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

```
import numpy as np # importing numpy for numerical computation
from sklearn.datasets import load_boston # here we are using sklearn's boston
from sklearn.metrics import mean_squared_error # importing mean_squared_error
```

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

```
x.shape
```

```
(506, 13)
```

```
x[: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
  $y^i_{pred} = \frac{1}{k}\sum_{\text{k= model which was buit on samples not included } x^i}(\text{predicted value of } x^i \text{ with } k$
  .
- 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**

```
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/
    # Please follow above pseudo code for generating samples

    #print(np.floor(0.6*len(input_data)))
    select_rows = np.random.choice(len(input_data),size = int(np.floor(0.6*len
    #print(select_rows,select_rows.shape)
    replace_rows = np.random.choice((select_rows),size = int(np.ceil(0.4*len(
    #print(replace_rows,replace_rows.shape)

    no_col = 8
    select_cols = np.random.choice(len(input_data[0,:]),size =no_col, replace
    #print(select_cols)
    sam = np.meshgrid(select_cols,select_rows)
    #print((sam[0].shape))
    #print(sam[1].shape)

    sam_data = input_data[sam[1],sam[0]]
    #print(sam_data.shape)

    target_sam_data = target_data[select_rows]
    #print(target_sam_data.shape)

    sam = np.meshgrid(select_cols,replace_rows)
    #print((sam[0].shape))
    #print(sam[1].shape)
```

```
    replicate_data = input_data[sam[1],sam[0]]
    #print(replicate_data.shape)

    target_rep_data = target_data[replace_rows]
    #print(target_rep_data.shape)

    final_sam_data = np.concatenate((sam_data,replicate_data),axis = 0)
    final_target_data  = np.concatenate((target_sam_data,target_rep_data))

    #print(final_sam_data.shape,final_target_data.shape,select_rows.shape,sele

    return list(final_sam_data) , list(final_target_data),list(select_rows),li
    #note please return as lists
```

Grader function - 1

```
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)
```

```
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)
```

```
# Use generating_samples function to create 30 samples
# store these created samples in a list
```

```
list_input_data =[]
list_output_data =[]
list_selected_rows= []
list_selected_columns=[]
no_base_models = 30
count = 0
i=0
while (i != (no_base_models+count)):
  a,b,c,d = generating_samples(x, y)
  d = sorted(d)
  i = i+1
  if ((d in list_selected_columns)):
    #print(count)
    count = count+1
    continue
  list_selected_columns.append(d)
  list_input_data.append(a)
  list_output_data.append(b)
  list_selected_rows.append(c)


```

```
print(len(list_selected_columns),len(list_selected_rows))
print(len(list_input_data),len(list_output_data))
```

```
30 30
30 30
```

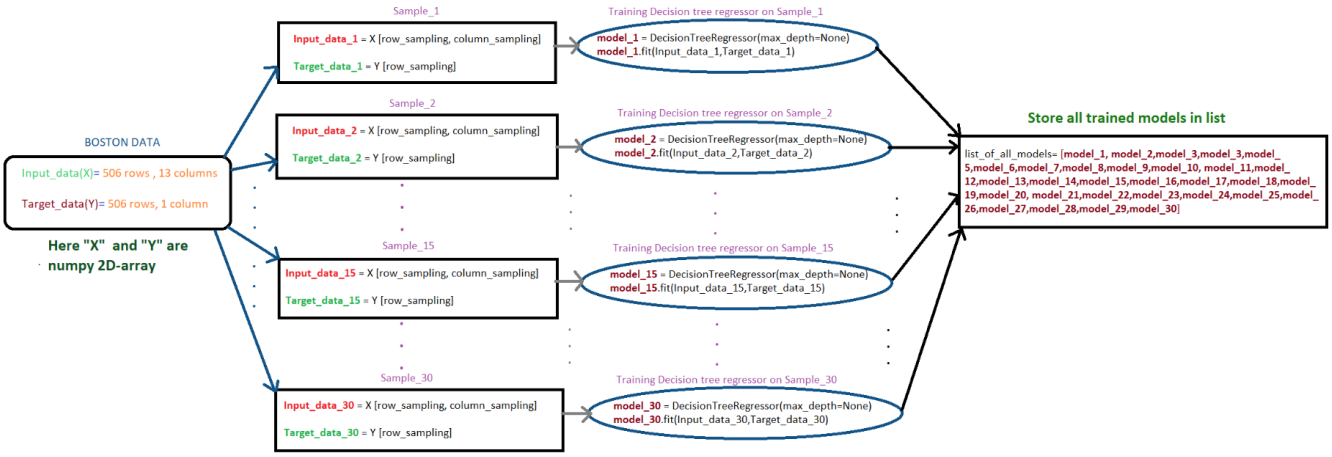
Grader function - 2

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

