Pattern Recognition and Machine Learning

Lab - 2 Assignment

Akriti Gupta
B21AI005

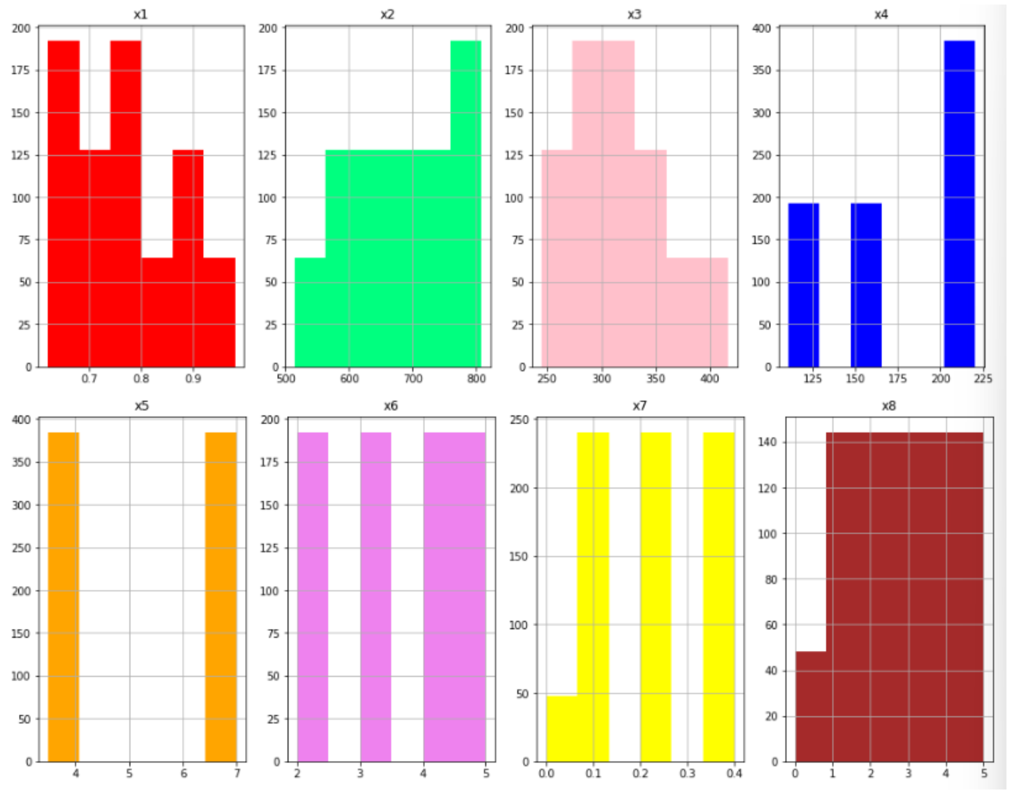
Question 1.

Firstly, we import all the libraries then import the energy analysis dataset consisting of 768 rows and 9 columns. Therefore, we have 8 feature attributes.

Q1.

Preprocessing and visualization:

We first check if there are any missing/null values in the dataset. We observe that there are no null values so next we check the datatype of each column and conclude that there is no need to to categorical encoding as no object type column is present. Next we describe the data and visually analyze the distribution of each column using histogram as follows:

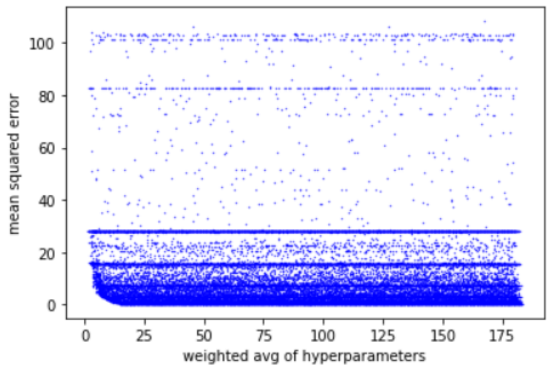


lastly we scale the features using MinMaxScalar().
Splitting the dataset:
We split the dataset into training validating and testing in the ratio 70:10:20 using the train_test_split function of sklearn.

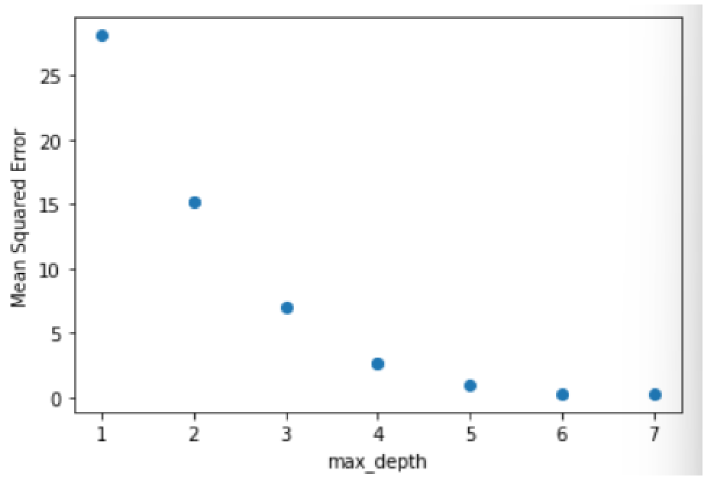
Q2.
We train the model using the DecisionTreeRegressor() function of sklearn.trees with the help of training dataset. Now to get the best hyper parameters, we write a function to first individually find the best value of each parameter. We use the hyperparameters max_depth,max_leaf_nodes and max_features. The thought process behind varying theses parameters is as follows:
Max_depth: If adjusted, we could reduce overfitting and halt the tree from growing in an unnecessary manner.
max_leaf_nodes:If selected correctly , can result in smaller divides, allowing us to keep overfitting under control.
Max_features: If modified, this could also aid in lowering the overfitting.

