# 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.</b>**

## Importing packages

In [51]:
```
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

# External Imports
import pandas as pd
import random
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from math import sqrt
```

In [114…
```
boston = load_boston()
x=boston.data   # independent variables
y=boston.target # target variable
```

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

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

In [4]:
```
x[:1]
```

Out[4]:
```
array([[6.320e-03, 1.800e+01, 2.310e+00, 0.000e+00, 5.380e-01, 6.575e+00,
        6.520e+01, 4.090e+00, 1.000e+00, 2.960e+02, 1.530e+01, 3.969e+02,
        4.980e+00]])
```

In [6]:
```
x.ndim
```

Out[6]:
```
2
```

In [7]:
```
# Using Dataframe for better manipulation & visibility
x_df = pd.DataFrame(x)
x_df.head(3)
```

Out[7]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.00632 | 18.0 | 2.31 | 0.0 | 0.538 | 6.575 | 65.2 | 4.0900 | 1.0 | 296.0 | 15.3 | 396.90 | 4.98 |
| 1 | 0.02731 | 0.0 | 7.07 | 0.0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2.0 | 242.0 | 17.8 | 396.90 | 9.14 |
| 2 | 0.02729 | 0.0 | 7.07 | 0.0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2.0 | 242.0 | 17.8 | 392.83 | 4.03 |

In [8]:
```
y[:3]
```

Out[8]:
```
array([24. , 21.6, 34.7])
```

In [9]:
```
(x[[0, 2]] == [x[0], x[2]]).all()
```

Out[9]:
```
True
```

In [10]:
```
x[0, [1, 2]]
```

Out[10]:
```
array([18.  ,  2.31])
```

In [11]:
```
y[[1, 2]]
```

Out[11]:
```
array([21.6, 34.7])
```

In [12]:
```
# Cannot select Rows & Columns at same time
x[[1, 2, 1], [1, 2]]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [12], in <cell line: 2>()
      1 # Cannot select Rows & Columns at same time
----> 2 x[[1, 2, 1], [1, 2]]

IndexError: shape mismatch: indexing arrays could not be broadcast together with shapes (3,) (2,)
```

In [ ]:
```
# Tweak for above issue
#https://stackoverflow.com/questions/22927181/selecting-specific-rows-and-columns-from-numpy-array
```

```
x[np.ix_([1,2,1], [1, 2])]
```

In [14]:
```
x.shape
```

Out[14]:
```
(506, 13)
```

In [57]:
```
t = x[0]
t
```

Out[57]:
```
array([6.320e-03, 1.800e+01, 2.310e+00, 0.000e+00, 5.380e-01, 6.575e+00,
       6.520e+01, 4.090e+00, 1.000e+00, 2.960e+02, 1.530e+01, 3.969e+02,
       4.980e+00])
```

In [60]:
```
t[[1,2]]
```

Out[60]:
```
array([18.  ,  2.31])
```

# 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 $i^{th}$
  $i^{th}$
  data point $y^i_{pred} = \frac{1}{30}\sum_{k=1}^{30}$(predicted value of $x^i$ with $k^{th}$ model)
  $y^i_{pred} = \frac{1}{30}\sum_{k=1}^{30}(\text{predicted value of } x^i \text{ with } k^{th} \text{ model})$
- Now calculate the $MSE = \frac{1}{506}\sum_{i=1}^{506}(y^i - y^i_{pred})^2$
  $MSE = \frac{1}{506}\sum_{i=1}^{506}(y^i - y^i_{pred})^2$

### 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 </li>
  - 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
  </ol>

# 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

```
def generating_samples(input_data, target_data):
    Selecting_rows <--- Getting 303 random row indices from the input_data
    Replacing_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 [Replacing_rows ]
    target_of_Replicated_sample_data<--- target_of_sample_data[ Replacing_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

- Write code for generating samples

In [17]:
```
row, col = x.shape
```

In [18]:
```
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

    # NOTE: As target is real vals (continuous) so we can take any random sample
    #       (otherwise we may need Proportional Sampling as what done by pandas.DataFrame.sampl())