```
True
```

Step - 2

Flowchart for building tree



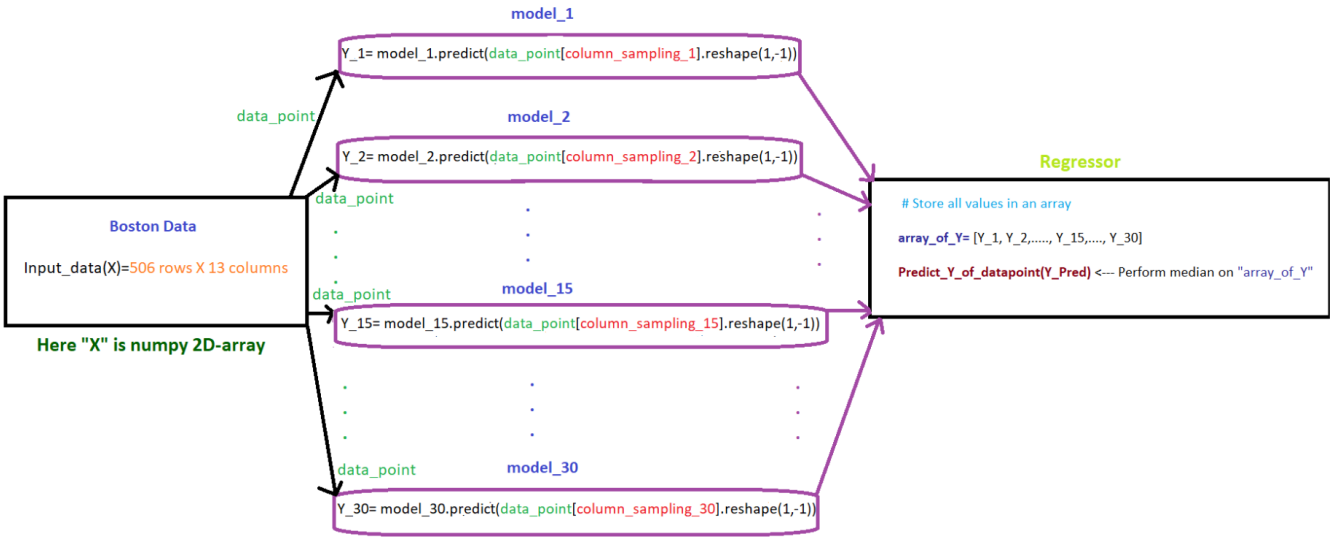
- **Write code for building regression trees**

```
from sklearn.tree import DecisionTreeRegressor

list_all_models = []
for i in range(no_base_models):
  input_data = list_input_data[i]
  target_data = list_output_data[i]
  model =  DecisionTreeRegressor()
  model.fit(input_data, target_data)
  list_all_models.append(model)

#print(list_all_models)
```

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**

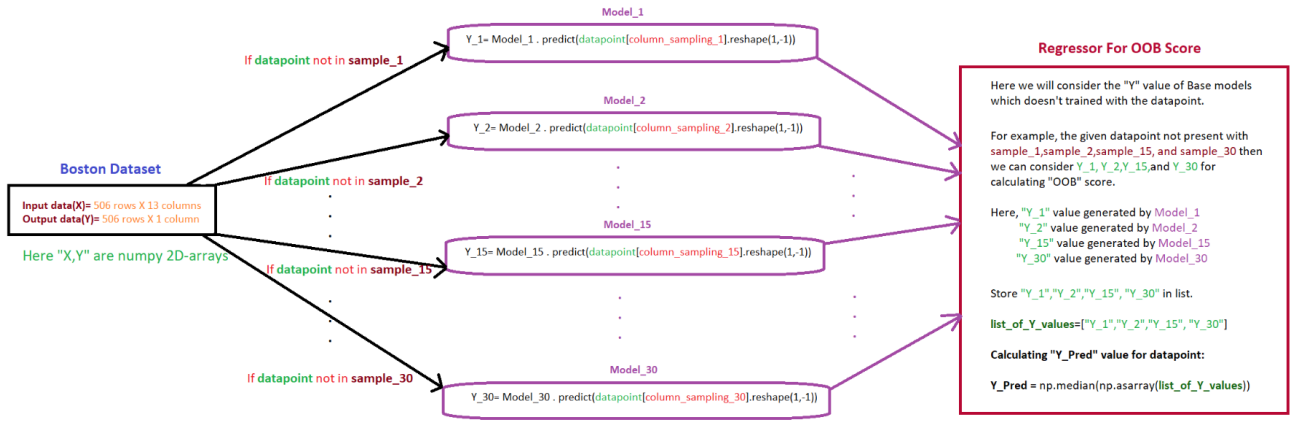
```
import numpy as np
from sklearn.metrics import mean_squared_error

