Machine Learning Assignment-1
Group Members:

Soni Aditya Bharatbhai (20CS10060)
Anand Manojkumar Parikh(20CS10007)

Approach used and assumptions made:

1. **Reading csv file and storing in numpy array**: We read data from the given csv file using the pandas library. Since the given dataset does not contain any missing attribute fields, we need not handle that. However some attributes in the dataset have values yes/no. So we first replace all occurrences of yes by 1 and all occurrences of no by 0 in the pandas dataframe. Now that we have numerical values in all attributes we use a 2d numpy array to store the dataset.
2. **Classes and their functions**:
    1. Node class: Class for the tree nodes. Since each node can be either a decision node or a leaf node we store these attributes in each Node object using a constructor (attributes a,b,c,d are used for a decision node and attribute e is used for a leaf node) :
        a. feature_index: index of feature the node tests
        b. threshold: threshold value of feature
        c. left: stores the left child of node
        d. right: stores the right child of node
        e. output_val: The value predicted by this node if it is a leaf node
    2. Tree class: Class for the regression tree. Attributes of each Tree object are root node and depth which are initialized by the constructor. All the member functions relating to Tree object are:
        a. train_data: Builds the regression tree by calling the build_tree function and then stores the root and depth of the tree.
        b. build_tree:
            i. A recursive function that builds the tree from the training samples and returns the root node.
            ii. If current samples are such that we can stop by making the current node as leaf node, we do so and return.
            iii. Else we find the best split of training samples by calling the find_best_split function.
            iv. We split the training samples into two parts based on the attribute and its threshold value.
            v. Then we recursively compute the left and right children of the current node.
        c. find_best_split:
            i. This function computes the best split in training data based on the sum of squared errors of output labels.
            ii. The approach is that we try all possible attributes. In order to find the possible thresholds for each attribute, what we do is that for

each attribute we sort the training examples based on that attribute's value. After this we consider the threshold values as the average of that particular attribute's value for every consecutive pair of training samples.

iii. We consider all possible attributes and all possible corresponding threshold values and calculate for each combination the sum of squared loss in the output labels. So we first compute the variance of output labels of training samples before splitting. Next we split the training samples into two based on this attribute and its threshold, and then we take the weighted average(weight is the number of training samples) of variance of the two splitted samples. The objective is to maximize the reduction in variance i.e maximize ((variance of samples before splitting) - (weighted average of variance of samples after splitting)).

iv. We choose that attribute, threshold combination for which we get maximum reduction in variance.

d. find_depth:
   i. Recursive function to compute depth of the tree. Depth of the tree is assumed to be the number of nodes in the tree. If the current node is leaf node, it returns 1. Else it returns 1 + (depth of left child) + (depth of right child).

e. predict_values:
   i. Given a set of test samples, this function computes the output labels predicted by the trained regression tree. It returns the array corresponding to the predicted output labels.

f. predict_y:
   i. Recursive function to find the output label of a single test sample.
   ii. If we reach leaf node, function returns output_value for the leaf node(base case).
   iii. Else we are at a decision node so we go to the left or right child depending on the attribute value of sample.

g. prune_tree:
   i. Recursive function to perform rule post pruning on the regression tree. Since we need to maintain tree structure after pruning operations, we follow a top-down approach while pruning.
   ii. The approach is that we try to make the current node as a leaf node by assigning this new leaf node's output value as the arithmetic mean of the output labels of the training samples which were initially splitted into two parts by the original decision node. We compute the test accuracy of this modified tree and compare it with the test accuracy before modification.
   iii. If test accuracy improves then we prune this node and make it a leaf node.

iv. Else we restore the tree and then recursively try and prune the left and right subtrees.

h. find_error:
   i. Given an array of test samples with their actual output labels and another array containing the model's predicted values, the find_error function calculates the root mean square error (assumed root mean square error as a measure of accuracy) as (sum over all test samples ((yi - fi)^2)) / number of samples, where yi is actual output label and fi is predicted output label for each sample.

i. Print_tree:
   i. This function performs a breadth-first traversal over the regression tree and prints the tree in a level by level manner such that for each level the nodes are printed from left to right as they appear in the tree.

