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
import pandas as pd
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
import warnings
warnings.filterwarnings("ignore")
import random
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

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

In [3]:

```
print(x)
```

```
[[6.3200e-03 1.8000e+01 2.3100e+00 ... 1.5300e+01 3.9690e+02 4.9800e+00]
 [2.7310e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9690e+02 9.1400e+00]
 [2.7290e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9283e+02 4.0300e+00]
 ...
 [6.0760e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 5.6400e+00]
 [1.0959e-01 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9345e+02 6.4800e+00]
 [4.7410e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 7.8800e+00]]
```

In [4]:

```
print(actual_y)
```

```
[24.  21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 15.  18.9 21.7 20.4
 18.2 19.9 23.1 17.5 20.2 18.2 13.6 19.6 15.2 14.5 15.6 13.9 16.6 14.8
 18.4 21.  12.7 14.5 13.2 13.1 13.5 18.9 20.  21.  24.7 30.8 34.9 26.6
 25.3 24.7 21.2 19.3 20.  16.6 14.4 19.4 19.7 20.5 25.  23.4 18.9 35.4
 24.7 31.6 23.3 19.6 18.7 16.  22.2 25.  33.  23.5 19.4 22.  17.4 20.9
 24.2 21.7 22.8 23.4 24.1 21.4 20.  20.8 21.2 20.3 28.  23.9 24.8 22.9
 23.9 26.6 22.5 22.2 23.6 28.7 22.6 22.  22.9 25.  20.6 28.4 21.4 38.7
 43.8 33.2 27.5 26.5 18.6 19.3 20.1 19.5 19.5 20.4 19.8 19.4 21.7 22.8
 18.8 18.7 18.5 18.3 21.2 19.2 20.4 19.3 22.  20.3 20.5 17.3 18.8 21.4
 15.7 16.2 18.  14.3 19.2 19.6 23.  18.4 15.6 18.1 17.4 17.1 13.3 17.8
 14.  14.4 13.4 15.6 11.8 13.8 15.6 14.6 17.8 15.4 21.5 19.6 15.3 19.4
 17.  15.6 13.1 41.3 24.3 23.3 27.  50.  50.  50.  22.7 25.  50.  23.8
 23.8 22.3 17.4 19.1 23.1 23.6 22.6 29.4 23.2 24.6 29.9 37.2 39.8 36.2
 37.9 32.5 26.4 29.6 50.  32.  29.8 34.9 37.  30.5 36.4 31.1 29.1 50.
 33.3 30.3 34.6 34.9 32.9 24.1 42.3 48.5 50.  22.6 24.4 22.5 24.4 20.
 21.7 19.3 22.4 28.1 23.7 25.  23.3 28.7 21.5 23.  26.7 21.7 27.5 30.1
 44.8 50.  37.6 31.6 46.7 31.5 24.3 31.7 41.7 48.3 29.  24.  25.1 31.5
 23.7 23.3 22.  20.1 22.2 23.7 17.6 18.5 24.3 20.5 24.5 26.2 24.4 24.8
 29.6 42.8 21.9 20.9 44.  50.  36.  30.1 33.8 43.1 48.8 31.  36.5 22.8
 30.7 50.  43.5 20.7 21.1 25.2 24.4 35.2 32.4 32.  33.2 33.1 29.1 35.1
 45.4 35.4 46.  50.  32.2 22.  20.1 23.2 22.3 24.8 28.5 37.3 27.9 23.9
 21.7 28.6 27.1 20.3 22.5 29.  24.8 22.  26.4 33.1 36.1 28.4 33.4 28.2
 22.8 20.3 16.1 22.1 19.4 21.6 23.8 16.2 17.8 19.8 23.1 21.  23.8 23.1
 20.4 18.5 25.  24.6 23.  22.2 19.3 22.6 19.8 17.1 19.4 22.2 20.7 21.1
 19.5 18.5 20.6 19.  18.7 32.7 16.5 23.9 31.2 17.5 17.2 23.1 24.5 26.6
 22.9 24.1 18.6 30.1 18.2 20.6 17.8 21.7 22.7 22.6 25.  19.9 20.8 16.8
 21.9 27.5 21.9 23.1 50.  50.  50.  50.  50.  13.8 13.8 15.  13.9 13.3
 13.1 10.2 10.4 10.9 11.3 12.3  8.8  7.2 10.5  7.4 10.2 11.5 15.1 23.2
  9.7 13.8 12.7 13.1 12.5  8.5  5.   6.3  5.6  7.2 12.1  8.3  8.5  5.
 11.9 27.9 17.2 27.5 15.  17.2 17.9 16.3  7.   7.2  7.5 10.4  8.8  8.4
 16.7 14.2 20.8 13.4 11.7  8.3 10.2 10.9 11.   9.5 14.5 14.1 16.1 14.3
 11.7 13.4  9.6  8.7  8.4 12.8 10.5 17.1 18.4 15.4 10.8 11.8 14.9 12.6
 14.1 13.  13.4 15.2 16.1 17.8 14.9 14.1 12.7 13.5 14.9 20.  16.4 17.7
 19.5 20.2 21.4 19.9 19.  19.1 19.1 20.1 19.9 19.6 23.2 29.8 13.8 13.3
 16.7 12.  14.6 21.4 23.  23.7 25.  21.8 20.6 21.2 19.1 20.6 15.2  7.
  8.1 13.6 20.1 21.8 24.5 23.1 19.7 18.3 21.2 17.5 16.8 22.4 20.6 23.9
 22.  11.9]
```