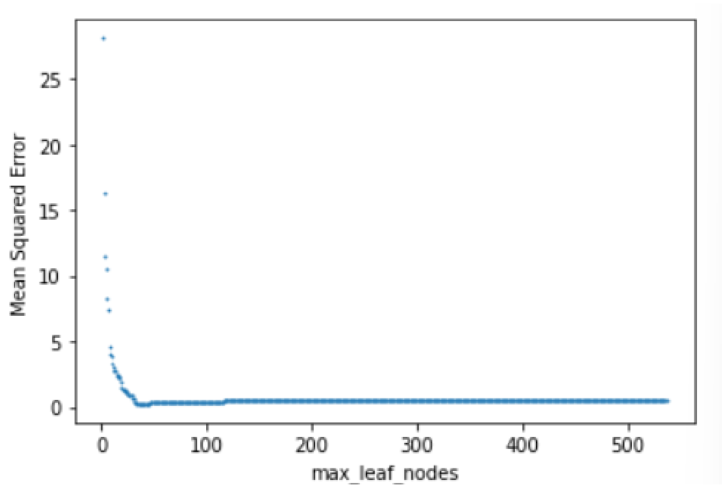
So individually the best score on validation data comes with max_depth=6,max_features=7 and max_leaf_nodes=45.

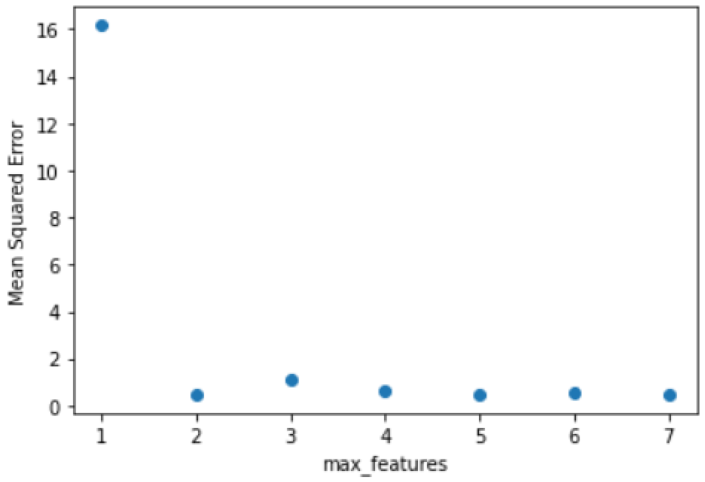
Now we vary all these hyperparameters together to find the best value for each in order to find the tree that generalizes best by using nested for loops and combining all of them together. The best score i.e 99.86 % on validation data comes with max_depth=6, max_features=6 and max_leaf_nodes=532.
Now, we plot the mse w.r.t the different hyperparameters and get the below plot:



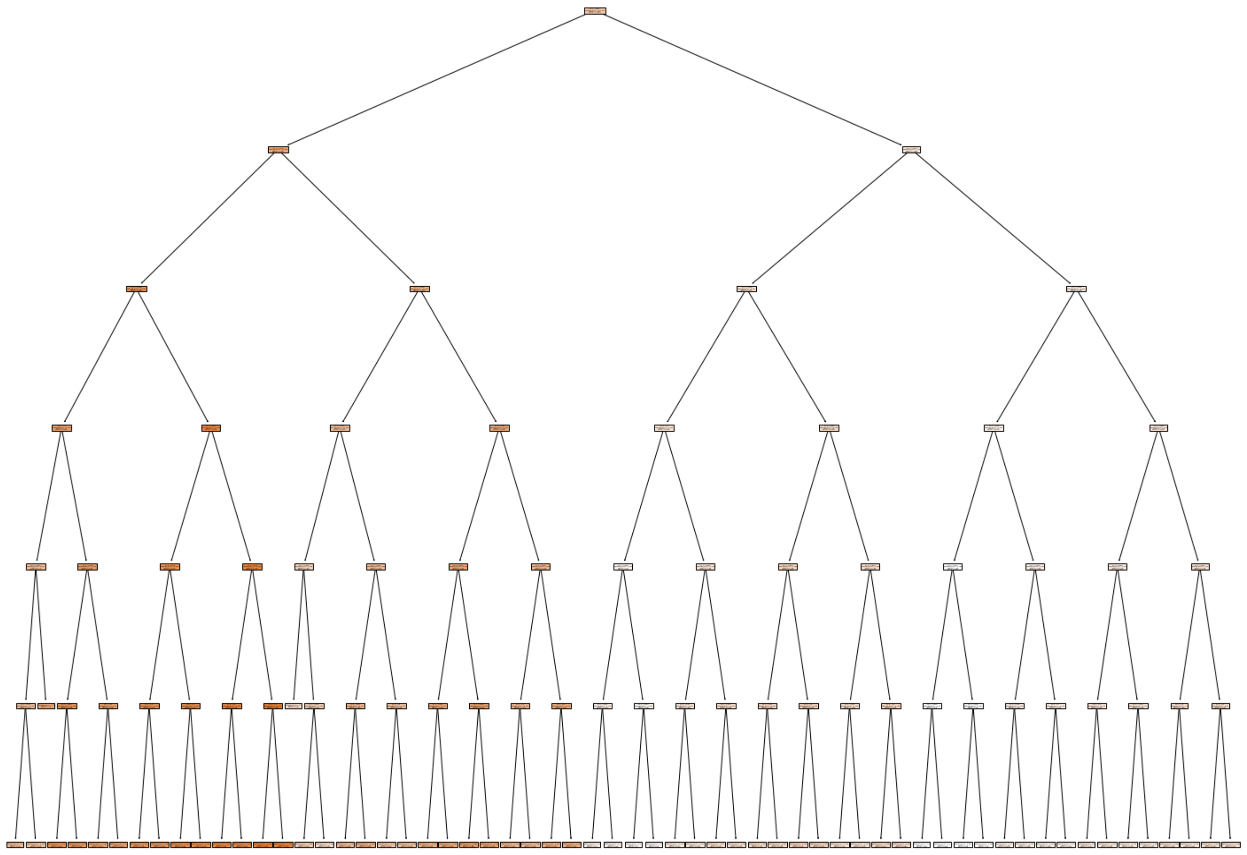
After that, we plot the errors and hyper-parameters while separately altering the hyper-parameters to get the score. The following plots appear:





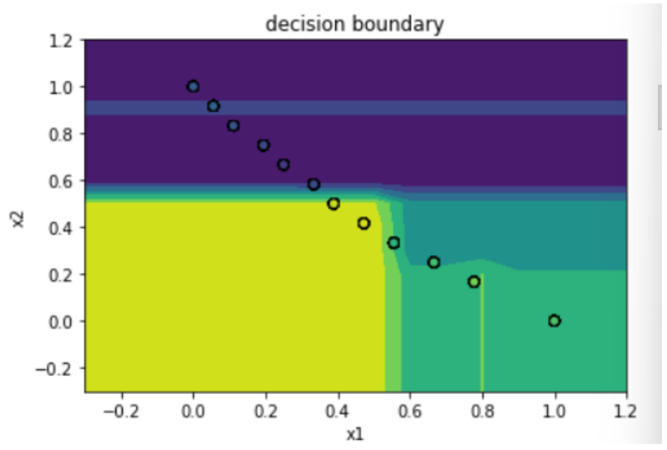


Q3.
For this question, we train the model by using the optimal hyper-parameters decided in the previous question. Now, we perform hold-out cross validation and get the score as 0.992.
Next, we perform 5-fold cross-validation and get the score as 0.9930 and lastly we perform repeated 5-fold validation and get the score as 0.9933 on the testing dataset.
Finally, the mean squared error between the predicted and the ground-truth values in the test data on our best model is obtained as 0.80028782.
THe decision tree for the best model is as follows:

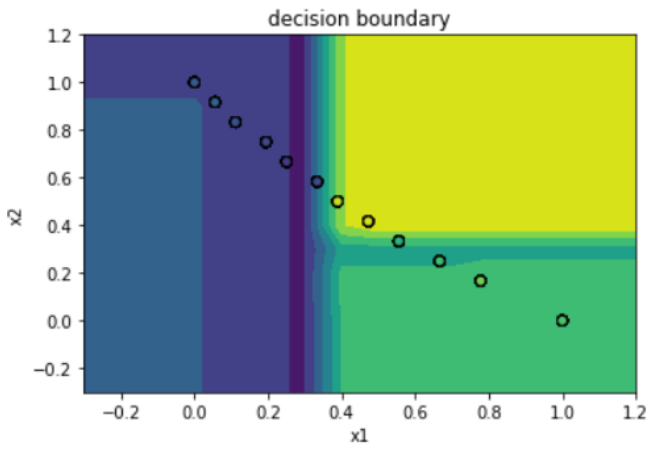


Q4.
<u>L1 criterion:</u>
“absolute_error” criterion is used for L1 regularization. We train the dataset using this criterion and get the L1-Regularization score as 99.603%.

The obtained decision boundary for this model using first two features is as follows:



<u>L2 criterion:</u>
“squared_error” criterion is used for L2 regularization. We train the dataset using this criterion and get the L2-Regularization score as 99.593%.
The obtained decision boundary for this model using first two features is as follows:



L1 regularization works better here as L2 regularization doesn’t perform feature selection, since weights are only reduced to values near 0 instead of 0. L1 regularization has built-in feature selection. Also, L1 regularization is robust to outliers, L2 regularization is not. L1 regularization penalizes the sum of absolute values of the weights, whereas L2 regularization penalizes the sum of squares of the weights.

Question 2.

Classification:

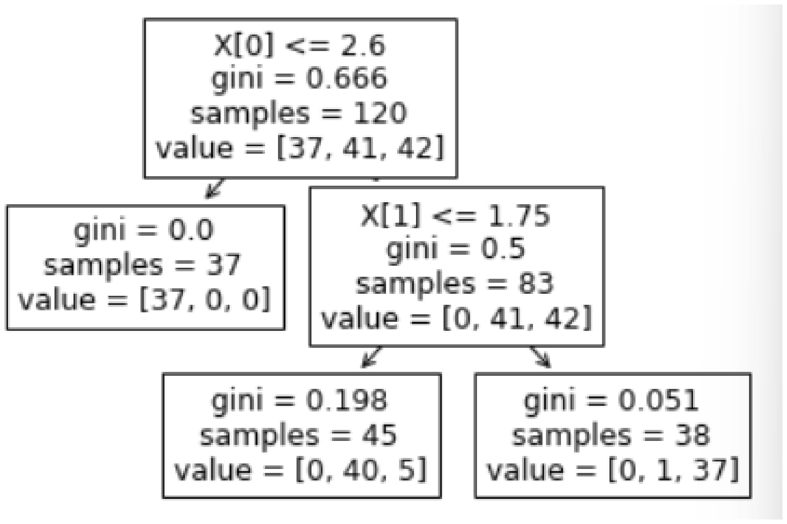
We firstly import the iris dataset and drop the attributes other than y Petal length and Petal width. Now, we preprocess the dataset by first checking if there are any null values. As no null values exist so we proceed further. We now encode the species attribute of the dataset as follows:

```
'Iris-setosa'=0
'Iris-versicolor'=1
'Iris-virginica'=2
```

Now, we split the dataset into training and testing in the ratio 80:20.

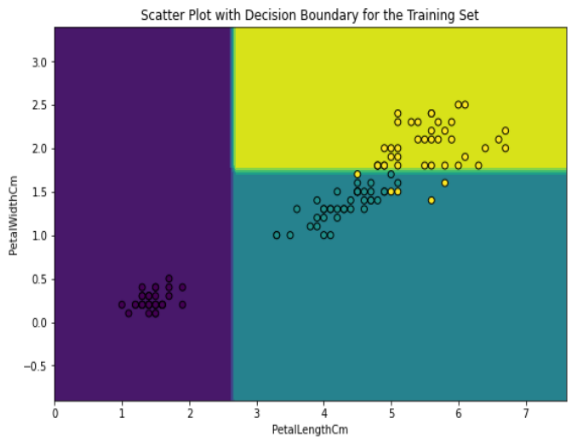
Q1.

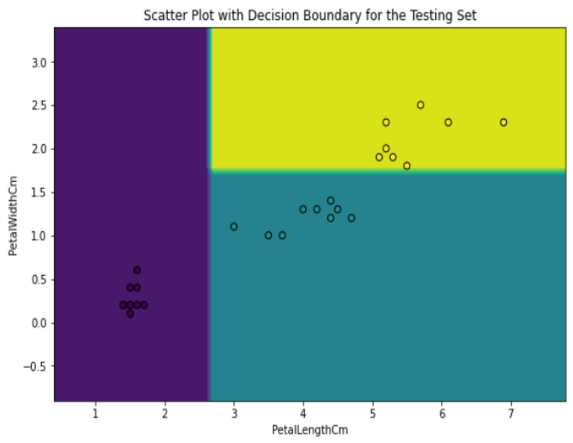
Now we train a DecisionTreeClassifier on the training dataset with hyperparameter max_depth=2. We get the tree as below:



We get the training accuracy of the model as 95% and testing accuracy as 100%.