    # Sample (60% of total rows)
    r_idx1 = random.sample(range(row), 303)  # -> Selecting Rows
    # Replicated 40% of r_idx1 (ie sampled data)
    r_idx2 = random.choices(r_idx1, k=203)  # -> Replicating Rows (506-303)

    r_idxs = [*r_idx1, *r_idx2]  # Sample + Replicated

    n_cols = random.randint(3, 13) # How many cols to pick !
    c_idxs = random.sample(range(col), n_cols)

    # https://stackoverflow.com/questions/22927181/selecting-specific-rows-and-columns-from-numpy-array
    data_x = x[np.ix_(r_idxs, c_idxs)]
    data_y = y[r_idxs]

    # Please have a look at this link https://docs.scipy.org/doc/numpy-1.16.0/reference/generated/numpy.random.
    # Please follow above pseudo code for generating samples

    # return sampled_input_data , sampled_target_data,selected_rows,selected_columns
    #note please return as lists
    return data_x.tolist(), data_y.tolist(), r_idx1, c_idxs
```

Grader function - 1 </fongt>

In [19]:
```
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[19]:
```
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 [20]:
```
# 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)
```

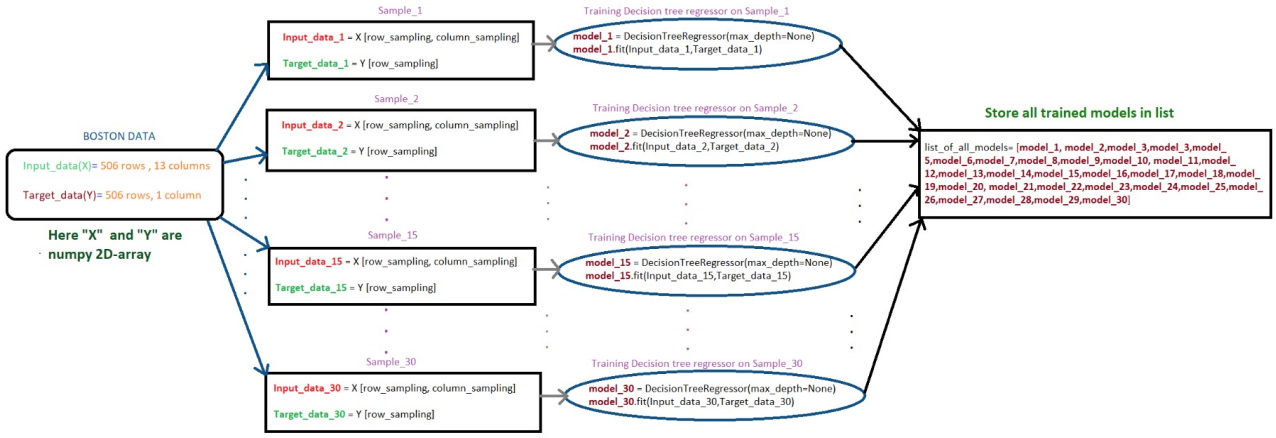
### Grader function - 2

In [21]:
```
def grader_30(a):
    assert(len(a)==30 and len(a[0])==506)
    return True
grader_30(list_input_data)
```

Out[21]:
```
True
```

## Step - 2

### Flowchart for building tree




- **Write code for building regression trees**

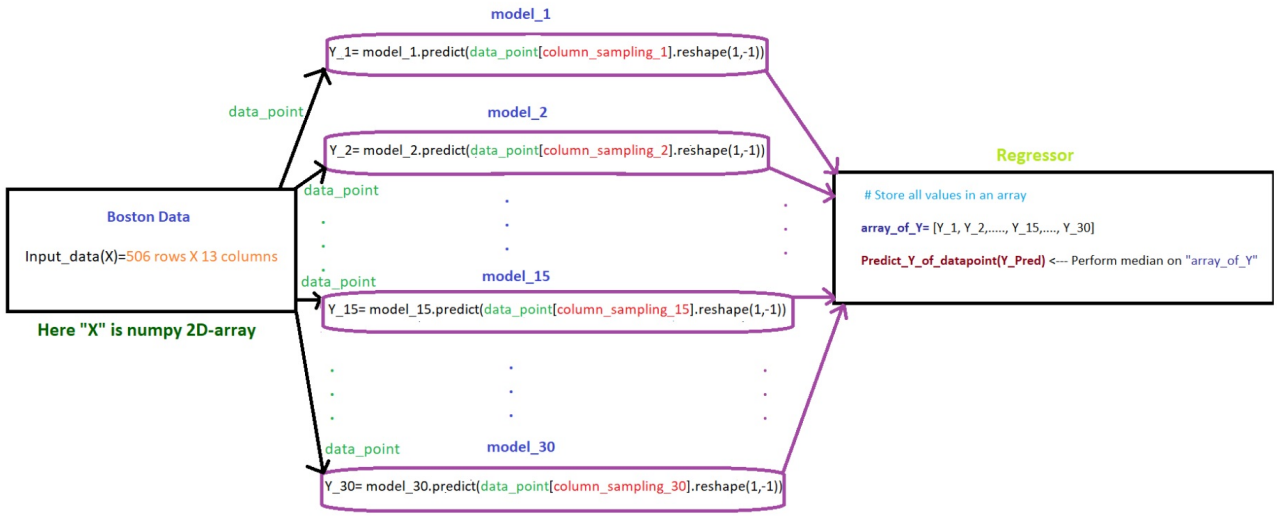
In [22]:
```
models = []