In [5]:

```
x.shape
```

Out[5]:

```
(506, 13)
```

In [6]:

```
x[:5]
```

Out[6]:

```
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]])
```

In [7]:

```
corr = np.corrcoef(x)
print(corr)
```

```
[[1.         0.99285038 0.99434335 ... 0.9960369  0.99637298 0.9971554 ]
 [0.99285038 1.         0.99915891 ... 0.99785741 0.99764644 0.99807141]
 [0.99434335 0.99915891 1.         ... 0.99670581 0.99660711 0.99774018]
 ...
 [0.9960369  0.99785741 0.99670581 ... 1.         0.99998426 0.99972706]
 [0.99637298 0.99764644 0.99660711 ... 0.99998426 1.         0.99978612]
 [0.9971554  0.99807141 0.99774018 ... 0.99972706 0.99978612 1.        ]]
```

In [8]:

```
actual_y[:5]
```

Out[8]:

```
array([24. , 21.6, 34.7, 33.4, 36.2])
```

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

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**

In [9]:

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


    # selecting 303 random row indices from the input_data, without replacement
    rows_selected = np.random.choice(len(input_data), 303, replace=False)

    # Replacing Rows Extracting 206 reandom row indices from the abvoe rows_selected
    rows_203_extracted_from_rows_selected = np.random.choice(rows_selected, 203, replace=Fa

    # Now get 3 to 13 random column indices from input_data
    number_of_columns_to_select = random.randint(3, 13)
    columns_selected = np.array(random.sample(range(0, 13), number_of_columns_to_select ))

    sample_data = input_data[rows_selected[:, None], columns_selected]

    target_of_sample_data = target_data[rows_selected]

    # Now Replication of Data for 203 data points out of 303 selected points
    replicated_203_sample_data_points = input_data[rows_203_extracted_from_rows_selected[:,
    target_203_replicated_sample_data = target_data[rows_203_extracted_from_rows_selected]

    # Concatenating data
    final_sample_data = np.vstack((sample_data, replicated_203_sample_data_points ))

    final_target_data = np.vstack((target_of_sample_data.reshape(-1, 1), target_203_replica

    return final_sample_data, final_target_data, rows_selected, columns_selected

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

Grader function - 1

In [10]:

```
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, actual_y)
grader_samples(a,b,c,d)
```

Out[10]:

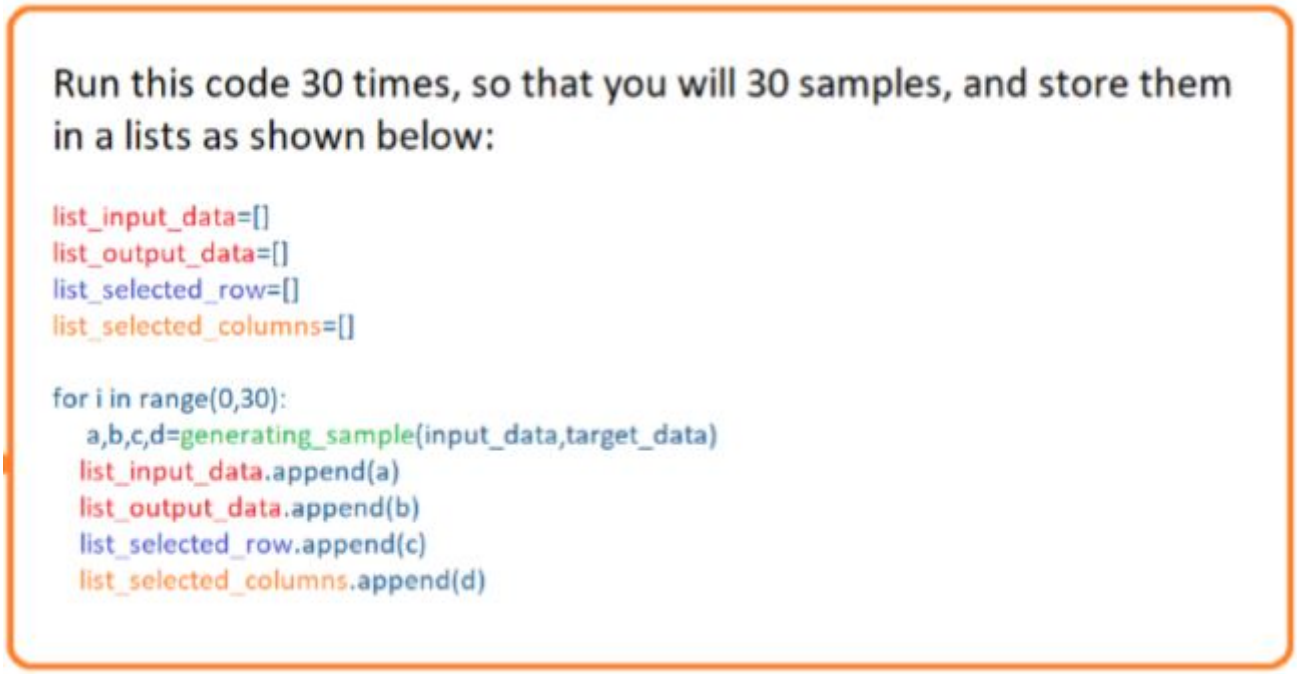
```
True
```

In [11]:

```
print(d)
```

```
[ 4  2  8 11  6  7 10  0  1  3  5]
```

- **Create 30 samples**



In [12]:

```
# 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, actual_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 [13]:

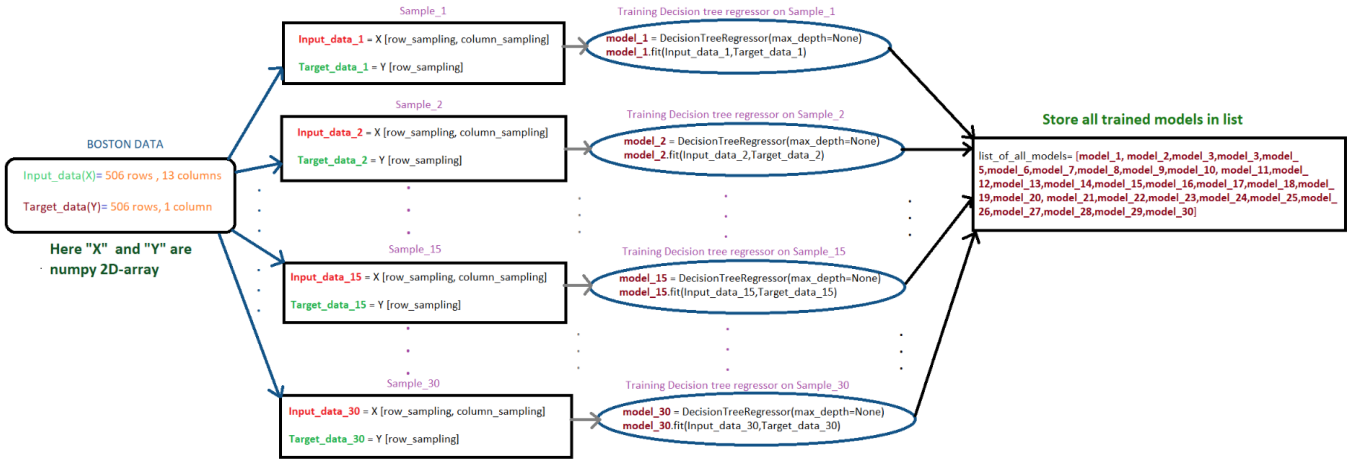
```
def grader_30(a):
    assert(len(a)==30 and len(a[0])==506)
    return True
grader_30(list_input_data)
```

Out[13]:

```
True
```

Step - 2

Flowchart for building tree

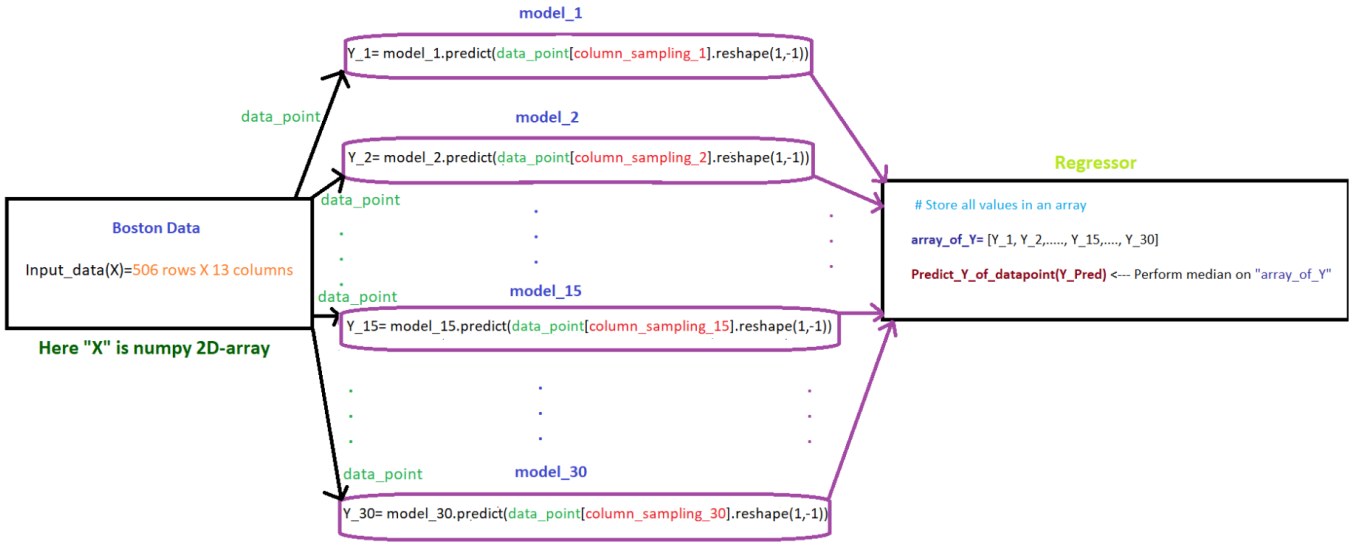


- **Write code for building regression trees**

In [14]:

```
#list_input_data[i] --> X_train, list_output_data[i] --> Y_train

from sklearn.tree import DecisionTreeRegressor
list_of_all_models_decision_tree = []
for i in range(0, 30):
    model_i = DecisionTreeRegressor(max_depth=None)
    model_i.fit(list_input_data[i], list_output_data[i])
    list_of_all_models_decision_tree.append(model_i)
```

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**

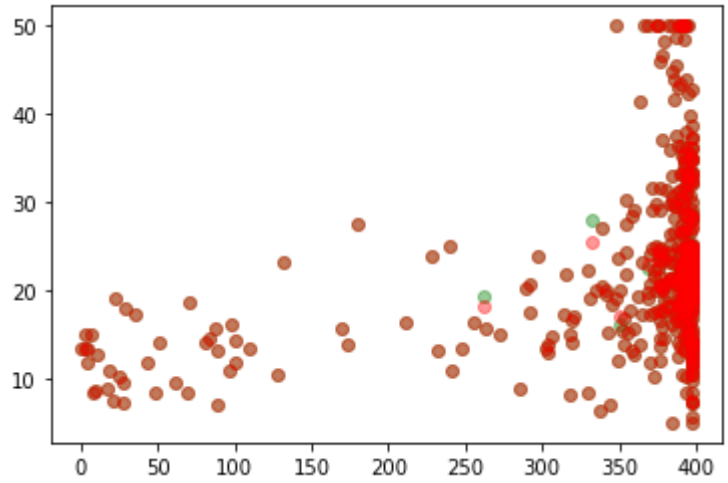
In [15]:

```
from sklearn.metrics import mean_squared_error
from statistics import median
from sklearn.metrics import mean_absolute_error

array_of_Y = []

for i in range(0, 30):
    data_point_i = x[:, list_selected_columns[i]]
    predict_y_i = list_of_all_models_decision_tree[i].predict(data_point_i) #data_point_i--
    array_of_Y.append(predict_y_i)


predicted_array_of_target_y = np.array(array_of_Y)
predicted_array_of_target_y = predicted_array_of_target_y.transpose()

print(predicted_array_of_target_y.shape)

# Now to calculate MSE, first calculate the Median of Predicted Y
# passing axis=1 will make sure the medians are computed along axis=1
predicted_y = np.median(predicted_array_of_target_y, axis=1)
predicted_y.shape

print("MSE : ", mean_squared_error(actual_y, predicted_y)) #actual_y --> Y_test, predicted_
print("RMSE : ",mean_squared_error(actual_y, predicted_y, squared=False))
print("MAE : ",mean_absolute_error(actual_y,predicted_y))
```

