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
```

In [2]:

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

In [3]:

```
boston.feature_names
```

Out[3]:

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

In [4]:

```
x.shape
```

Out[4]:

```
(506, 13)
```

In [5]:

```
x[:5]
```

Out[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

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 [6]:

```
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
    # Please have a look at this link https://docs.scipy.org/doc/numpy-1.16.0/reference/generated/numpy
.random.choice.html for more details
    # Please follow above pseudo code for generating samples
    N = input_data.shape[0]
    C = input_data.shape[1]

    selecting_rows = np.random.choice(N,size=303,replace=False)
    replacing_rows = np.random.choice(selecting_rows,size=203,replace=False)
    selecting_columns = np.random.choice(C,size=np.random.randint(3,13),replace=False)

    sample_data    =   input_data[selecting_rows[:,None],selecting_columns]
    target_of_sample_data = target_data[selecting_rows]

    # replicating data
    replicating_sample_data = input_data[replacing_rows[:,None],selecting_columns]
    target_of_replacing_sample_data = target_data[replacing_rows]


    # concatinating data
    sampled_input_data = np.vstack((sample_data,replicating_sample_data))
    sampled_target_data = np.vstack((target_of_sample_data.reshape(-1,1),target_of_replacing_sample_dat
```

```
a.reshape(-1,1)))



    return list(sampled_input_data),list(sampled_target_data),list(selecting_rows),list(selecting_colum
ns)


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

Grader function - 1 </fongt>

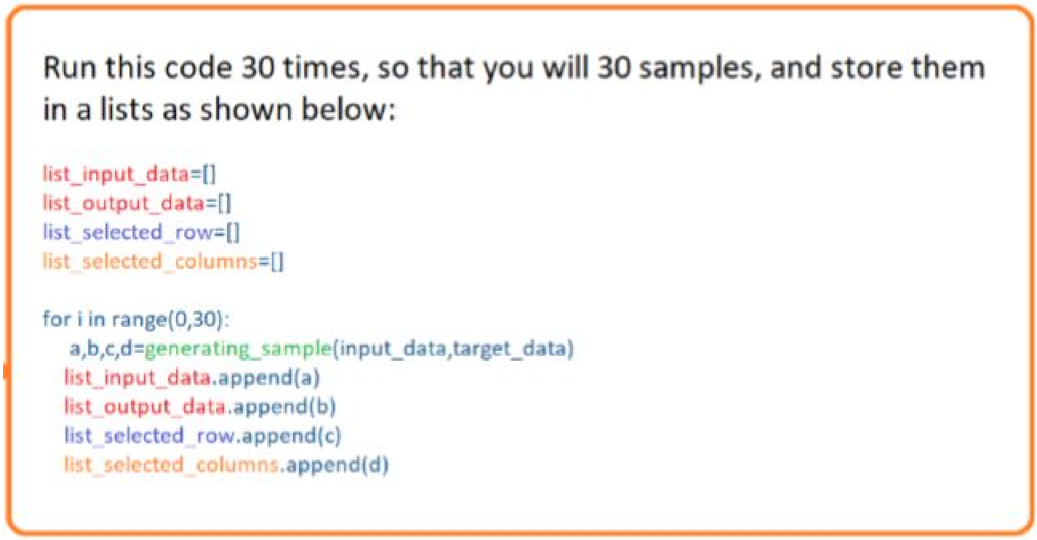
In [7]:

```
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)
    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**



In [27]:

```
# 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

In [28]:

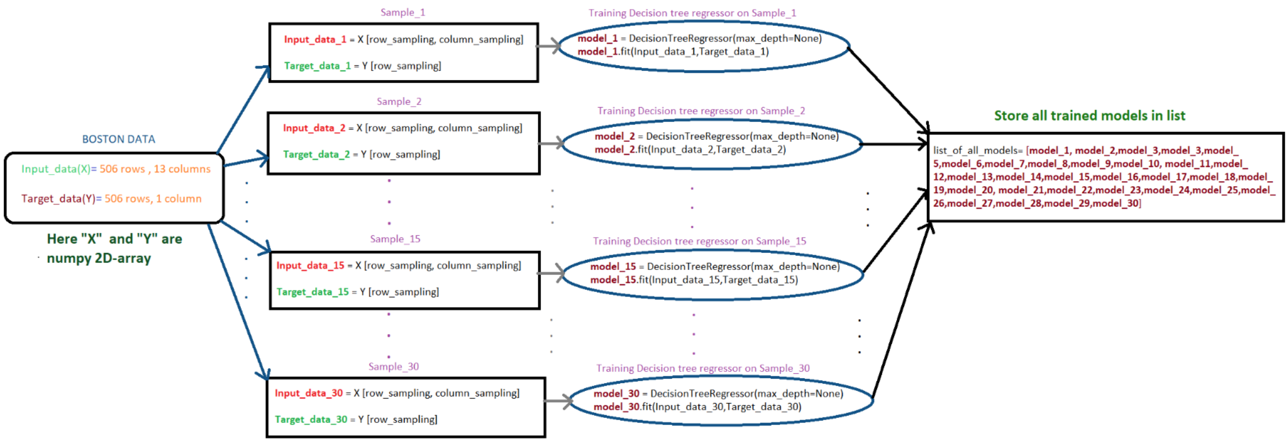
```
def grader_30(a):
    assert(len(a)==30 and len(a[0])==506)
    return True
grader_30(list_input_data)
```

Out[28]:

```
True
```

Step - 2

Flowchart for building tree



- **Write code for building regression trees**

In [29]:

```
from sklearn.tree import DecisionTreeRegressor
```

In [30]:

```
list_of_all_models = []
for i in range(len(list_input_data)):
  model = DecisionTreeRegressor(max_depth=None)
  model.fit(list_input_data[i],list_output_data[i])
  list_of_all_models.append(model)
```

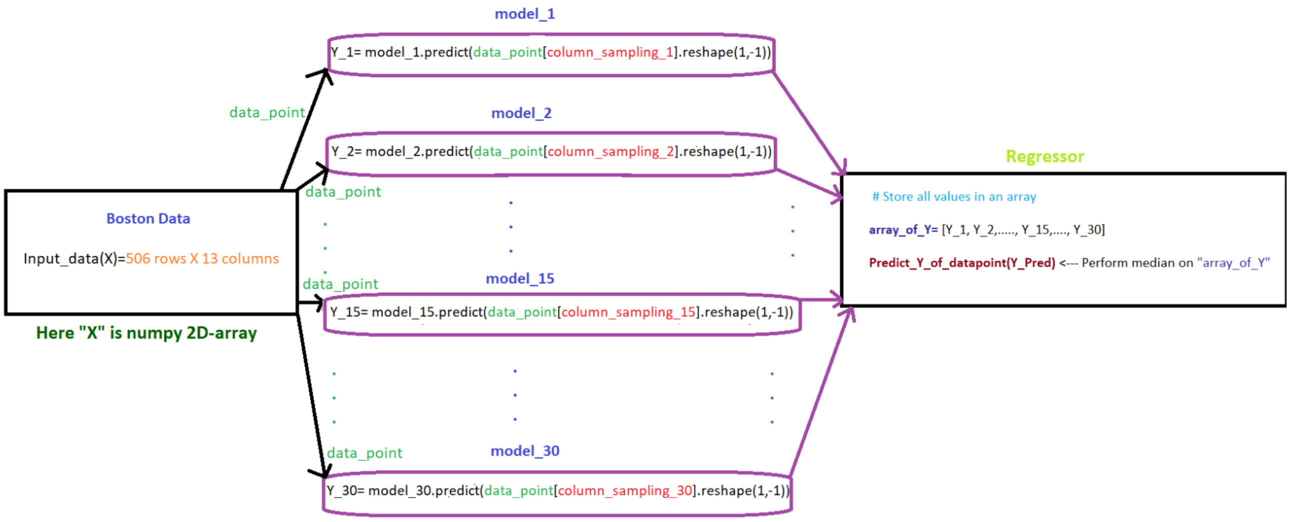
In [31]:

```
len(list_of_all_models)
```

Out[31]:

```
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 [32]:

```
predicted_y_of_datapoint = []
for data_point in x:
  predicted_y = []
  for i in range(len(list_of_all_models)):
    y_pred = list_of_all_models[i].predict(data_point[tuple([list_selected_columns[i]])].reshape(1,-1))
    predicted_y.append(y_pred)
    # using mean
  avg  = np.mean(predicted_y)
  predicted_y_of_datapoint.append(avg)
```

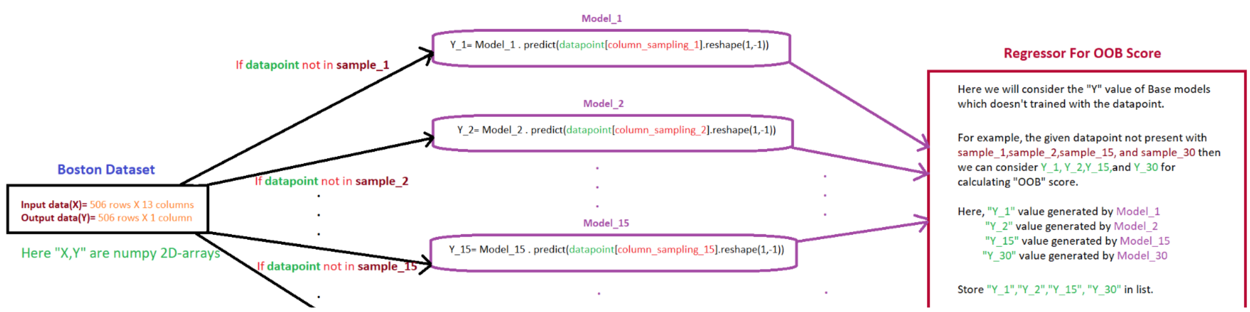
In [33]:

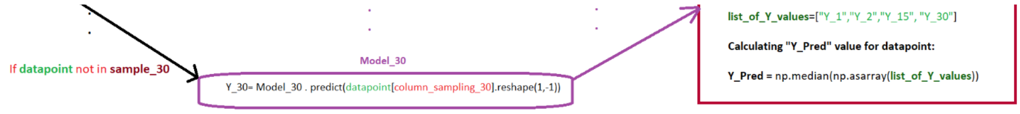
```
# mse score
mse = mean_squared_error(y,predicted_y_of_datapoint)
print("MSE score = "+str(mse))
```

```
MSE score = 2.460451477854294
```

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 [34]:

```
predicted_ys = []
for index in range(len(x)):
  pred_y = []
  for i in range(30):
    if (index not in list_selected_row[i]):
      y_pred = list_of_all_models[i].predict(x[index][tuple([list_selected_columns[i]])].reshape(1,-1))

      pred_y.append(y_pred)
  avg = np.mean(pred_y)
  predicted_ys.append(avg)
```

In [35]:

```
# OOB score
oob= mean_squared_error(y,predicted_ys)
print("OOB score = "+str(oob))
```

```
OOB score = 13.094766379865051
```

Task 2

In [36]:

```
# function to get MSE and OOB
def mseAndOob(input_data,target_data):
  # lists to store 30 samples of input,output,selected_rows and selected_columns.
  list_input_data =[]
  list_output_data =[]
  list_selected_row= []
  list_selected_columns=[]
  list_of_all_models = []
  # generate 30 samples using generating_samples
  for i in range(0,30):
    final_sample_data,final_target_data,selecting_rows,selecting_columns = generating_samples(input_dat
a,target_data)
    # training model on input and output data and storing model in a list.
    model = DecisionTreeRegressor(max_depth=None )
    model.fit(final_sample_data,final_target_data)
    list_of_all_models.append(model)
   # store 30 samples of input,output,selected_rows and selected_columns
    list_input_data.append(final_sample_data)
    list_output_data.append(final_target_data)
    list_selected_row.append(selecting_rows)
    list_selected_columns.append(selecting_columns)

  # calculating MSE:
  predicted_y_of_datapoint = []
  for data_point in input_data:
    predicted_y = []
    for i in range(0,30):
```

```
      pred = list_of_all_models[i].predict(data_point[tuple([list_selected_columns[i]])].reshape(1,-1))
      predicted_y.append(pred)
    predicted_y = sorted(predicted_y)
    avg  = np.mean(predicted_y)
    predicted_y_of_datapoint.append(avg)


  mse = mean_squared_error(target_data,predicted_y_of_datapoint)