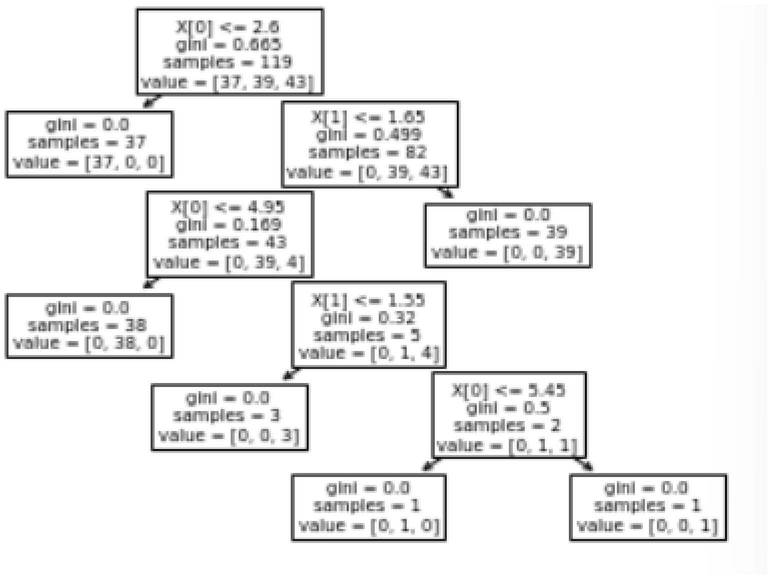
The decision_boundary plots with scatter plots of training and testing dataset we receive are as follows:



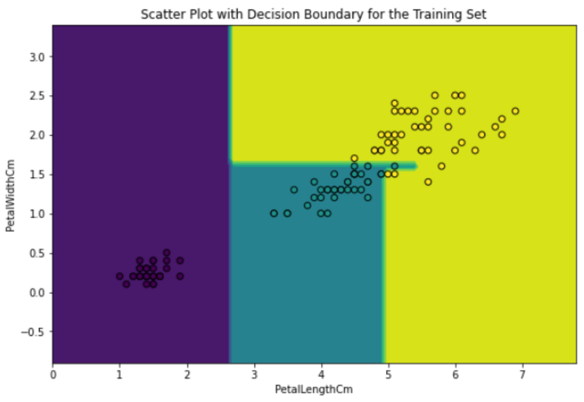


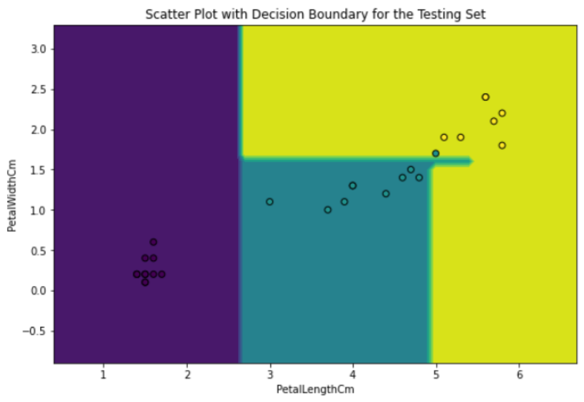
Q2.

Now, we remove the widest Iris Versicolor from the training dataset by dropping it from the dataset and train a new model on the obtained dataset. The tree recieved is as follows:



We get the training accuracy of the model as 100% and testing accuracy as 96.66%.The decision_boundary plots with scatter plots for training and testing dataset we receive are as follows:

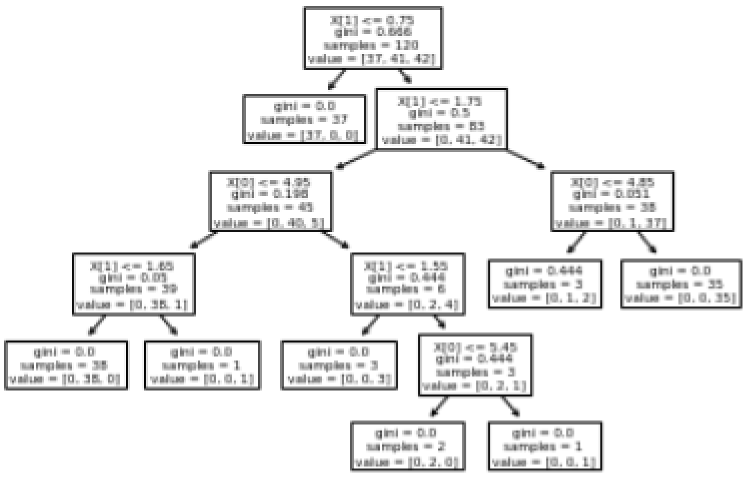




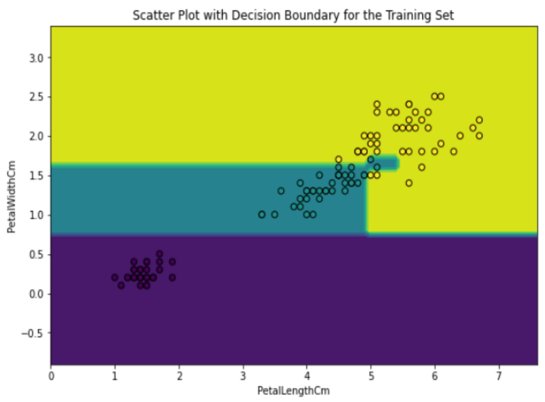
By comparing these plots with the previous ones, we can see that how by changing only one row of the dataset, the decision boundary changes drastically. So, we can say that decision trees are highly sensitive.

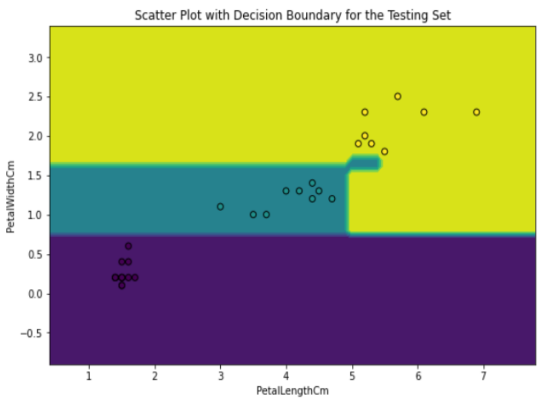
Q3.

