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

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

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

In [4]:
```python
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
- 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

alt text

- **Write code for generating samples**

In [5]:
```python
#Modules used
from sklearn.tree import DecisionTreeRegressor
import pandas as pd
from math import sqrt
import numpy as np
```

In [6]:
```python
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


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

    Selecting_rows=np.random.choice(np.arange(input_data.shape[0]),size=303,replace=False)
    Replacing_rows=np.random.choice(np.arange(Selecting_rows.shape[0]),size=203,replace=False)
    Selecting_columns=np.random.choice(np.arange(input_data.shape[1]),size=np.random.randint(4,13),replace=False)
    sample_data=input_data[Selecting_rows[:,None],Selecting_columns]
    target_of_sample_data=target_data[Selecting_rows]
    #Replacted data
    Replicated_sample_data=sample_data[Replacing_rows]
    target_of_replicated_sample_data=target_of_sample_data[Replacing_rows]
    #concantacing the data
    final_sample_data=np.vstack([sample_data,Replicated_sample_data])
    final_target_data=np.vstack([target_of_sample_data.reshape(-1,1),target_of_replicated_sample_data.reshape(-1,1)])
    return list(final_sample_data),list(final_target_data),list(Selecting_rows),list(Selecting_columns)
```

In [ ]:

Grader function - 1 </fongt>

In [7]:
```python
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**

alt text

In [8]:
```python
# 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 [9]:
```python
def grader_30(a):
    assert(len(a)==30 and len(a[0])==506)
    return True
grader_30(list_input_data)
```

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

Step - 2

Flowchart for building tree

alt text

- **Write code for building regression trees**

In [10]:
```python
list_of_all_models=[]
def Tree(input_data,target_data):
    clf=DecisionTreeRegressor(max_depth=None)
    clf.fit(input_data,target_data)
    return clf
for i in range(0,30):# Forming 30 differnet base trees
    X=list_input_data[i]
    Y=list_output_data[i]
    T=Tree(X,Y)
    list_of_all_models.append(T)
```

Flowchart for calculating MSE

alt text

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 [11]:
```python
list_of_all_predcts=[]
for i in range(0,506):
    temp_y_pred=[]
    for j in range(0,30):
        pred=x[i,list_selected_columns[j]].reshape(1,-1)
        y_pred=list_of_all_models[j].predict(pred)#predcit y_value from model j which dont have that point
        temp_y_pred.append(y_pred)
    list_of_all_predcts.append((1/30)*np.sum(np.array(temp_y_pred)))
MSE=(1/506)*np.sum((y-list_of_all_predcts)**2)
print(f'Mean Square ERROR {MSE}' )
```

```
Mean Square ERROR 2.0956678902552084
```

Step - 3

Flowchart for calculating OOB score

alt text

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

- **Write code for calculating OOB score**

In [12]:
```python
#input_data should be a dataframe
#point_check should be a array with len of columns
#columns should be indexes of columns that are present
def check_point(df,pc,col):#Return True if the row is found in that input_data set
    df=pd.DataFrame(df)
    a=np.array(pc[col])
    return (df==a).all(1).any()
```

In [13]:
```python
list_of_new_y_pred=[]


for i in range(0,506):
    some_model_predcts=[]#used to store the moddel outputs for a point i from dataset X
    found_tree=False#Just to check whether a datapoint is found a tree without that datapoint
    k=0
    for j in range(0,30):
        a=check_point(list_input_data[j],x[i],list_selected_columns[j])#check whether a point exists in the jth model
        if a:#If exists pass
            pass
        else:#else pred the value for that datapoint
            found_tree=True
            k=k+1
            pred=x[i,list_selected_columns[j]].reshape(1,-1)
            y_pred=list_of_all_models[j].predict(pred)#predcit y_value from model j which dont have that point
            some_model_predcts.append(y_pred)#stroing all the predicted values for ith point
    list_of_new_y_pred.append((1/k)*np.sum(np.array(some_model_predcts)))
```

In [14]:
```python
OOB=(1/506)*np.sum((np.array(y)-np.array(list_of_new_y_pred))**2)
print(f'The OOBScore is {OOB}')
```