```
(506, 30)
MSE :  0.03241724308300394
RMSE :  0.18004789108180064
MAE :  0.019318181818181814
```

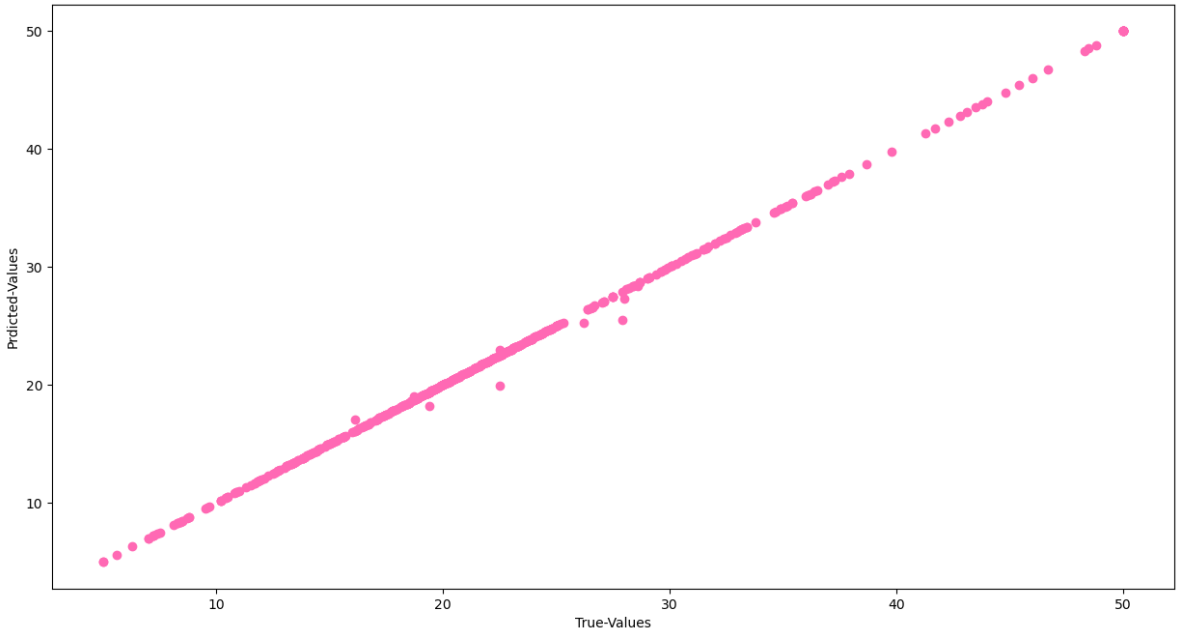
In [16]:

```
import matplotlib.pyplot as plt
plt.scatter(data_point_i[:,2],actual_y,color = 'green',alpha=0.4)
plt.scatter(data_point_i[:,2],predicted_y,color = 'red',alpha=0.4)
plt.rcParams.update({'figure.figsize':(15,8), 'figure.dpi':100})
plt.show()
```



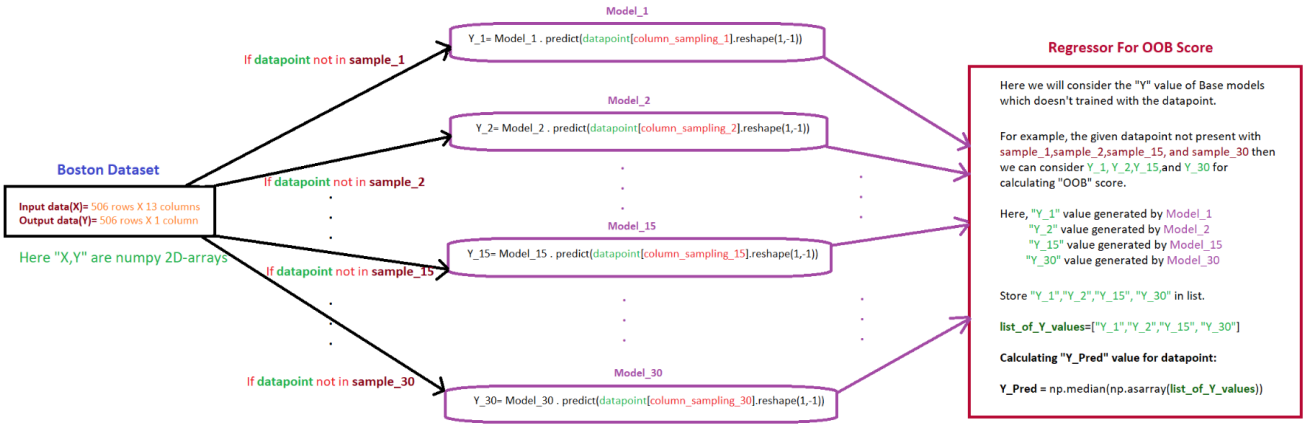
In [17]:

```
import matplotlib.pyplot as plt
plt.scatter(actual_y,predicted_y,color = 'hotpink')
#plt.scatter(data_point_i[:,4],predicted_y,color = 'red')

plt.xlabel("True-Values")
plt.ylabel("Prdicted-Values")
plt.rcParams.update({'figure.figsize':(15,8), 'figure.dpi':100})
plt.show()
```



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 [18]:

```
y_predicted_oob_median_list = []
y_predicted_oob_list = []

for i in range(0, 506):
    indices_for_oob_models = []

  # For each of i-th row I shall build a list, of sample size 30
  # ONLY condition being that this i-th row should not be part of the list_selected_row[i-t
  # e.g. say for i = 469 and index_oob in below loop is 10 then
  # list_selected_row[10] (which is an array of row-numbers) should not contain the 469-th
    for index_oob in range(0, 30):
        if i not in list_selected_row[index_oob]:
            indices_for_oob_models.append(index_oob)

    for oob_model_index in indices_for_oob_models:
        model_oob = list_of_all_models_decision_tree[oob_model_index]
        row_oob = x[i]
        # print('oob_model_index ', oob_model_index)

        # Now extract ONLY those specific columns/featues that were selected during the boo
        x_oob_data_point = [row_oob[columns] for columns in list_selected_columns[oob_model
        # print('np.array(x_oob_data_point) ', np.array(x_oob_data_point))
        x_oob_data_point = np.array(x_oob_data_point).reshape(1, -1)

        y_predicted_oob_data_point = model_oob.predict(x_oob_data_point)
        y_predicted_oob_list.append(y_predicted_oob_data_point)

        y_predicted_median = np.median(y_predicted_oob_list)
        y_predicted_oob_median_list.append(y_predicted_median)
```

In [19]:

```
def calculate_oob_score(num_rows):
    oob_score = 0
    for i in range(0, num_rows):
        oob_score += ((actual_y[i] - y_predicted_oob_median_list[i] ) ** 2)
    final_oob_score = oob_score/506
    return final_oob_score

print("final_oob_score is ", calculate_oob_score(506))
```

```
final_oob_score is  91.45892786561274
```

Task 2

In [20]:

```
# Function to build the entire bootstrapping steps that we did above and
# Reurning from the function the MSE and oob score
def bootstrapping_and_oob(x, y):

  # 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)

  # building regression trees
  list_of_all_models_decision_tree = []
  for i in range(0, 30):
    model_i = DecisionTreeRegressor(max_depth=None)
    model_i.fit(list_input_data[i], list_output_data[i])
    list_of_all_models_decision_tree.append(model_i)

  # calculating MSE
  array_of_Y = []

  for i in range(0, 30):
    data_point_i = x[:, list_selected_columns[i]]
    target_y_i = list_of_all_models_decision_tree[i].predict(data_point_i)
    array_of_Y.append(target_y_i)


  predicted_array_of_target_y = np.array(array_of_Y)
  predicted_array_of_target_y = predicted_array_of_target_y.transpose()

  # print(predicted_array_of_target_y.shape)

  # Now to calculate MSE, first calculate the Median of Predicted Y
  # passing axis=1 will make sure the medians are computed along axis=1
  median_predicted_y = np.median(predicted_array_of_target_y, axis=1)

  # And now the final MSE
  MSE = mean_squared_error(y, median_predicted_y )

  # Calculating OOB Score
  y_predicted_oob_median_list = []

  for i in range(0, 506):
    indices_for_oob_models = []

    # For each of i-th row I shall build a list of sample size 30
    # ONLY condition being that this ith row should not be part of
    # the list_selected_row
    for index_oob in range(0, 30):
      if i not in list_selected_row[index_oob]:
        indices_for_oob_models.append(index_oob)
```