Now we train a DecisionTreeClassifier on the training dataset with hyperparameter max_depth=None. We get the tree as below:



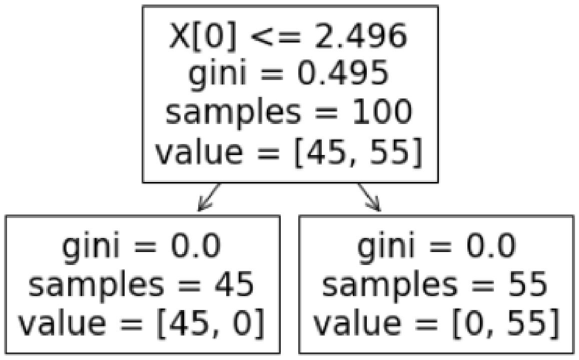
We get the training accuracy of the model as 99.16% and testing accuracy as 100%.
The decision_boundary plots with scatter plots for training and testing dataset we receive are as follows:



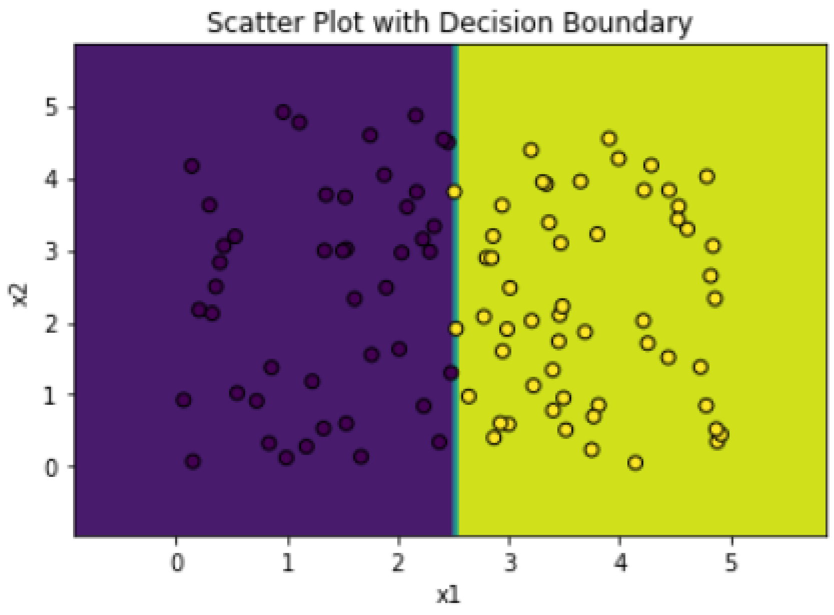


By comparing and analyzing the results of this model with that in part1, we can say that in this model when we take max_depth as None then the tree continues to grow unless it overfits the dataset and hence the accuracy for training dataset in this model is close to 100%. The decision boundary obtained for this model is also complex due to its overfitting nature.
So, max_depth hyper-parameter helps to reduce overfitting.

Q4.
To create a random dataset for this ques, we use the uniform() method of the random library which returns a random floating number between two specified numbers(0,5 in this case). We create 2 lists for the two attributes X1,X2 by using this method and created a dataframe by combing both the lists. Next we created the class attribute Y by checking the value of X1. When X1<2.5, Y=0 and when X1>2.5 then Y=1.
Next, we trained the dataset using DecisionTreeClssifier() function of sklearn with hyper-parameter max_depth=2. We received the tree as follows:



The decision boundary received is as follows:

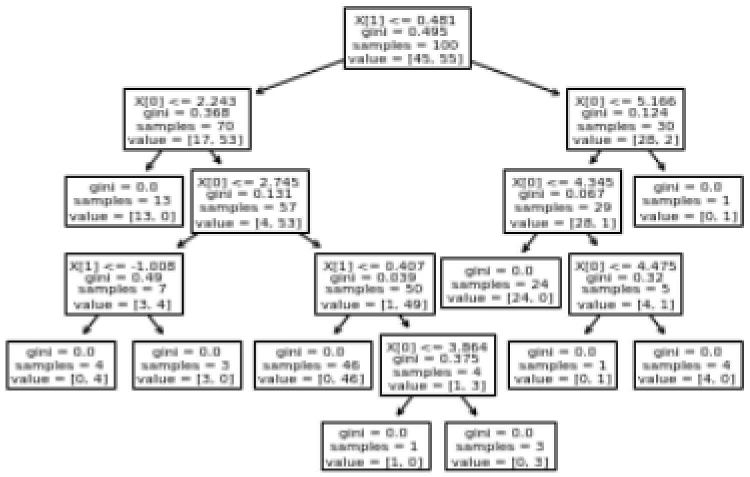


Now, we rotate the datapoints by 45 degrees in clockwise direction by using the formulas:

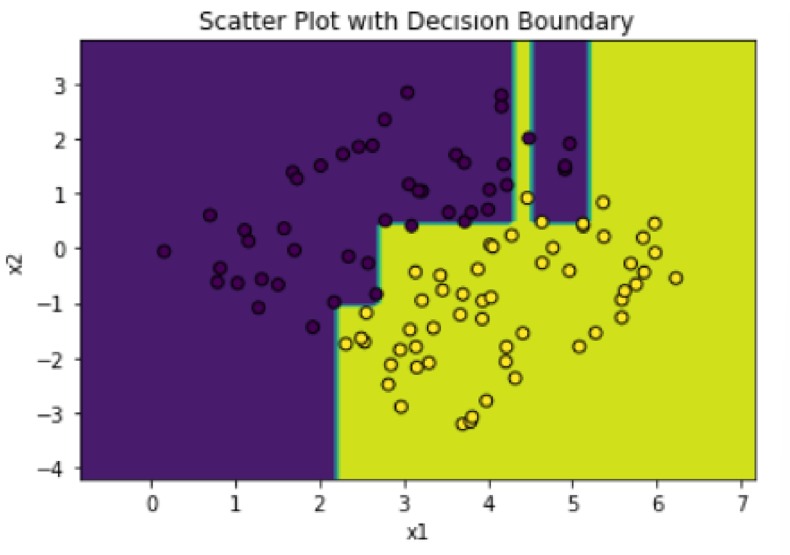
x=(x+y)/√ 2

y=(y-x)/√ 2

So we get a new dataframe and train a new classifier using this. The tree we obtain is as follows:



The decision boundary we received ia as follows:

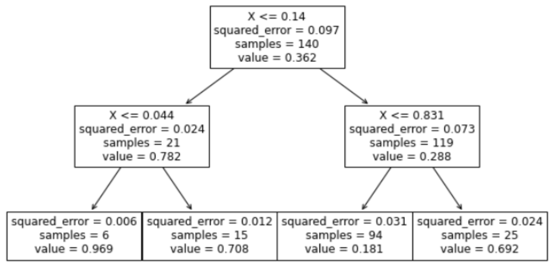


We observe that as we rotate the data points, the decision boundary obtained also changes accordingly. Now it has become more complex.

