Lab 2B Report

Nakkina Vinay (B21AI023)

Question 1

Subpart 1 : Pre-processing the dataset and splitting it

- First using the read_csv function reading the dataset and using the head()

	X1	X2	X3	X4	X5	X6	X7	X8	Y1
0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55
1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55
2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55
4	0.9	563.5	318.5	122.5	7.0	2	0.0	0	20.84

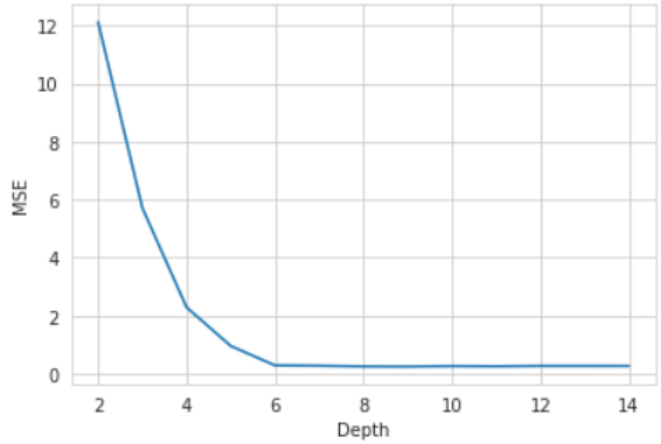
- The dataset was analyzed and it was a complete dataset with no missing values
- Dataset was observed carefully and it was found that X6 was a categorical label and therefore categorically encoded using get_dummies. X6 column was encoded and is replaced by 3,4,5 columns

	X1	X2	X3	X4	X5	X7	X8	Y1	3	4	5
0	0.98	514.5	294.0	110.25	7.0	0.0	0	15.55	0	0	0
1	0.98	514.5	294.0	110.25	7.0	0.0	0	15.55	1	0	0
2	0.98	514.5	294.0	110.25	7.0	0.0	0	15.55	0	1	0
3	0.98	514.5	294.0	110.25	7.0	0.0	0	15.55	0	0	1
4	0.9	563.5	318.5	122.5	7.0	0.0	0	20.84	0	0	0

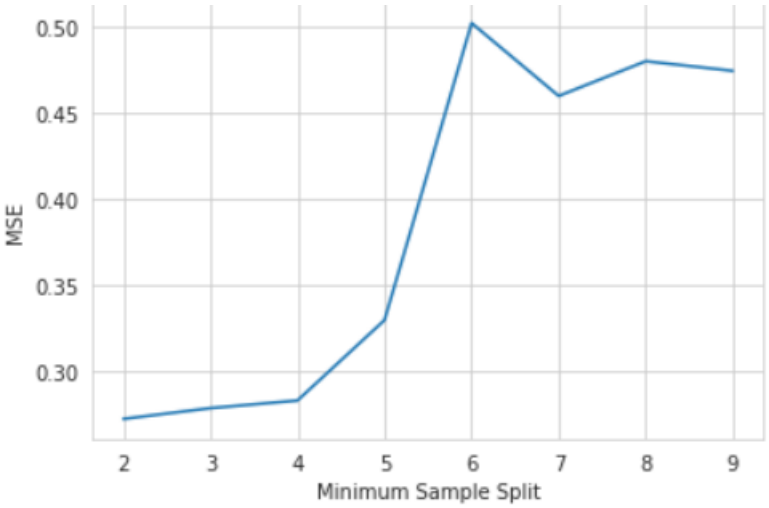
- First dataset was split into train and test set, first in the ratio of 80:20
- Then it was further split into train and validation set, in the ratio of 87.5 : 12.5
- Data preprocessing was completed

Subpart 2:

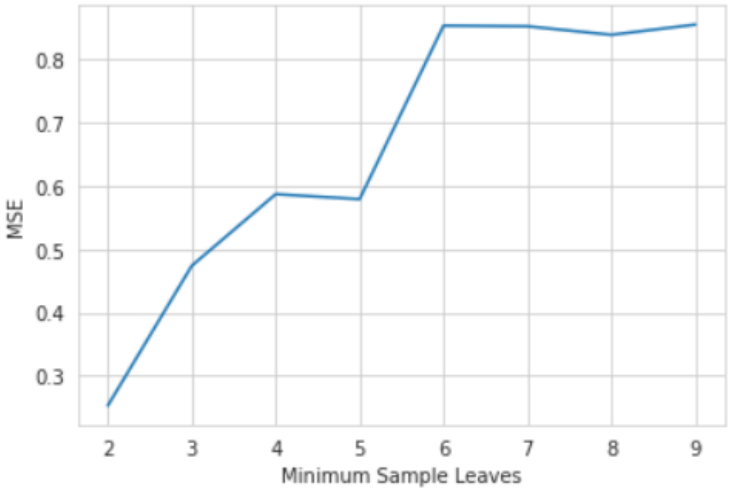
- To get the intuition of how our model behaves on changing hyperparameters, we vary one hyperparameter at a time, training on the training set and calculating MSE on the validation set.
- The results are plotted