```
The OOBScore is 12.288839332109914
```

Task 2

In [15]:
```python
#Repeating task1 For 35 times
MSE_35=[]
OOB_35=[]
for _ in range(0,35):
    # Use generating_samples function to create 30 samples
    # store these created samples in a list
    list_input_data_35 =[]
    list_output_data_35 =[]
    list_selected_row_35= []
    list_selected_columns_35=[]
    for i in range(0,30):
        a,b,c,d = generating_samples(x, y)
        list_input_data_35.append(a)
        list_output_data_35.append(b)
        list_selected_row_35.append(c)
        list_selected_columns_35.append(d)
    list_of_all_models_35=[]
    def Tree(input_data,target_data):
        clf=DecisionTreeRegressor(max_depth=None)
        clf.fit(input_data,target_data)
        return clf
    for i in range(0,30):
        X=list_input_data_35[i]
        Y=list_output_data_35[i]
        T=Tree(X,Y)
        list_of_all_models_35.append(T)
    list_of_all_predcts_35=[]
    for i in range(0,506):
        temp_y_pred_35=[]
        for j in range(0,30):
            pred=x[i,list_selected_columns_35[j]].reshape(1,-1)
            y_pred=list_of_all_models_35[j].predict(pred)#predcit y_value from model j which dont have that point
            temp_y_pred_35.append(y_pred)
        list_of_all_predcts_35.append((1/30)*np.sum(np.array(temp_y_pred_35)))
    MSE=(1/506)*np.sum((np.array(y)-np.array(list_of_all_predcts_35))**2)
    #print(f'The MSE score is {MSE}')
    MSE_35.append(MSE)
    list_of_new_y_pred_35=[]
    for i in range(0,506):
        some_model_predcts_35=[]#used to store the moddel outputs for a point i from dataset X
        found_tree=False
        k=0
        for j in range(0,30):
            a=check_point(list_input_data_35[j],x[i],list_selected_columns_35[j])#check whether a point exists in the jth model
            if a:#If exists pass
                pass
            else:#else pred the value for that datapoint
                found_tree=True
                k=k+1
                pred=x[i,list_selected_columns_35[j]].reshape(1,-1)
                y_pred=list_of_all_models_35[j].predict(pred)#predcit y_value from model j which dont have that point
                some_model_predcts_35.append(y_pred)#stroing all the predicted values for ith point
        list_of_new_y_pred_35.append((1/k)*np.sum(np.array(some_model_predcts_35)))#find mean of the points and store in a new array
    OOB=(1/506)*np.sum((np.array(y)-np.array(list_of_new_y_pred_35))**2)
    #print(f'The OOB score is {OOB}')
    OOB_35.append(OOB)
    print(f'{_}', end=' ')
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
```

In [16]:
```python
print(OOB_35)
print(MSE_35)
```

```
[13.851426226748801, 13.448073027656783, 12.558260261587503, 13.672198242439741, 11.884333794890965, 14.067577550053254, 12.649400054228744, 13.641431683249365, 14.363030086444452,
14.277909113694674, 15.456461855139315, 14.44697205495984, 13.489906754108379, 15.092412901532821, 11.605218582644417, 13.885719970579938, 13.84268656940532, 14.307102281613947, 1
4.642076539555324, 16.88380249374791, 14.696349530746028, 14.151338012295108, 13.423881447757719, 13.164984463342053, 15.279193139103945, 12.6425369128406, 13.567132715542039, 15.3
56227055581536, 15.351175802113401, 14.267427883373157, 11.984782599173311, 13.704549773912065, 14.70050516610204, 12.828463287961071, 13.606294762288092]
[2.25068308629776, 2.190505307910018, 2.5246168900982404, 2.291803683721269, 2.04580164690382, 2.3123905357927095, 2.0121528546332885, 2.758933383527988, 2.46655966231156, 2.582642
7768018343, 2.8019691279005845, 2.463394397935833, 2.3613526053042495, 2.5328084114087726, 1.7465541501976274, 2.252641168713654, 2.1606913665641923, 2.7757870297784604, 2.80189559
7228321, 2.600298145873888, 2.4585574771690326, 2.4981203107672947, 2.3821162139930343, 2.2889391749621817, 2.7737661187114613, 2.281520138544569, 2.2155091128678084, 2.50255868031
4806, 2.427236209925339, 2.2020905717806087, 1.8268674700553502, 2.2974526127824126, 2.8648773425322815, 2.294777592348606, 2.2581529016005457]
```

In [17]:
```python
# OOB_35=[14.404254247090917, 11.356516121696368, 13.231594747137741, 16.010466855943655, 13.756024396771558, 14.092364041828846, 12.000842528238277, 12.228644015500443, 12.7244094
# MSE_35=[2.4604760679442155, 1.9940492319054686, 2.215918712800015, 2.81720723641689, 2.2205558123505575, 2.2496287623544964, 1.8501277231050526, 2.133561784170205, 2.283687479842
```

In [18]:
```python
def find_CI(li):#Find confidence interval for the given sampled data
    m=np.mean(li)
    se=np.std(li)/sqrt(len(li))
    CI=[m-(2*se),m+(2*se)]
    return CI
print(f'Confidence Interval of MSE is {find_CI(MSE_35)}')
print(f'Confidence interval of OOB is {find_CI(OOB_35)}')
```

```
Confidence Interval of MSE is [2.2969856011565746, 2.4747871849725342]
Confidence interval of OOB is [13.536496023269269, 14.280123553668659]
```

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.

alt text

- **Write code for TASK 3**

In [19]:
```python
#Predicting the Point using task1
temp_y_pred_=[]
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])
for j in range(0,30):
        pred=xq[list_selected_columns[j]].reshape(1,-1)
        y_pred=list_of_all_models[j].predict(pred)#predcit y_value from model j which dont have that point
        temp_y_pred_.append((y_pred))
print((1/30)*np.sum(np.asarray(temp_y_pred_)))
```

```
19.32333333333333
```

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

In [ ]:
```python
#Observations
#For Task-1
#1.In task one we have build a random forest with 30-base trees
#2. In this task-1 we have calculated the MSE and OOB for the given RandomForest

#for task-2
#In this task we have repeated task-1 for 35 times and got 35 different values of MSE and OOB
#With that sample we have calculated the CI

#for task3
#we have predicted the value for a given query point using the First RandomForest model we built in this.
```