```
    y_predicted_oob_list = []

    for oob_model_index in indices_for_oob_models:
      model_oob = list_of_all_models_decision_tree[oob_model_index]

      row_oob = x[i]
      # print('oob_model_index ', oob_model_index)

      x_oob_data_point = [row_oob[col] for col in list_selected_columns[oob_model_index] ]
      # print('np.array(x_oob_data_point) ', np.array(x_oob_data_point))
      x_oob_data_point = np.array(x_oob_data_point).reshape(1, -1)

      y_predicted_oob_data_point = model_oob.predict(x_oob_data_point)
      y_predicted_oob_list.append(y_predicted_oob_data_point)
      #
    y_predicted_oob_list = np.array(y_predicted_oob_list)

    y_predicted_median = np.median(y_predicted_oob_list)
    y_predicted_oob_median_list.append(y_predicted_median)


  oob_score = 0

  for i in range(0, 506):
    # oob_score = (oob_score + (y[i] - y_predicted_oob_median_list[i] ) ** 2)
    # 13.828377285079045
    oob_score += (y[i] - y_predicted_oob_median_list[i] ) ** 2

  final_oob_score = oob_score/506

  return MSE, final_oob_score

print(bootstrapping_and_oob(x,actual_y))
```

```
(0.08141729753232378, 16.38836025487383)
```

In [21]:

```
import scipy

x=boston.data #independent variables
y=boston.target #target variable

mse_boston_35_times_arr = []
oob_score_boston_35_times_arr = []

# Repeat Task 1 for 35 times, and for each iteration store the Train MSE and OOB score
for i in range(0, 35):
  mse, oob_score = bootstrapping_and_oob(x, y)
  mse_boston_35_times_arr.append(mse)
  oob_score_boston_35_times_arr.append(oob_score)


mse_boston_35_times_arr = np.array(mse_boston_35_times_arr)
oob_score_boston_35_times_arr = np.array(oob_score_boston_35_times_arr)

mean_of_sample_mse_35 = np.mean(mse_boston_35_times_arr)
standard_error_of_sample_mse_35 = scipy.stats.sem(mse_boston_35_times_arr)
```

In [22]:

```
print(mse_boston_35_times_arr)
print("*"*20)
print(oob_score_boston_35_times_arr)
```

```
[0.11761913 0.05154644 0.02628953 0.05737772 0.03786561 0.00314229
 0.10498518 0.06372036 0.06819417 0.15317128 0.15047925 0.0795423
 0.03445268 0.04687253 0.39506786 0.11894873 0.01818182 0.07028911
 0.19772069 0.02292767 0.41246449 0.1993083  0.04501661 0.18197628
 0.10956947 0.03342633 0.02331028 0.09009881 0.05320652 0.03405402
 0.12573862 0.0650446  0.21073205 0.00989625 0.1201    ]
********************
[12.8537358  13.2128674  15.24908559 17.59125879 11.36601504 14.53082016
 13.19997912 18.05556708 16.73239583 15.60467364 12.62311045 12.36024641
 16.12179109 12.55705918 14.09337785 14.17654539 16.24837945  8.86945899
 13.67446957 16.2917308  16.66246206 14.90867592 14.0478831  14.94037549
 16.96537281 13.16012309 14.92570652 13.4835919  12.53979551 11.87414416
 13.75985288 12.24660925 13.88236007  9.35620059 10.56190563]
```

In [23]:

```
print(mean_of_sample_mse_35)
print("*"*20)
print(oob_score_boston_35_times_arr)
```

```
0.10092391315636505
********************
[12.8537358  13.2128674  15.24908559 17.59125879 11.36601504 14.53082016
 13.19997912 18.05556708 16.73239583 15.60467364 12.62311045 12.36024641
 16.12179109 12.55705918 14.09337785 14.17654539 16.24837945  8.86945899
 13.67446957 16.2917308  16.66246206 14.90867592 14.0478831  14.94037549
 16.96537281 13.16012309 14.92570652 13.4835919  12.53979551 11.87414416
 13.75985288 12.24660925 13.88236007  9.35620059 10.56190563]
```

In [24]:

```
#Finding Confidence Level of MSE
sample_mean = mse_boston_35_times_arr.mean()
sample_std =  mse_boston_35_times_arr.std()
sample_size = len(mse_boston_35_times_arr)
# here we are using sample standard deviation instead of population standard deviation
left_limit  = np.round(sample_mean - 2*(sample_std/np.sqrt(sample_size)), 3)
right_limit = np.round(sample_mean + 2*(sample_std/np.sqrt(sample_size)), 3)
print('95% Confidence Interval of MSE is', left_limit, 'to', right_limit)
```