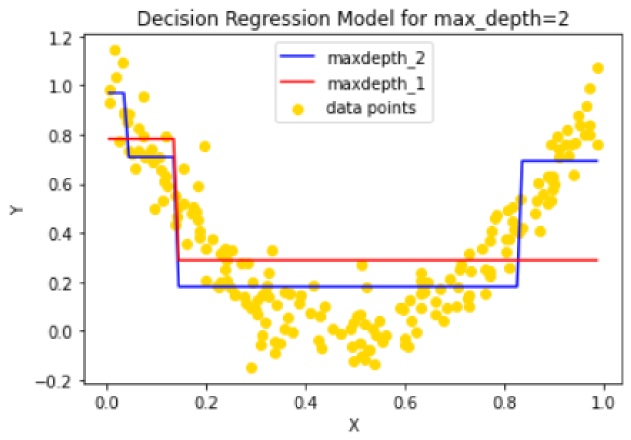
Q5.
It is evident from part 2,3,4 that even slight modifications to the data and hyper-parameters can cause significant changes to the decision tree, indicating that the algorithm is extremely responsive to changes in the data and tends to overfit by attempting to fit every point in the training data.

Regression:
We firstly import the task dataset. Now, we preprocess the dataset by first checking if there are any null values. As no null values exist so we proceed further. We split the dataset into training and testing in the ratio 70:30.

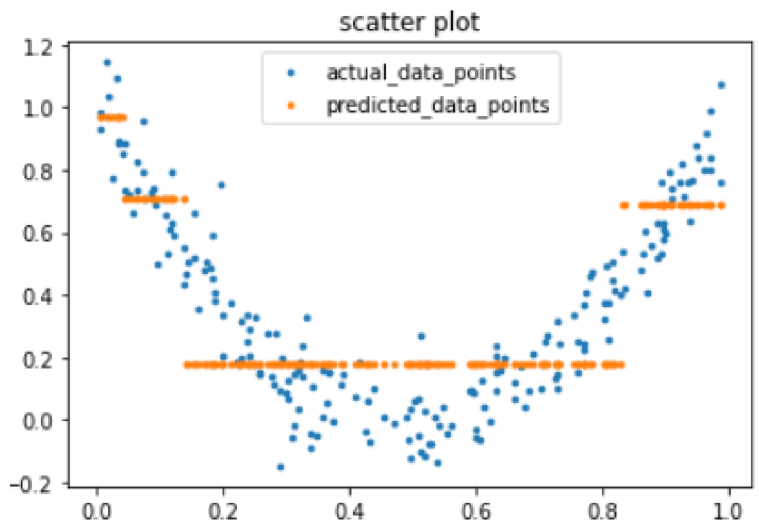
Q1.
Now, we train a decision tree model for max_depth=2. We obtain the tree as follows:



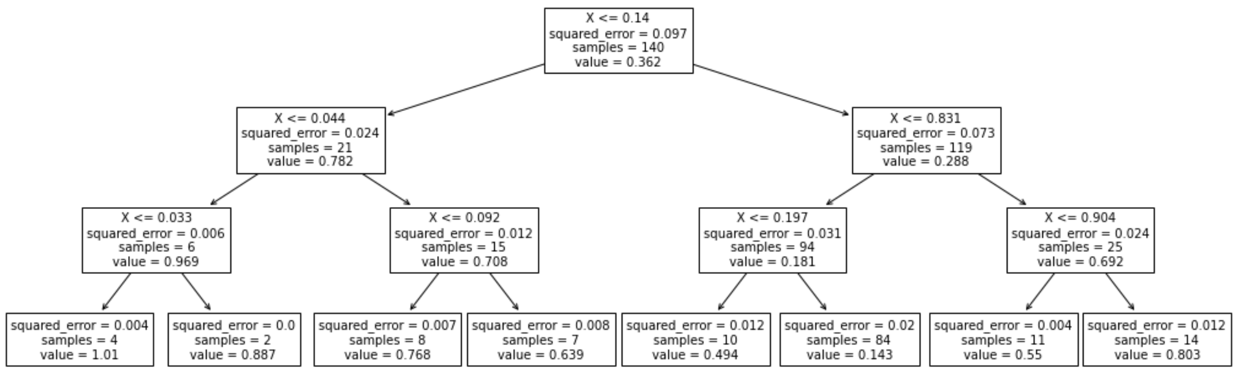
Next, we plot the regression predictions at each depth i.e. 1 and 2 using line plot:



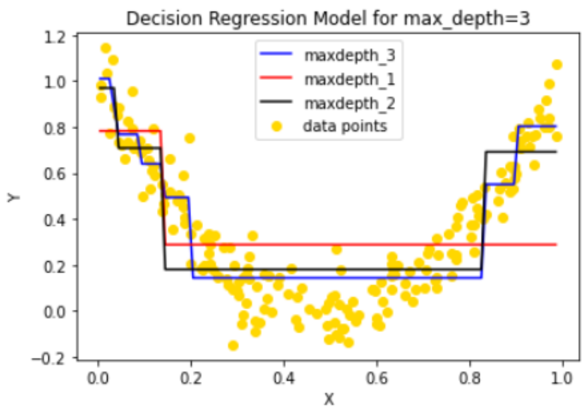
Also, we make scatter plot of the actual and predicted datapoints:



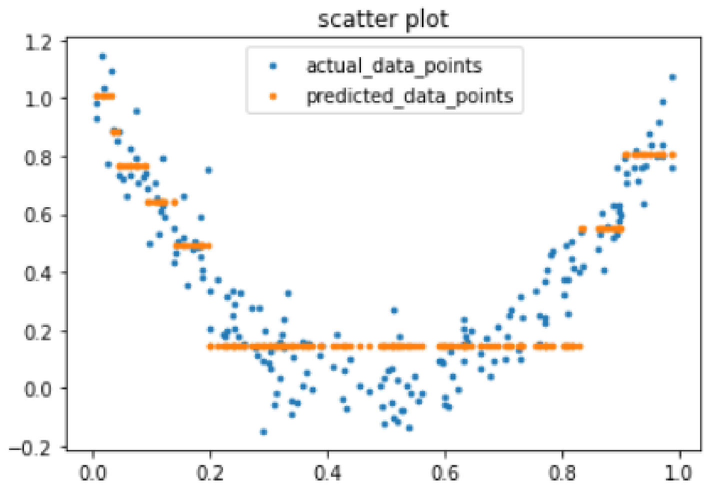
Now, we train a decision tree model for max_depth=3. We obtain the tree as follows:



Next, we plot the regression predictions at each depth i.e. 1 and 2 using line plot:



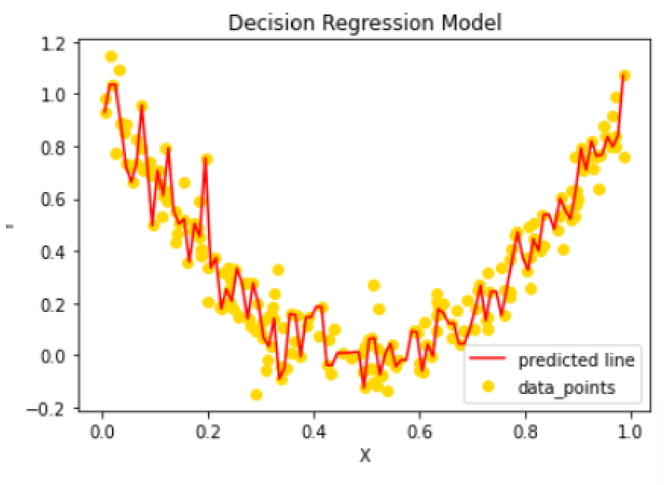
Also, we make scatter plot of the actual and predicted datapoints:



The testing score for max_depth=2 model is less than that of max_depth=3. It is apparent that by increasing the max_depth of our model by 1, there was a noticeable improvement in its performance in the testing data, as also evidenced by the scatter plot. Also, from the line plot we can see that the on increasing max_depth the line plot fits are data more nicely.

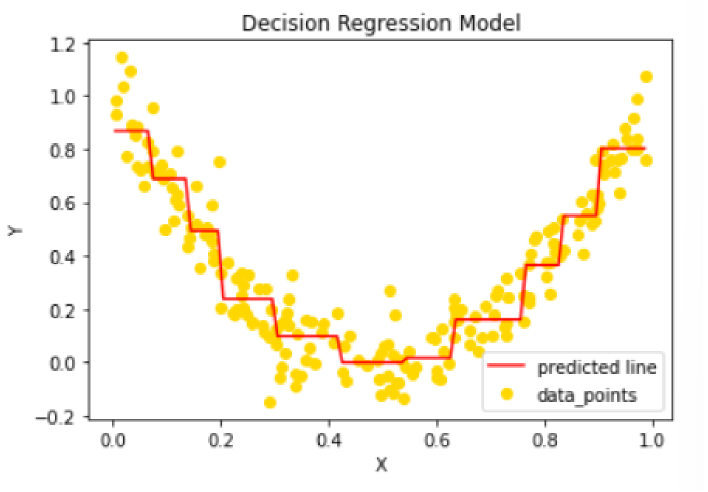
Q2.

We first train a model by taking min_samples_leaf = 0. We then plot a line graph for the model which we obtain as follows:



The training score of the model= 100%
The test score of the model= 82.44%

Now, we train another model by taking min_samples_leaf = 10. We then plot a line graph for the model which is as follows:



The training score of the model= 91.19%
The test score of the model= 85.84%

We can see that model trained by taking min_samples_leaf = 0, overfits the data while taking min_samples_leaf=10 we get a more smooth curve which also gives better testing score.

Question 3.
We import the penguins dataset.

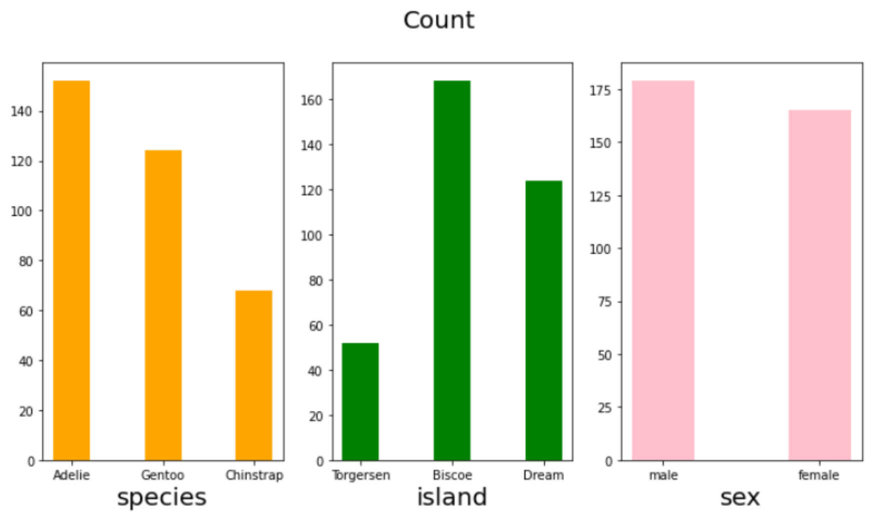
Q1.

We now preprocess the dataset by firstly checking if there are any null values present or not. We observe that the below given columns have null values:

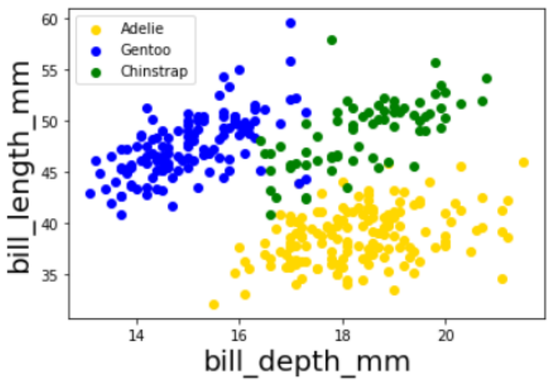
```
bill_length_mm        2
bill_depth_mm         2
flipper_length_mm     2
body_mass_g           2
sex                  11
```

As the first four columns are of float datatype. So we replace the null values with the mean. For the sex atrribute, we replace the null values with the most frequent one i.e male.