  # calculating OOB:
  predicted_ys = []
  for index in range(len(input_data)):
    pred_y = []
    for i in range(30):
      if (index not in list_selected_row[i]):
        y_pred = list_of_all_models[i].predict(x[index][tuple([list_selected_columns[i]])].reshape(1,-1
))
        pred_y.append(y_pred)
    avg = np.mean(pred_y)
    predicted_ys.append(avg)
  oob= mean_squared_error(y,predicted_ys)

  return mse,oob
```

In [37]:

```
from tqdm.notebook import tqdm
```

In [38]:

```
list_of_mse = []
list_of_oob = []

for i in tqdm(range(0,35)):
  mse,oob = mseAndOob(x,y)
  list_of_mse.append(mse)
  list_of_oob.append(oob)
```

In [39]:

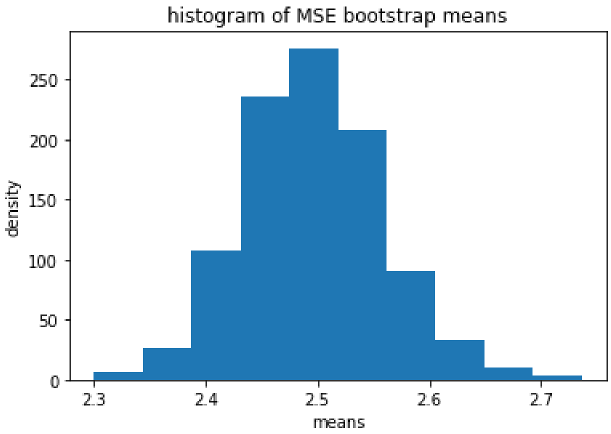
```
from sklearn.utils import resample
import matplotlib.pyplot as plt
```

C.I of MSE

In [40]:

```
# using bootstrap
n_iterations = 1000
n_size = 30

# using MSE values:
input = np.array(list_of_mse)

# run bootstrap:
means = list()
for i in range(n_iterations):
  s = resample(input,n_samples=n_size )
  m = np.mean(s)
  means.append(m)

# plot scores:
plt.hist(means)
plt.title("histogram of MSE bootstrap means")
plt.xlabel("means")
plt.ylabel("density")
plt.show()
```

```
# Confidence Interval:
alpha = 0.95
p = ((1.0-alpha)/2.0)*100
lower = np.percentile(means,p)

p = (alpha+((1.0-alpha)/2.0))*100
upper = np.percentile(means,p)
```



In [41]:

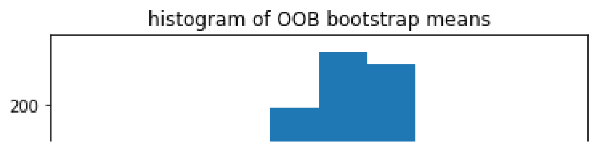
```
# C.I of MSE
print("%.1f percent confidence interval of MSE %.1f and %.1f "%(alpha*100,lower,upper))
```

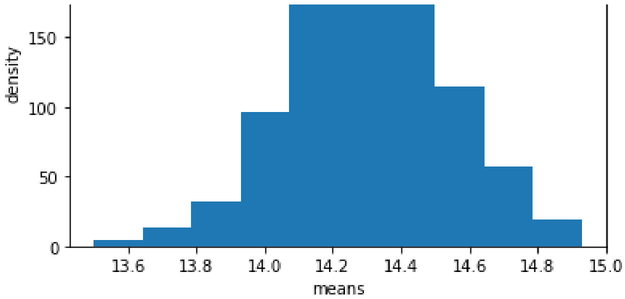
```
95.0 percent confidence interval of MSE 2.4 and 2.6
```

C.I of OOB

In [42]:

```
# using bootstrap
n_iterations = 1000
n_size = 30
# using OOB values
input = np.array(list_of_oob)
# run bootstrap:

means = list()
for i in range(n_iterations):
  s = resample(input,n_samples=n_size)
  m = np.mean(s)
  means.append(m)

# plot scores:
plt.hist(means)
plt.title("histogram of OOB bootstrap means")
plt.xlabel("means")
plt.ylabel("density")
plt.show()

