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.metrics import median_absolute_error
import random
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from tqdm import tqdm
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

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

In [3]:

```
x.shape[0]
```

Out[3]:

```
506
```

In [4]:

```
# from sklearn.ensemble import RandomForestRegressor
# regr = RandomForestRegressor(max_depth=None, random_state=0,oob_score = True)
# regr.fit(x, y)
```

In [5]:

```
print (x[:2])
print (y[:2])
```

```
[[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]]
[24.  21.6]
```

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.

SOLUTIONS

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 [25]:

```
def generating_samples(input_data, target_data):

    '''In this function, we will write code for generating 30 samples '''
    len_i_row = input_data.shape[0]
    len_i_col = input_data.shape[1]

    selecting_rows = np.random.choice(len_i_row,303,replace = False)
    selecting_rows.sort()
```

```
    replicating_rows = np.random.choice(selecting_rows,203)
    replicating_rows.sort()

    random_int = random.randint(3, len_i_col-3)
    selecting_columns = random.sample(range(3, len_i_col),random_int)
    selecting_columns.sort()


    sample_data = input_data[selecting_rows[:,None],selecting_columns]
    target_sample_data = target_data[selecting_rows]
    target_sample_data = target_sample_data.reshape(-1,1)

    replicated_sample_data = input_data[replicating_rows[:,None],selecting_columns]
    target_replicated_sample_data = target_data[replicating_rows]
    target_replicated_sample_data =target_replicated_sample_data.reshape(-1,1)

    final_sample_data = np.vstack((sample_data,replicated_sample_data))
    final_target_data = np.vstack((target_sample_data,target_replicated_sample_data))


    final_sampled_input_data = final_sample_data.tolist()
    final_sampled_target_data = final_target_data.tolist()
    selected_rows    = selecting_rows.tolist()
    selected_columns = selecting_columns

#     print ("Single Sample Generated")
    return final_sampled_input_data , final_sampled_target_data,selected_rows,selected_co
lumns
```

Grader function - 1 </fongt>

In [26]:

```
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[26]:

```
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 [27]:

```
# 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)

#     print("30 Samples Generated")
    return list_input_data,list_output_data,list_selected_row,list_selected_columns

list_input_data,list_output_data,list_selected_row,list_selected_columns = generate_30_sa
mples(x,y)
```

Grader function - 2

In [28]:

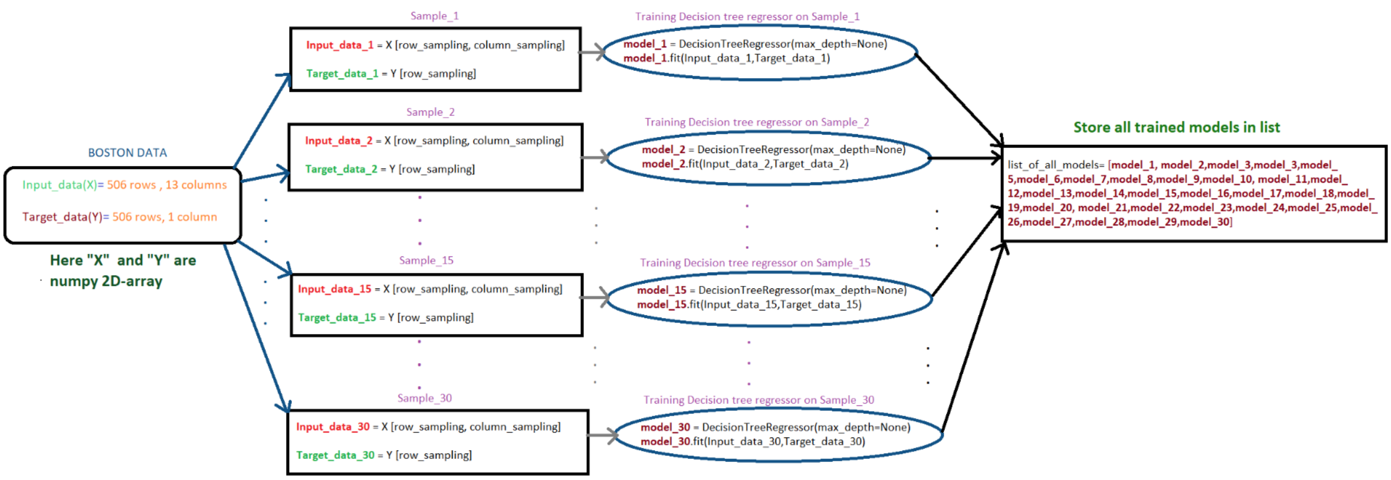
```
def grader_30(a):
    assert(len(a)==30 and len(a[0])==506)
    return True
grader_30(list_input_data)
```