y_pred = []
for i in range(no_base_models):
  sam_cols = list_selected_columns[i]
  sam_data = x[:,sam_cols]
  #print(sam_data.shape)
  model = list_all_models[i]
  y = model.predict(sam_data)
  #print(y.shape)
  y_pred.append(y)
y_pred = np.array(y_pred)
print((y_pred.shape))
y_pred= np.median(y_pred, axis= 0)
print((y_pred.shape))
mse = mean_squared_error(y,y_pred)
print(mse)
```

```
(30, 506)
(506,)
22.11139328063241
```

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**

```
y_pred = []
for i in range(len(x)):
  sam=[]
  for j in range(no_base_models):
    if i not in list_selected_rows[j]:
      model = list_all_models[j]
      sam.append(model.predict(x[i,list_selected_columns[j]].reshape(1,-1)))
  y_pred.append(np.median(sam))

#print(len(y_pred))
oobscore = mean_squared_error(y,y_pred)
print(oobscore)


```

```
48.836062252964425
```

```
print(no_base_models,mse,oobscore)
```

```
30 22.11139328063241 48.836062252964425
```

Task 2

```
Train_mse = []
OOb_score = []
no_iter = 35
for count2 in range(no_iter):
  list_input_data =[]
  list_output_data =[]
  list_selected_rows= []
  list_selected_columns=[]
  no_base_models = 30
  count = 0
  i=0
  while (i != (no_base_models+count)):
    a,b,c,d = generating_samples(x, y)
    d = sorted(d)
    i = i+1
    if ((d in list_selected_columns)):
      #print(count)
      count = count+1
      continue
    list_selected_columns.append(d)
    list_input_data.append(a)
    list_output_data.append(b)
```

```
      list_output_data.append(b)
      list_selected_rows.append(c)

    #print(len(list_input_data), len(list_selected_columns ))
    list_all_models = []
    i=0
    for i in range(no_base_models):
      input_data = list_input_data[i]
      target_data = list_output_data[i]
      model = DecisionTreeRegressor()
      model.fit(input_data, target_data)
      list_all_models.append(model)

    y_pred = []
    for i in range(no_base_models):
      sam_cols = list_selected_columns[i]
      sam_data = x[:,sam_cols]
      #print(sam_data.shape)
      model = list_all_models[i]
      y_pred.append(model.predict(sam_data))

    y_pred = np.array(y_pred)
    #print((y_pred.shape))
    y_pred= np.median(y_pred, axis= 0)
    #print((y_pred.shape))
    mse = mean_squared_error(y,y_pred)
    #print(mse)
    Train_mse.append(mse)

    y_pred = []
    for i in range(len(x)):
      sam=[]
      for j in range(no_base_models):
        if i not in list_selected_rows[j]:
          model = list_all_models[j]
          sam.append(model.predict(x[i,list_selected_columns[j]].reshape(1,-1))
      y_pred.append(np.median(sam))

    #print(len(y_pred))
    oobscore = mean_squared_error(y,y_pred)
    print(mse,oobscore)
    OOb_score.append(oobscore)

  print(len(Train_mse))
  print(len(OOb_score))
```

```
34.98829051383399 45.772623517786506
27.723399209486146 29.881210474308304
27.935177865612715 34.208211462450606
25.881111660079156 29.504160079051466
19.98754940711454 21.600968379446552
24.246689723319996 41.32165019762818
15.200286561264864 20.85303359683799
7.379609683794486 14.544293478260863
17.458710474308205 20.80408102766797
26.312895256917127 37.78571146245058
14.318824110671793 20.11350296442683
26.29342885375504 32.893542490118634
20.154298418972207 26.444392292490043
21.782677865612577 33.84576086956515
27.77843379446646 27.244426877470392
23.41337944664018 23.297524703557258
27.819061264822103 28.64650691699599
23.01041996047444 22.844693675889417
23.303913043478186 22.502801383399188
33.51211956521746 47.2355583003952
27.935177865612626 29.55178853754942
13.776704545454422 20.667826086956385
30.73194664031629 44.32474802371543
16.82976778656112 21.768246047430765
23.508285573122603 24.676210474308377
27.93517786561269 28.218463438735213
18.09698616600785 25.15606719367582
```

```
print((Train_mse))
print((OOb_score))
```

```
[34.98829051383399, 27.723399209486146, 27.935177865612715, 25.88111166007915
[45.772623517786506, 29.881210474308304, 34.208211462450606, 29.5041600790514
```