Command to run the code: **python3 q1.py > output_q1.txt**

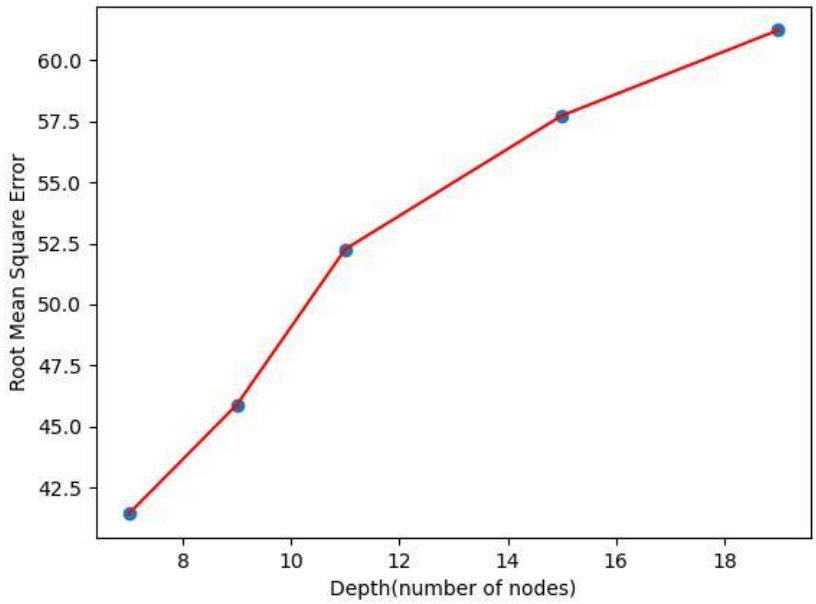
Since we have a random split of test and train data we will get different output each time we run the code. output.txt dynamically changes as we run the code.

Some outputs of our code(the plot of accuracy vs depth is shown as root mean square error vs depth, so higher the root mean square error, lower is the accuracy). This plot captures the change in error as depth changes during the pruning process and also includes the (error,depth) pair for initial overfitted tree before pruning. Since during pruning we prune a node only if test accuracy improves, the root mean square error will reduce as depth of tree reduces. Since the depth of the tree and the error always reduces during pruning ,the model gradually generalizes during pruning and the overfitting gradually reduces as pruning proceeds. So the depth of the tree for which the model overfits is essentially the initial depth of the tree we obtain from the model before performing any pruning operations.

Sample Output 1:
Tree obtained:
Level 0 Extra Spicy = yes
Level 1 Size by Inch <= 10.5 Extra Mushroom = yes
Level 2 Extra Mushroom = yes Size by Inch <= 13.5 Extra Cheeze = yes Extra Cheeze = yes
Level 3 Extra Cheeze = yes 550 Extra Mushroom = yes 700 700 750 900.0 950
Level 4 575.0 650.0 700 650

Test Error = 61.237243569579455

Depth of tree = 19
Depth of tree for which the model overfits = 19
Pruned Tree:
Level 0 Extra Spicy = yes
Level 1   Size by Inch <= 10.5   Extra Mushroom = yes
Level 2 600.0 683.3333333333334 725.0   912.5
Test error for pruned tree = 41.422898033352034
Depth of the pruned tree = 7



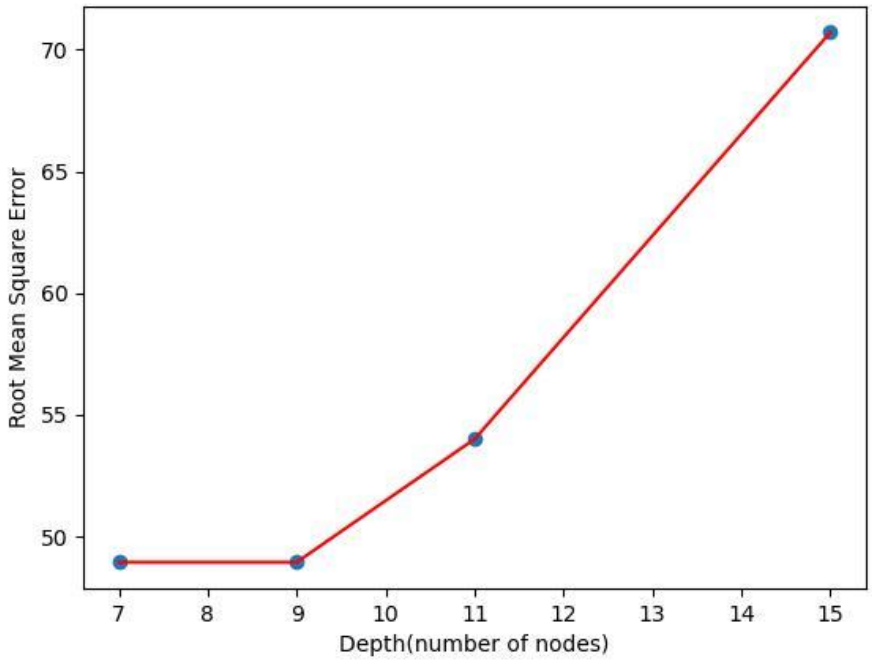
Sample Output 2:

Tree obtained:
Level 0 Extra Spicy = yes
Level 1   Size by Inch <= 10.5   Extra Mushroom = yes
Level 2 Extra Mushroom = yes   Extra Cheeze = yes   700.0   Extra Cheeze = yes
Level 3 Extra Cheeze = yes 550   700   650   900.0   925.0
Level 4 575.0 700

Test Error = 70.71067811865476

Depth of tree = 15
Depth of tree for which the model overfits = 15
Pruned Tree:

Level 0 Extra Spicy = yes
Level 1   Size by Inch <= 10.5   Extra Mushroom = yes
Level 2 600.0 675.0   700.0   910.0
Test error for pruned tree = 48.947250518628046
Depth of the pruned tree = 7

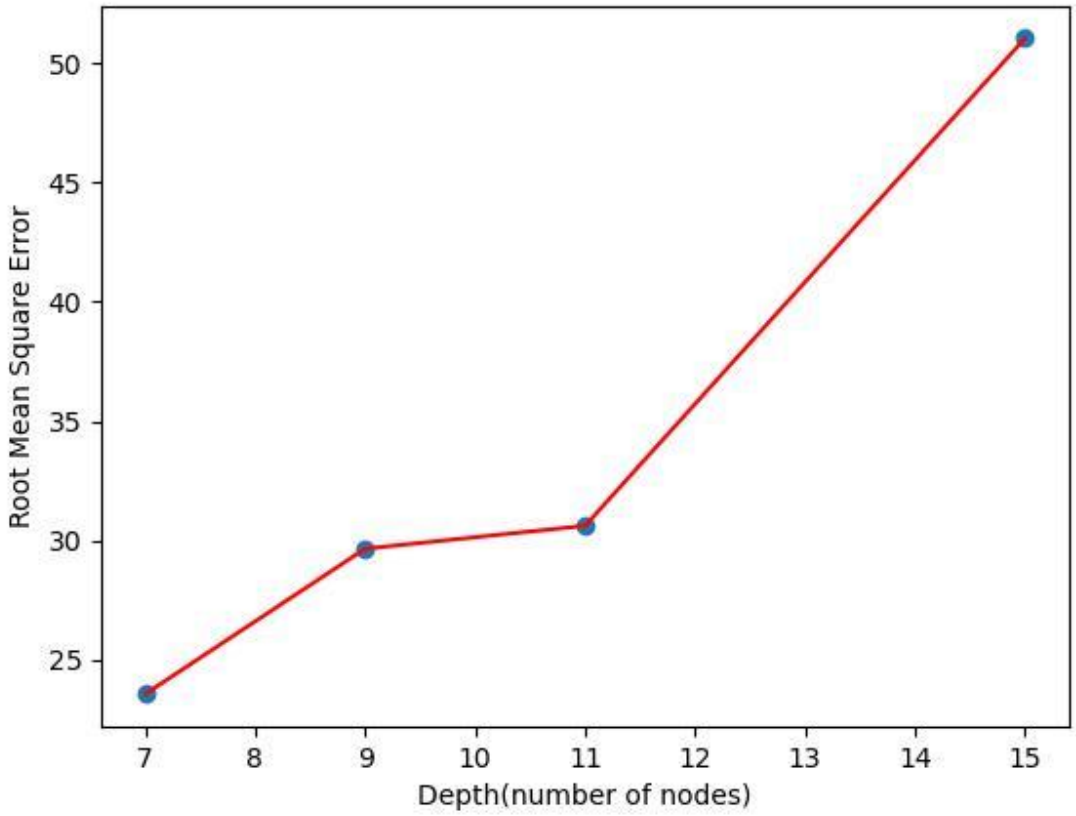


Sample Output 3:
Tree obtained:
Level 0 Extra Spicy = yes
Level 1   Size by Inch <= 10.5   Extra Mushroom = yes
Level 2 Extra Mushroom = yes   700.0   Size by Inch <= 10.5 Extra Cheeze = yes
Level 3 Extra Cheeze = yes 575.0   700   725.0   900.0   950
Level 4 575.0 700

Test Error = 51.03103630798287

Depth of tree = 15
Depth of tree for which the model overfits = 15
Pruned Tree:
Level 0 Extra Spicy = yes
Level 1   Size by Inch <= 10.5   Extra Mushroom = yes

Level 2 600.0 700.0   716.6666666666666 916.6666666666666
Test error for pruned tree = 23.57022603955157
Depth of the pruned tree = 7



Sample Output 4:

Tree obtained:
Level 0   Size by Inch <= 10.5
Level 1 Extra Mushroom = yes Extra Mushroom = yes
Level 2 Extra Cheeze = yes 575.0 Extra Cheeze = yes Extra Spicy = yes
Level 3 575.0 650.0 700 725.0 Extra Cheeze = yes Extra Cheeze = yes
Level 4 700 750 900.0 950

Test Error = 54.962108159470496

Depth of tree = 17
Depth of tree for which the model overfits = 17
Pruned Tree:

Level 0 Size by Inch <= 10.5
Level 1 Extra Mushroom = yes Extra Mushroom = yes
Level 2 Extra Cheeze = yes 575.0 716.6666666666666 Extra Spicy = yes
Level 3 575.0 650.0 725.0 916.6666666666666
Test error for pruned tree = 39.96526269427265
Depth of the pruned tree = 11