Out[28]:

```
True
```

Step - 2

Flowchart for building tree

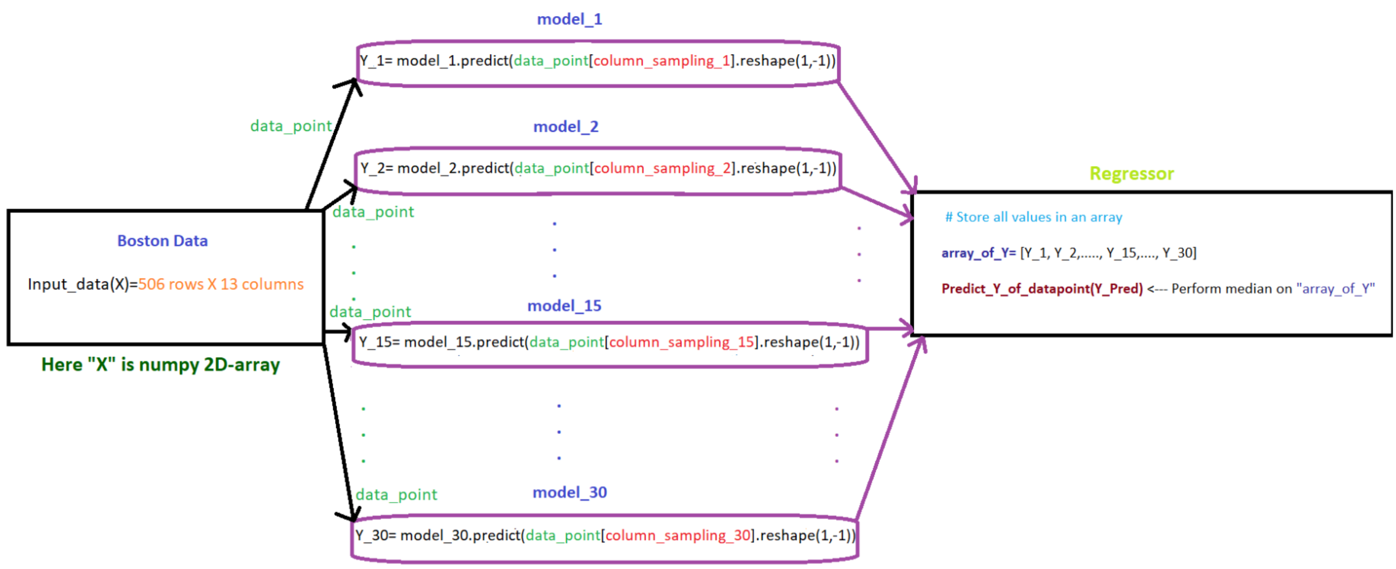


- **Write code for building regression trees**

In [76]:

```
def dt_regressors(list_input_data,list_output_data):

    list_of_all_models = []

    for i in range(0,30):
        dt = DecisionTreeRegressor(max_depth=None)
        dt.fit(list_input_data[i],list_output_data[i])
        list_of_all_models.append(dt)
#      print ("30 DecisionTree Regressors Stored in a list")
    return list_of_all_models

list_of_all_models = dt_regressors(list_input_data,list_output_data)
```