```
95% Confidence Interval of MSE is 0.069 to 0.133
```

In [25]:

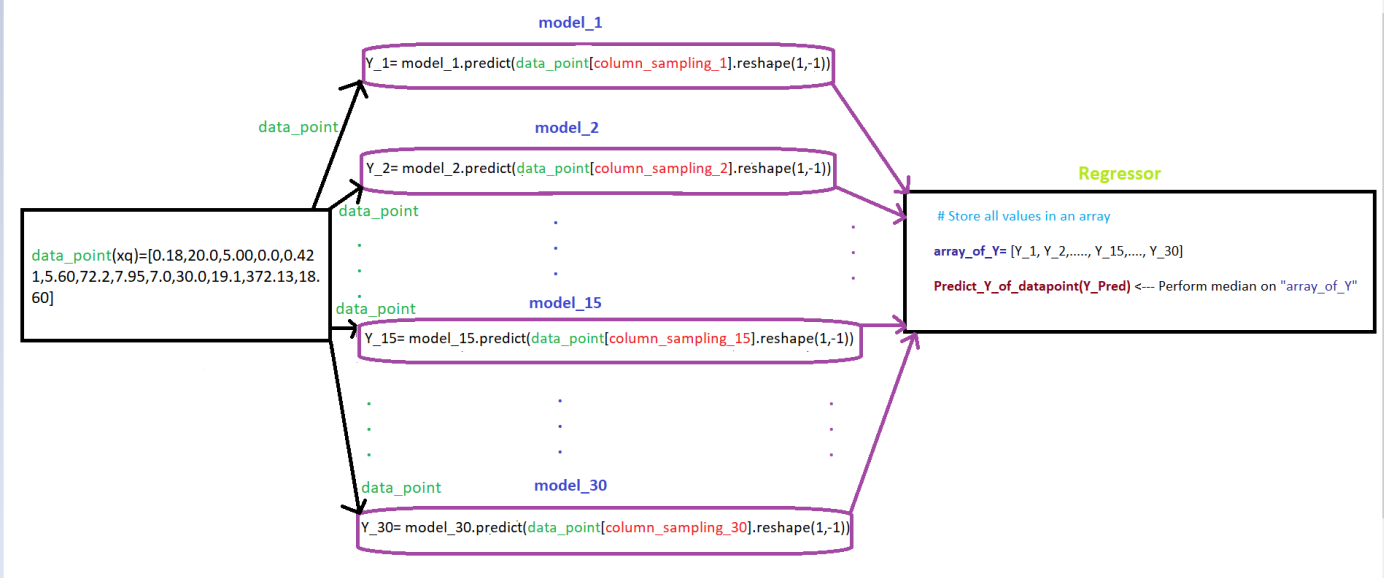
```
#Finding Confidence Level of OOB
sample_mean = oob_score_boston_35_times_arr.mean()
sample_std =  oob_score_boston_35_times_arr.std()
sample_size = len(oob_score_boston_35_times_arr)
# here we are using sample standard deviation instead of population standard deviation
left_limit  = np.round(sample_mean - 2*(sample_std/np.sqrt(sample_size)), 3)
right_limit = np.round(sample_mean + 2*(sample_std/np.sqrt(sample_size)), 3)
print('95% Confidence Interval of OOB is', left_limit, 'to', right_limit)
```

```
95% Confidence Interval of OOB is 13.238 to 14.689
```

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 [26]:

```
def predict_y_given_x_bootstrap(x_query):
  y_predicted_array_30_sample = []

  for i in range(0, 30):
    model_i = list_of_all_models_decision_tree[i]
    # Extract x for ith data point with specific number of featues from list_selected_colum
    x_data_point_i = [x_query[column] for column in list_selected_columns[i]]
    x_data_point_i = np.array(x_data_point_i).reshape(1, -1)
    y_predicted_i = model_i.predict(x_data_point_i)
    y_predicted_array_30_sample.append(y_predicted_i)

  y_predicted_array_30_sample = np.array(y_predicted_array_30_sample)
  y_predicted_median = np.median(y_predicted_array_30_sample)
  return y_predicted_median


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]
y_predicted_for_xq = predict_y_given_x_bootstrap(xq)
print("Predicted value of House Price",y_predicted_for_xq)
```

```
Predicted value of House Price 18.5
```

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

```
Task1:
As we know that Larger the MSE means large error in the model.The MSE and MAE are also
less in the results of task1 means training error is less so model works well for
training data effectively.
The final oob_score of Task1 is  91.45892786561274 means we can take it as model works
well for unseen data too.

Task2:
By definition we know the interpretation of a 95% confidence interval for the population
mean as - If repeated random samples were taken and the 95% confidence interval was
computed for each sample, 95% of the intervals would contain the population mean.
So in this case
MSE - There is a 95% chance that the confidence interval of (0.069, 0.133) contains the
true population mean of MSE.
OOB Score - There is a 95% chance that the confidence interval of (13.238, 14.689)
contains the true population mean of OOB Score.

Task3:
Given query point "xq" to 30 models and performed the regression on the output generated
by 30 models is 18.5
```