for i in range(30):
    model = DecisionTreeRegressor()
    sx, sy = list_input_data[i], list_output_data[i]
    model.fit(sx, sy)
    models.append(model)
```

In [23]:
```
len(models)
```

Out[23]:
```
30
```

### 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 [79]:
```
# Model Wise Prediction
preds = [] # preds of 506 data pts for Each Sample Wise
for i in range(30):
    model, cols = models[i], list_selected_columns[i]
    y_pred = model.predict(x[:, cols]).reshape(1, -1).ravel() # prediction of all 506 data pts
    preds.append(y_pred)

preds[0].shape
```

Out[79]:
```
(506,)
```

In [80]:
```
# Data Point Wise Prediction
preds2 = np.array(preds).T  # Inorder to get all the samples prediction in 1 row
```

In [81]:
```
preds2[0]
```

Out[81]:
```
array([24. , 33.3, 28.4, 24. , 24. , 22.9, 24. , 30.5, 24. , 24.4, 24. ,
       24. , 24. , 32.2, 24. , 24. , 24. , 32. , 24. , 24. , 32.7, 24. ,
       24. , 24. , 30.5, 24. , 24. , 29.8, 32.4, 24. ])
```

In [83]:
```
pred_y = [np.median(v) for v in preds2]
len(pred_y)
```

Out[83]:
```
506
```

In [84]:
```
mse = mean_squared_error(y, pred_y)
mse
```

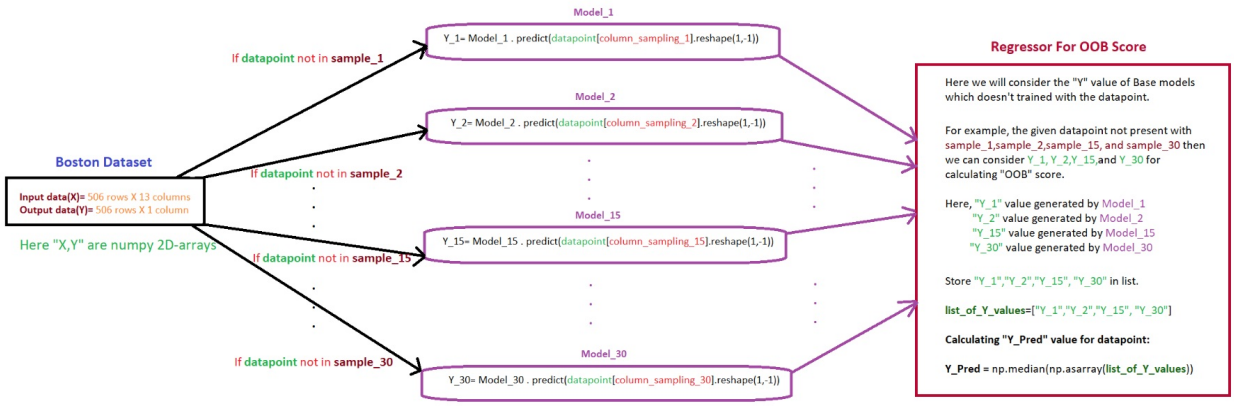
Out[84]:
```
0.1442019508350463
```

In [85]:
```
diff = y - pred_y
mse = (np.sum(diff**2))/506
mse
```

Out[85]:
```
0.1442019508350463
```

## 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 [86]:
```
# preds = [] # preds of 506 data pts for Each Sample Wise
# for i in range(30):
#     sel_idxs = set(list_selected_row[i]) # samples considered by this model
#     model, cols = models[i], list_selected_columns[i]
#     y_pred = model.predict(x[:, cols]).reshape(1, -1) # prediction of all 506 data pts
#     preds.append(y_pred)