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**

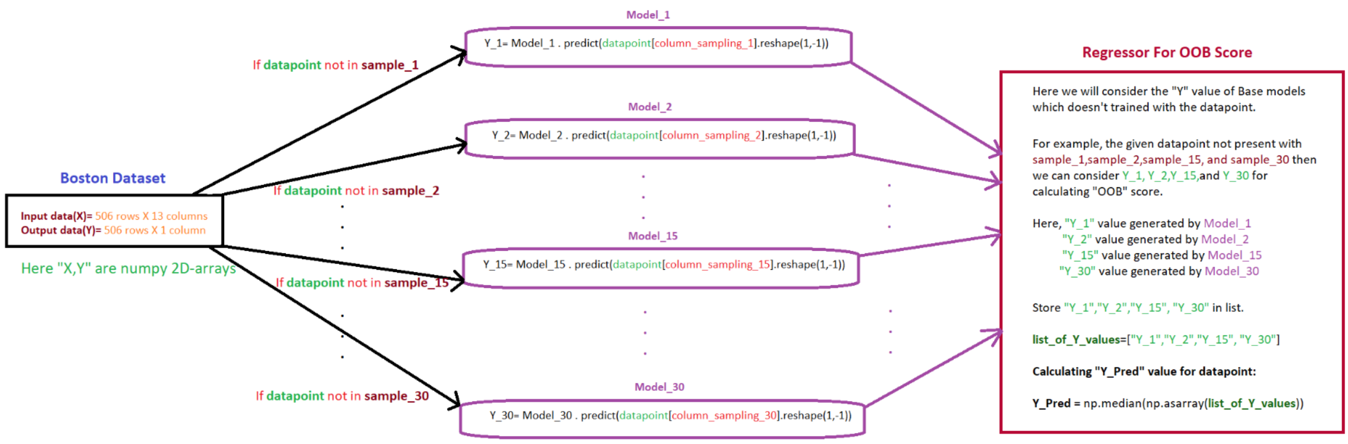
In [38]:

```
def mse():
    y_pred = []
    for data_point in x:
        array_y = []
        for k in range(0,30):
            y_pred_k = list_of_all_models[k].predict(data_point[list_selected_columns[k]
].reshape(1,-1))
            array_y.append(y_pred_k)
#          print (array_y)
        y_mean = np.mean(array_y)
#          print (y_mean)
#          print (y[0])
        y_pred.append(y_mean)
    mse = mean_squared_error(y, y_pred)
    return mse

mse_score = mse()
print (mse_score)
```

```
2.1670423822898752
```

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 [72]:

```
def oob_score():
    y_pred_oob = []
    for i in range(len(x)):
        y_pred_array_not_k = []
        for j in range(0,30):
            if i not in list_selected_row[j]:
#                 print ("The row number {0} is not in the sample {1}".format(i,j))
#                 print (list_selected_row[j])
                y_pred_not_k = list_of_all_models[j].predict(x[i][list_selected_columns[
j]].reshape(1,-1))
                y_pred_array_not_k.append(y_pred_not_k)

            else:
                continue
        y_mean_oob = np.round(np.mean(y_pred_array_not_k),1)
        y_pred_oob.append(y_mean_oob)
    OOB_Score = np.round(np.mean(np.square(y-y_pred_oob)),1)
    return OOB_Score

OOBScore = oob_score()
print (OOBScore)
```

13.4

Task 2

In [73]:

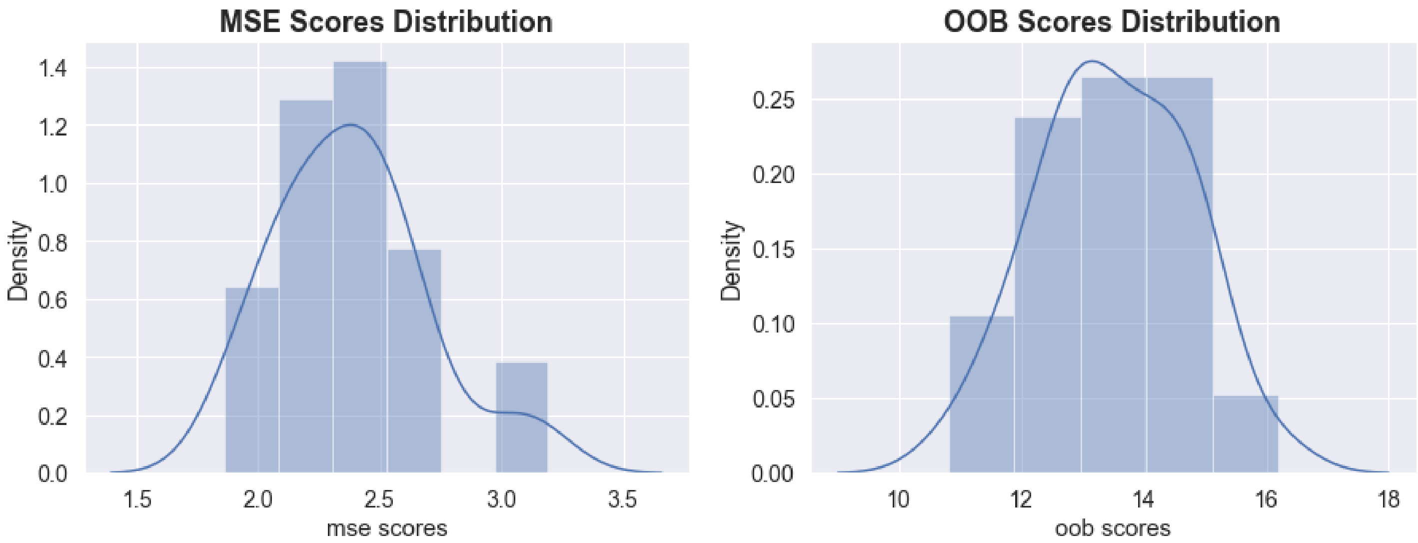
```
MSE_Scores = []
OOB_Scores = []

for i in tqdm(range(0,35)):
    list_input_data,list_output_data,list_selected_row,list_selected_columns = generate_
30_samples(x,y)
    list_of_all_models = dt_regressors(list_input_data,list_output_data)
```

```
    mse_score = mse()
    OOBScore = oob_score()
    MSE_Scores.append(mse_score)
    OOB_Scores.append(OOBScore)
    i = i+1
```

```
100%|██████████████████████████████████████████████████|
35/35 [01:05<00:00,  1.86s/it]
```

In [77]:

```
sns.set_theme(style="whitegrid")
sns.set(font_scale = 1.3)
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(15,5))

pdf1 = sns.distplot(MSE_Scores,ax = ax1)
pdf2 = sns.distplot(OOB_Scores,ax = ax2)
pdf1.set_title('MSE Scores Distribution',fontsize = 18,weight = 'bold')
pdf2.set_title('OOB Scores Distribution',fontsize = 18,weight = 'bold')
ax1.set_xlabel('mse scores',fontsize = 15)
ax2.set_xlabel('oob scores',fontsize = 15)
plt.show()
```



1. **PDF of both MSE and OOB Scores are almost normally distributed**
2. **As population standard deviation is not known, we will find CI using sample mean, standard error and quantiles of normal distribution**
3. **SE ~ s/sqrt(n)**
4. **Let's calculate 99% CI. Value of z is 2.58 in that case.**

In [79]:

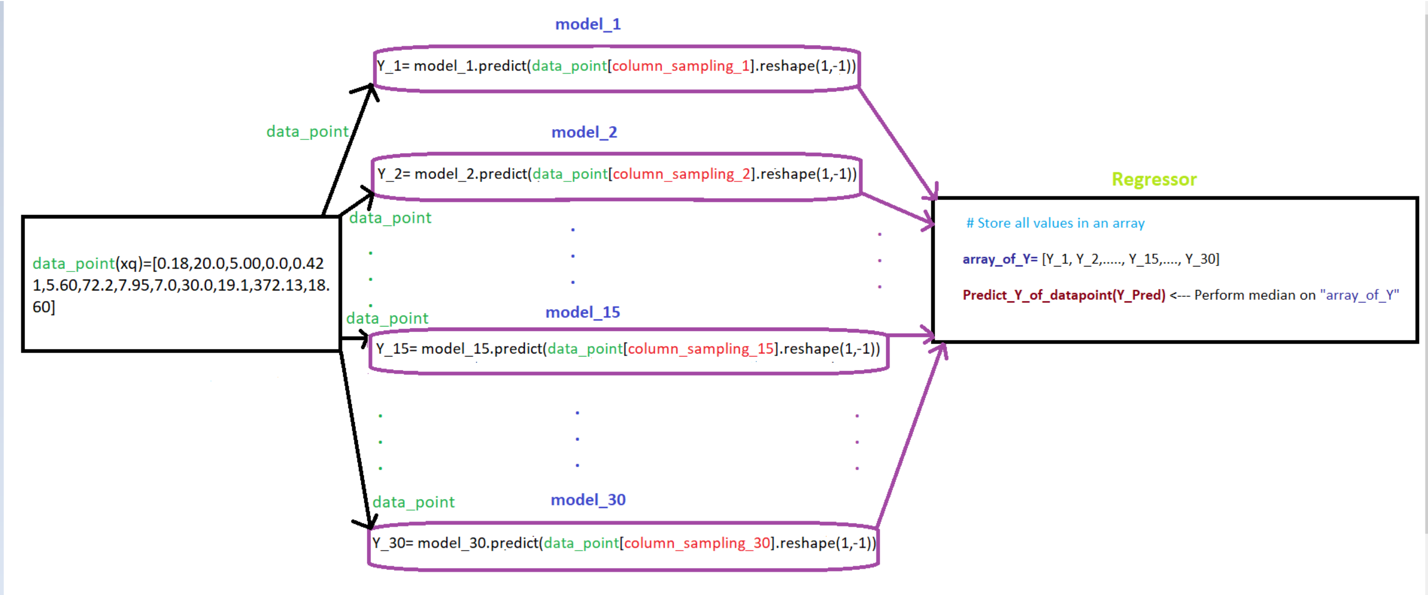
```
mse_std_dev = np.array(MSE_Scores).std()
oob_std_dev = np.array(OOB_Scores).std()
mse_mean =  np.array(MSE_Scores).mean()
oob_mean =np.array(OOB_Scores).mean()
n = 35

left_limit_mse  = np.round(mse_mean - 2.58*(mse_std_dev/np.sqrt(n)), 3)
right_limit_mse = np.round(mse_mean + 2.58*(mse_std_dev/np.sqrt(n)), 3)

left_limit_oob  = np.round(oob_mean - 2.58*(oob_std_dev/np.sqrt(n)), 3)
right_limit_oob = np.round(oob_mean + 2.58*(oob_std_dev/np.sqrt(n)), 3)

print ("99% CI of MSE Score is [{0} , {1}]".format(left_limit_mse,right_limit_mse))
print ("99% CI of OOB Score is [{0} , {1}]".format(left_limit_oob,right_limit_oob))
```

```
99% CI of MSE Score is [2.246 , 2.518]
99% CI of OOB Score is [12.978 , 14.027]
```

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 [80]:

```
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]
```

In [81]:

```
y_pred = []
array_y = []
for k in range(0,30):
    xq_k = (np.array([xq[i] for i in list_selected_columns[k]])).reshape(1,-1)
    y_pred_k = list_of_all_models[k].predict(xq_k)
    array_y.append(y_pred_k)
y_pred = np.mean(array_y)

print (y_pred)
```

```
21.123333333333334
```

Observations

In Task 1

1. **Each tree in a random forest learns from a random sample of the data points. The samples are drawn with replacement, this is known a s bootstrapping. In this task firstly, we have created 30 datasets. Each dataset have different rows and columns. We build Decision Tree Regressor model on each of the dataset till maximum depth so that model has high variance. The idea is that by training each tree on different samples, although each tree might have high variance with respect to a particular set of the training data, overall, the entire forest will have lower variance but not at the cost of increasing the bias.**
2. **We have made predictions by averaging the predictions of each decision tree.**
3. **By using multiple sample data sets and then testing multiple models, it can increase robustness.**

In Task 2

1. **From PDF we can see that MSE Scores are almost normally distributed. Hence, to calculate 99% CI of MSE Score we can take 2 standard deviations from mean which is [2.246 , 2.518]**

2. **Same observation as above. 99% CI of OOB Score is [12.978 , 14.027]**

In Task 3

1. **y_q of the x_q given is ~21.1**

In [ ]:

```
```