# Confidence Interval:
alpha = 0.95
p = ((1.0-alpha)/2.0)*100
lower = np.percentile(means,p)

p = (alpha+((1.0-alpha)/2.0))*100
upper = np.percentile(means,p)
```





In [43]:

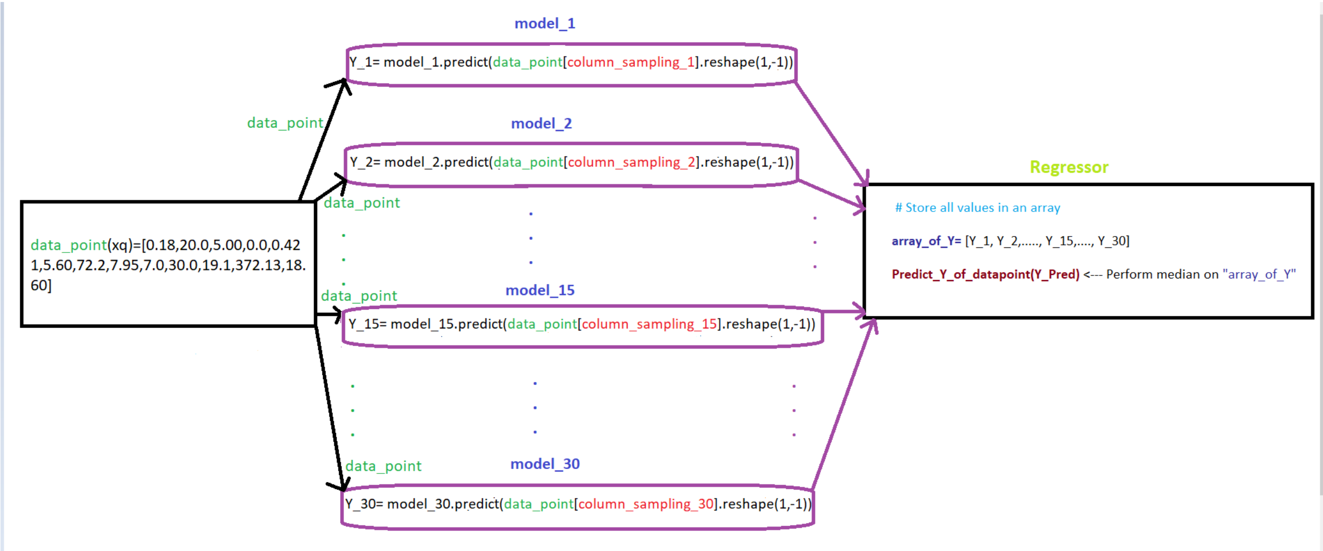
```
# C.I of OOB
print("%.1f percent confidence interval of OOB %.1f and %.1f "%(alpha*100,lower,upper))
```

```
95.0 percent confidence interval of OOB 13.8 and 14.8
```

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 [44]:

```
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])
```

In [45]:

```
query_point = xq
pred_y = []
for i in range(30):
  pred = list_of_all_models[i].predict(query_point[tuple([list_selected_columns[i]])].reshape(1,-1))
  pred_y.append(pred)
predicted_value = np.mean(pred_y)

print("predicted value of xq = "+str(predicted_value))
```

```
predicted value of xq = 21.589444444444446
```

```
predicted value of xq = 21.589444444444446
```

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

Observations:

Task-1

1. to get predicted value for each input data point in MSE, we are using all models(30 high variance models) irresptive of whether the model is trained on the given data points or not. so out of all models, there are some models which are already trained on that perticular point (with different set of columns) help us to get a predicted value closer to actual values, hence we get 30 predicted values for each data point and on average of those we get very close value to our actual data point.
2. so, in MSE task, there is a high chance of our predicted values are closer to actual values.
3. But in OOB task, for each input data point we are using some selected models which are not trained previously on the that given data point. so our predicted values have less chance to be very closer to actual values, but it helps us to understand how our models are performing on unseen data points.
4. hence, MSE score is lesser than the OOB score.

Task-2

1. A sample having 35 MSE's and OOB's which are obtained from high variance models (where high variance models changes with change in small amount of train data, which we did here to get each MSE/OOB by chaning train data (using random selection) for each iteration)
2. 95% of C.I of MSE, would contain the population MSE
3. 95% of C.I of OOB, would contain the population OOB

Task-3

1. predicted value of give query point is calculated as above.