# preds2 = np.array(preds).T  # Inorder to get all the samples prediction in 1 row
```

In [87]:
```
x[0]
```

Out[87]:
```
array([6.320e-03, 1.800e+01, 2.310e+00, 0.000e+00, 5.380e-01, 6.575e+00,
       6.520e+01, 4.090e+00, 1.000e+00, 2.960e+02, 1.530e+01, 3.969e+02,
       4.980e+00])
```

In [ ]:
```

```

In [88]:
```
pred_y = []
for i in range(506):
    pt = x[i]
    pred = []
    for j in range(30):
        sel_idxs = set(list_selected_row[j]) # samples considered by this model
        if i in sel_idxs: continue # Skip this model prediction as this model sees {i} point whist training
        model = models[j]
        cols = list_selected_columns[j]
        est = model.predict(pt[cols].reshape(1, -1)).reshape(1, -1)
        pred.append(est)

    if pred:
        pred_y.append(np.median(pred))
    else:
        pred_y.append(-1)
```

In [89]:
```
pred_y[:5]
```

Out[89]:
```
[30.5, 23.9, 29.85, 33.95, 33.4]
```

In [90]:
```
# pred_y = [np.median(m[0]) for m in preds2]
# len(pred_y)
```

In [91]:
```
oob = mean_squared_error(y, pred_y)
oob
```

Out[91]:
```
14.968870833985898
```

In [92]:
```
diff = y - pred_y
mse2 = (np.sum(diff**2))/506
mse2
```

Out[92]:
```
14.968870833985898
```

# Task 2

In [93]:
```
def get_scores(x, y):
    '''Return MSE & OOB after DT Regressor application on Data := x,y '''

    mse, oob = -1, -1

    # 1. Prepare 30 Samples
    # 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):
        # 1. generate sample
        g_x, g_y, i_r, i_c = generating_samples(x, y)
        list_input_data.append(g_x)
        list_output_data.append(g_y)
        list_selected_row.append(i_r)
        list_selected_columns.append(i_c)

    # 2. Get Model for 30 Samples
    models = []
    for i in range(30):
        model = DecisionTreeRegressor()
        sx, sy = list_input_data[i], list_output_data[i]
        model.fit(sx, sy)
        models.append(model)

    # 3. Prediction From all samples
    preds = [] # preds of 506 data pts for Each Sample Wise
    for i in range(30):
        model, cols = models[i], list_selected_columns[i]
        y_pred = model.predict(x[:, cols]).reshape(1, -1) # prediction of all 506 data pts
        preds.append(y_pred)

    preds2 = np.array(preds).T  # Inorder to get all the samples prediction in 1 row
```

```
    pred_y = [np.median(m[0]) for m in preds2]

    mse = mean_squared_error(y, pred_y)

    # 4. Prediction for selective samples
    pred_y = []
    for i in range(506):
        pt = x[i]
        pred = []
        for j in range(30):
            sel_idxs = set(list_selected_row[j]) # samples considered by this model
            if i in sel_idxs: continue # Skip this model prediction as this model sees {i} point whist training
            model = models[j]
            cols = list_selected_columns[j]
            est = model.predict(pt[cols].reshape(1, -1)).reshape(1, -1)
            pred.append(est)

        if pred:
            pred_y.append(np.median(pred))
        else:
            pred_y.append(-1)

    oob = mean_squared_error(y, pred_y)