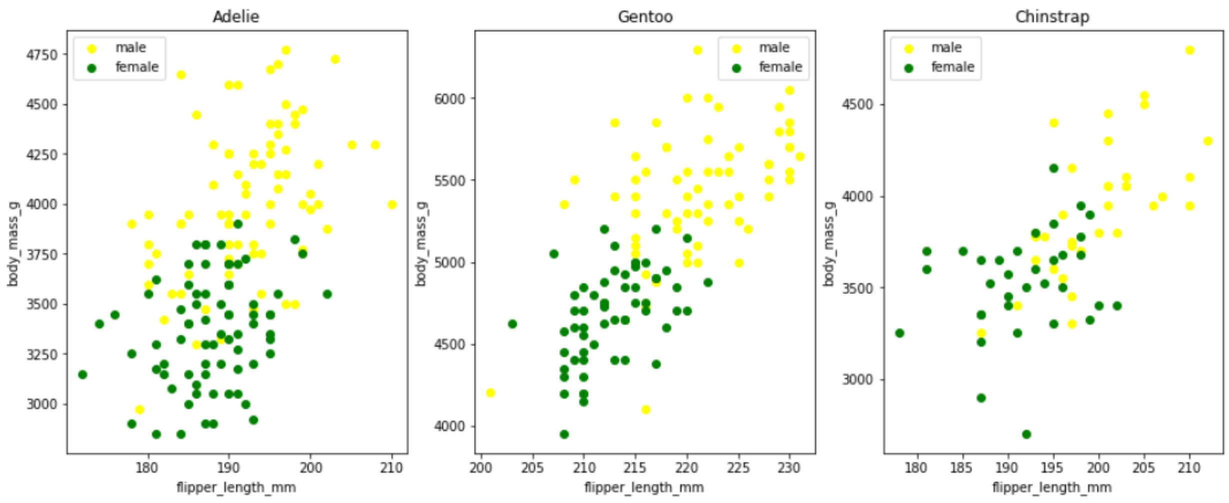
Next, we visualize the dataset by plotting histograms for categorical attributes:



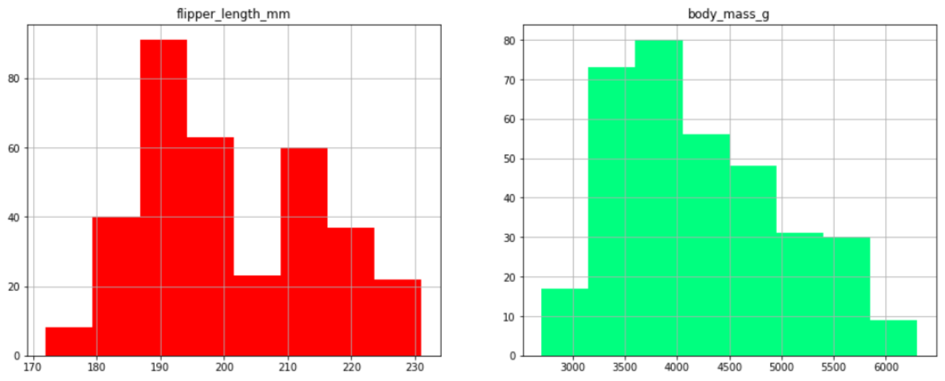
Then we plot the `bill_length_mm vs bill_depth_mm` scatter plot for different species:



Also, we plot the scatter plot of flipper length vs body mass for different species based on gender:



Lastly, we visualize body mass and flipper length distribution using histograms.



Now, we encode categorical attributes i.e `['species','island','sex','year']`
By using if else and replacing them with integers.
Lastly, we scale the features to make their weightage equal for developing a model.
Now, we split the dataset into training and testing in the ratio 70:30.

Q2.
To implement the cost function, we find gini index of each attribute using the formula

$$Gini = 1 - \sum_j p_j^2$$

Hence we calculate the probability of each class in the attribute and then find the gini index using the function written in the colab file.
For y_train we get gini index as 0.6371875.

Q3.
Now, in order to change continuous data to categorical, we write a function cont_to_cat in which we find the best threshold value for an attribute. If value is greater than threshold, we encode it as 1 else by 0. The threshold is obtained by finding which value provides minimum gini index.
To convert all features into binary form, we apply one-hot encoding to the features that have more than two values. This allows us to transform our decision tree into a binary tree.

Q4.
Now we create helper functions:
First we write a function called split which returns the giniIndex as well as the left side dataset and the right side dataset. Then, we create another function which gives us the best attribute for split i.e. the attribute which has minimum gini.

Q5.
Now, Our decision tree is represented by the class Node, which represents each node of the tree. Each node has the following attributes:

a) Data: the name of the attribute used for splitting/ class the test data belongs to.
b) df0: the dataset which goes to the left side and belongs has 0 value for the node on which split is based on.
c) df1: the dataset which goes to the right side and belongs has 1 value for the node on which split is based on.
d) Left: the name of the left child.
e) Right: name of the right child.

In the "tree" function, we utilize a recursive algorithm to construct a tree by following these base cases:

1. If all data points in a node belong to a single class, the node is made a leaf node with a label equal to the class of all data points in that node, and the tree is not further expanded.
2. If the height of the tree exceeds the "max_height" parameter passed to the function, the node is made a leaf node with a label equal to the majority class label of the data points in that node, and the tree is not further expanded.
3. If the gini index becomes same for all features, we stop expanding the tree and make that node a leaf node with a label equal to the majority class label of the data points in that node.

Next, we evaluate all features by dividing the data set into two parts based on the value of each feature. The feature that results in the minimum gini index is selected as the "best feature," and the dataset is split into two parts based on the value of this feature. The left subtree and right subtree data frames are recursively called, and the recursion returns the root node of each subtree. Finally, we assign the left subtree root node to the "left" attribute of the root node and the right subtree root node to the "right" attribute of the root node.
Now we print the tree by using the function traverse in which we traverse to each node and print the data of the node along with its left and right child nodes.
Now, we train our dataset using this function and by taking max_depth=20.

Q6.
Now, we write a function to classify the test data. This is done by recursively visiting each node using features of the testing data and do this until we reach a leaf node which tells us the class, the test data belongs to. This is how we predict the output label of a test data.

Q7.
Here, we calculate the overall accuracy and class wise accuracy by creating a function which first creates the confusion matrix and then uses the below mentioned formulas:

Accuracy= **(TP+TN)/(TP+TN+FP+FN)**
Class-wise Accuracy= **((TP/(TP+FN))+ (FP/(FP+TN)))/2.**
We get the overall accuracy for the testing dataset as 98.0769% and the class-wise accuracy as 97.62411%.