```
x_mean = np.mean(Train_mse)
x_std = np.std(Train_mse)
#print(x_mean, x_std)
# for 95% confidence interval
upper_lim = x_mean-2*(x_std/np.sqrt(len(Train_mse)))
lower_lim = x_mean+2*(x_std/np.sqrt(len(Train_mse)))
#print(upper_lim,lower_lim)
print('95% confidence interval of mean square error is [{0},{1}]'.format(upper
```

```
95% confidence interval of mean square error is [20.92001236793118,26.1007883
```

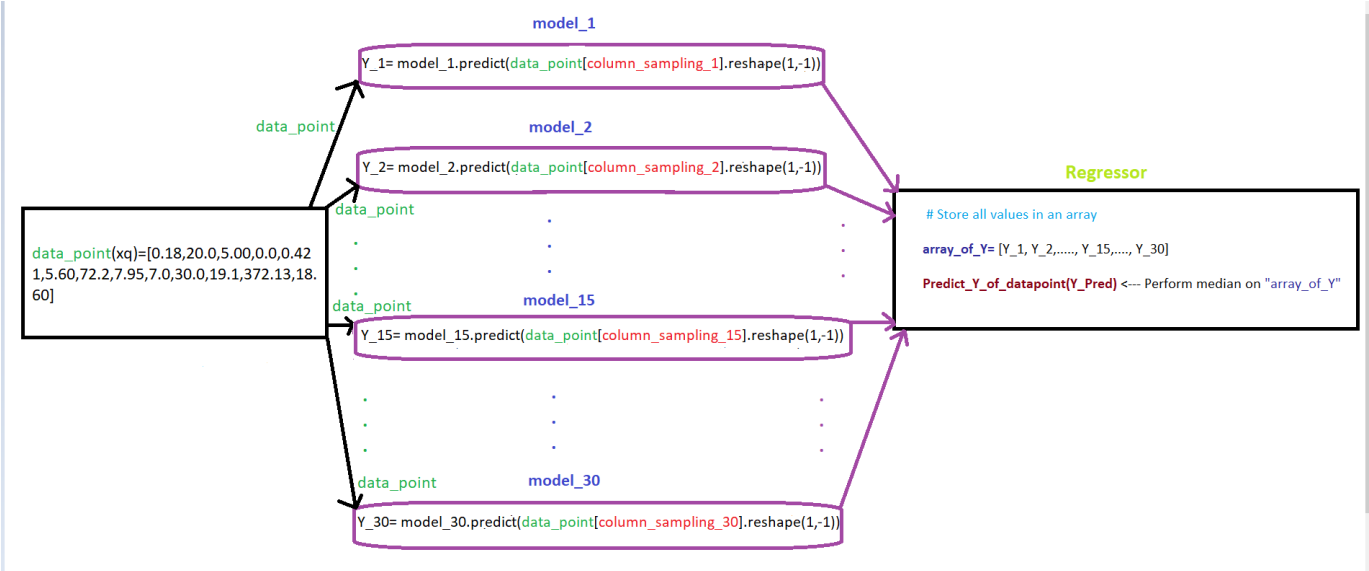
```
x_mean = np.mean(OOb_score)
x_std = np.std(OOb_score)
#print(x_mean, x_std)
# for 95% confidence interval
upper_lim = x_mean-2*(x_std/np.sqrt(len(OOb_score)))
lower_lim = x_mean+2*(x_std/np.sqrt(len(OOb_score)))
#print(upper_lim,lower_lim)
print('95% confidence interval for Out of Bag(OOB) score is [{0},{1}]'.format(
```

```
95% confidence interval for Out of Bag(OOB) score is [26.580092596300947,32.5
```

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**

```
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
for i in range(no_base_models):
  data = xq[list_selected_columns[i]].reshape(1,-1)
  model = list_all_models[i]
  y_pred.append(model.predict(data))
target = np.median(y_pred)
print(target)
```

```
21.9
```

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

Task1

1. By increasing the number of base learners in the Random Forest, Training MSE and OOB_score can be reduced as the more models learn different aspects of the data.
2. we can observe that by decreasing the no of features sampled Training_mse and OOB_Score decreased to some extent and later increased.(Conclusion is to keep the no of features as reasonably well so that good learning happens by the different base learners.)

Task2

1. As the number of iterations increases, the sample size of train_mse and oob_scores increases and there by 95% confidence interval range decreases by a lot extent.

Task3

1. As the no of base learners increases and the no of sampled features are reasonably well, the prediction of the input test point will be accurate and the test mse will be less.