    return mse, oob
```

In [94]:
```
mse_scores = []
oob_scores = []
for i in range(35):
    mse, oob = get_scores(x, y)
    mse_scores.append(mse)
    oob_scores.append(oob)

mses = np.array(mse_scores)
oobs = np.array(oob_scores)
```

In [95]:
```
def calc_ci(data):
    '''Confidence Interval Computation'''
    size=len(data)
    avg, stdev = data.mean(), data.std()
    left_lim  = round(avg - 2*(stdev/sqrt(size)), 2)
    right_lim = round(avg + 2*(stdev/sqrt(size)), 2)
    return left_lim, right_lim
```

In [96]:
```
le, ri = calc_ci(mses)
print(f'CI of MSE: [{le}, {ri}]')
```

```
CI of MSE: [0.07, 0.12]
```

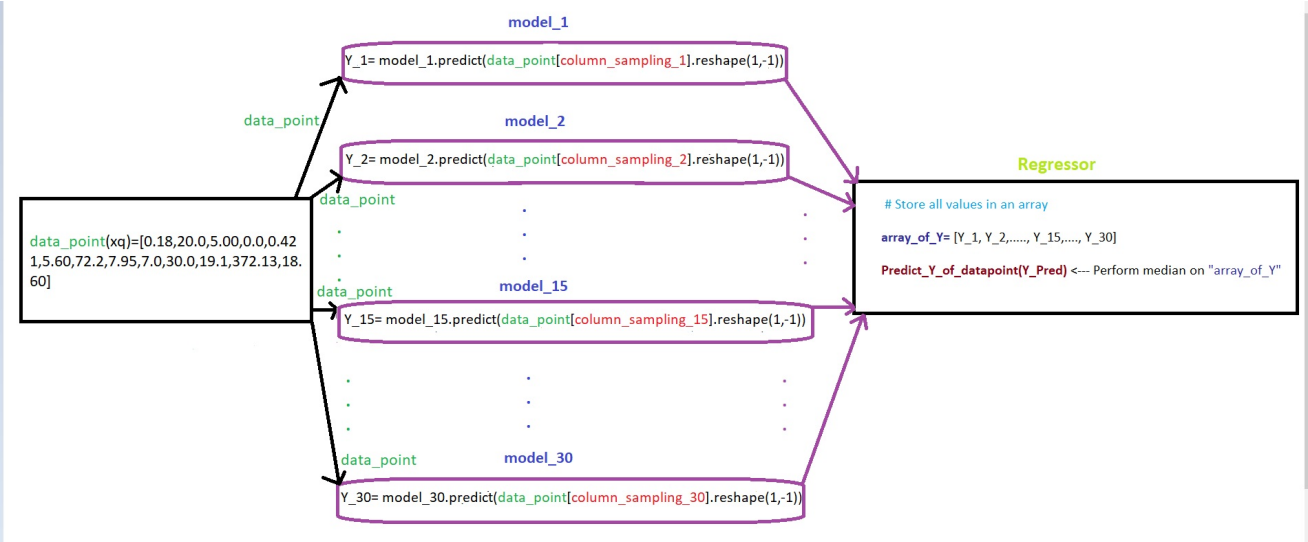
In [97]:
```
le, ri = calc_ci(oobs)
print(f'CI of OOBS: [{le}, {ri}]')
```

```
CI of OOBS: [13.48, 14.78]
```

# 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 [111…
```
def predict(xq):
    '''Predict the target(y) for given(xq) via 30 trained bagged models + Regressor(median)'''
    preds = [] # preds across all 30 models
    for i in range(30):
        model, cols = models[i], list_selected_columns[i]
        est = model.predict(xq[cols].reshape(1,-1)) # prediction of Xq point
        preds.append(round(est.item(),3))
    return np.median(preds)
```

In [112…
```
# Data Point
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])

y_pred = predict(xq)
y_pred
```

Out[112]:
```
18.5
```

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

**MSE when considering all the models, gives lower value; but when MSE is considered for OOB, its giving higher value comparatively**

**Thus when all the training models are considered its working fine but when only undiscovered model are considered then its not working good comparatively. Hence it seems that individual model has some tendency to overfit to some collection of points, Which is mitigated when considered collective model consideration**

In [ ]:
```
```