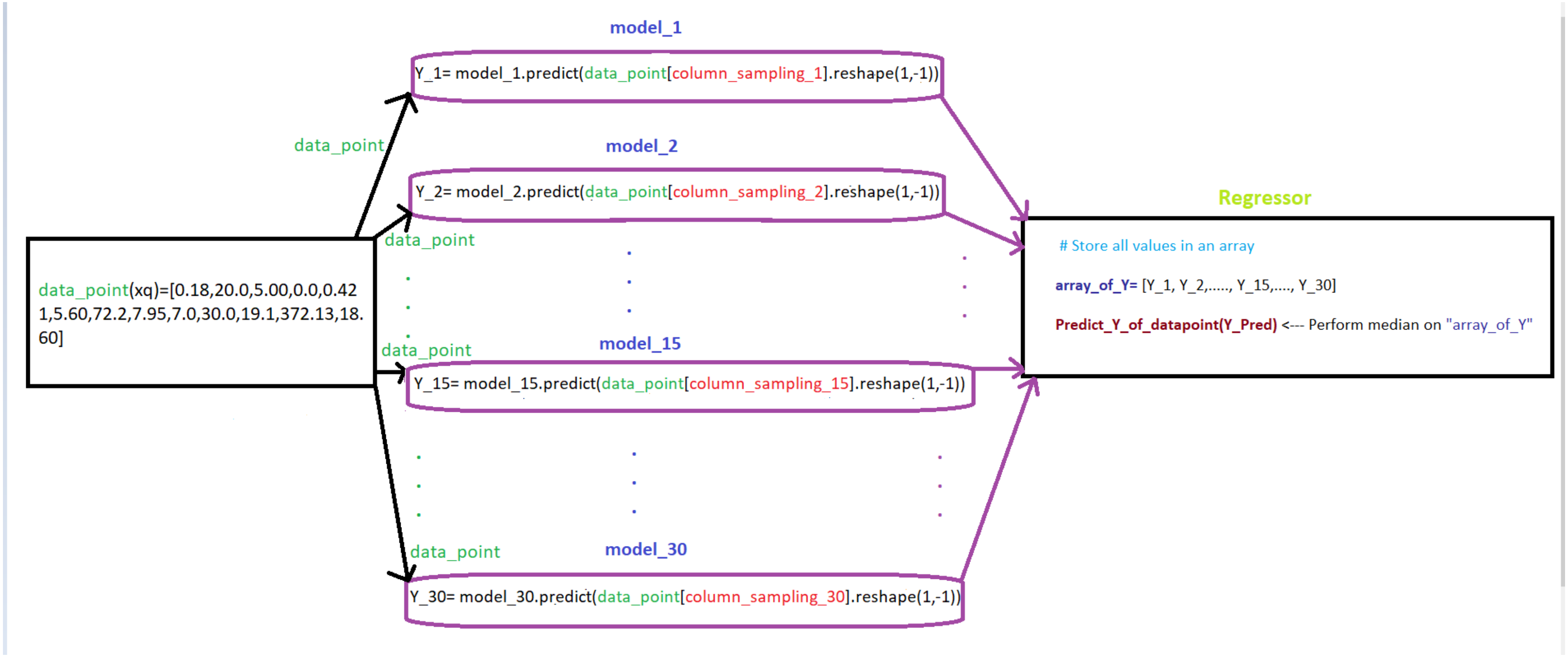
```
[14.457697628458494, 15.622062472551596, 15.372937423144496, 19.416014629995598, 15.890523852656989, 10
.984925889328053, 15.122248572683354, 11.626037549407108, 18.46641414141413, 14.044640563241106, 14.557
964975845408, 15.394581274703556, 16.56861482213439, 14.525815217391296, 12.732129446640311, 13.8183564
18667598, 15.296516798418962, 14.510167984189707, 14.913837834870424, 19.376882411067193, 18.2077673473
86913, 13.45850790513833, 13.812371541501978, 12.31249066754502, 14.827291392182696, 13.51091681077075,
14.817940272288094, 16.1739476284585, 14.151744861660076, 16.53228874264993, 12.28227272727273, 13.1965
0490037912, 12.74126482213438, 13.205454545454542, 12.078468379446635]
15.368543160809601
14.00331983639494
```



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.



data_point(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]
data_point
model_1
Y_1= model_1.predict(data_point[column_sampling_1].reshape(1,-1))
model_2
Y_2= model_2.predict(data_point[column_sampling_2].reshape(1,-1))
model_15
Y_15= model_15.predict(data_point[column_sampling_15].reshape(1,-1))
model_30
Y_30= model_30.predict(data_point[column_sampling_30].reshape(1,-1))
Regressor
# Store all values in an array
array_of_Y= [Y_1, Y_2,....., Y_15,...., Y_30]
Predict_Y_of_datapoint(Y_Pred) <--- Perform median on "array_of_Y"

- **Write code for TASK 3**

In [23]:
```
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])
yq_list = []
for j in range(0,30):
    cols = list_selected_columns[j]
    datapoint = xq[cols].reshape(1,-1)
    model = list_of_all_models[j]
    y_pred = model.predict(datapoint)
    yq_list.append(y_pred)
yq = np.average(np.asarray(yq_list))
print(yq)
```

```
20.363333333333337
```

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

Observation

- The samples we created have 303 random datapoints and other 203 datapoints are repeated from these 303 points.
- 30 such samples are created.
- Mean squared error is 2.199
- Out of Bag error is 12.61
- Based on task 2, it can be said that the confidence interval of both the distributions is almost equally spread.
- The oob score distribution is more normally distributed than the mse score distribution.
- The predicted value of given query datapoint is 20.36