Variation of MSE with max_depth



Variation of MSE with min_samples_split



Variation of MSE with min_samples_leaf

- On a rough note MSE increases with increase in min_sample_split and min_samples_leaf and decreases with increase in max_depth
- This is because, as the depth increases our model can made diverse threshold conditions to fit more of the data and define complex decision boundaries
- And as the min_samples_split decreases, our model has more freedom to reduce the entropy of a leaf and increase the information gain by further splitting

- To find the best combination of parameters a function is defined which trains our model over varying hyperparameters and return the dictionary which contains the combination which gives the best MSE

```
def best_parameters(X_train,y_train,X_val,y_val,mse_best):
  for depth in range(2,30):
    for ss in range(2,12):
      for sl in range(1,10):
```

```
{'Depth': 7, 'Sample_Split': 5, 'Sample_Leaf': 2}
```

Subpart 3:

- To perform the different types of validations on the optical hyperparameters obtained first we need to train our model with the optical hyperparamters using the DecisionTreeClassifier()
- After training the model, first performing the Hold-out Cross Validation on the train set and calculating the accuracy

```
Accuracy Score in Hold Out Cross Validation:  0.025974025974025976
```

- After performing the hold-out cross validation now performing the 5 fold cross validation on the train set and calculating the accuracy

```
Accuracy Scores in 5 fold Cross Validation
[0.02777778 0.00925926 0.02803738 0.00934579 0.01869159]
Mean Accuracy Score in 5 fold cross validation is: 0.018622360678435444
```

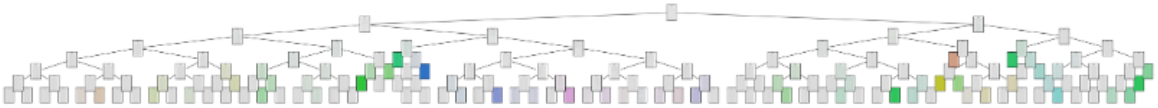
- After performing the 5 fold cross validation now performing the Repeated 5 fold cross validation and calculating the accuracy

```
Mean Accuracy Score in Repeated 5 fold cross validation is: 0.02916475681181564
```

- Finally, we can calculate the mean squared error between the predicted and the ground-truth values in the test data for the model

```
8.045482467532468
```

- Finally, tree was plotted



Question 2

Classification

- Getting the dataset from the read_csv() and doing the head() and info() on the dataset

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

- Dropping the SepalLength and SepalWidth columns from the dataset
- Converting the Species to 0,1 and 2

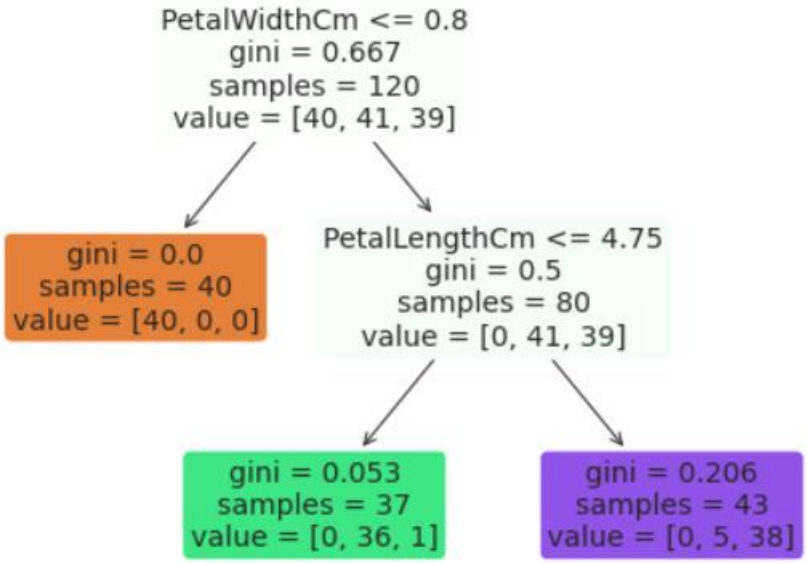
```
dataset2.replace({'Species': {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica':2}})
```

	PetalLengthCm	PetalWidthCm	Species
0	1.4	0.2	0
1	1.4	0.2	0
2	1.3	0.2	0
3	1.5	0.2	0
4	1.4	0.2	0

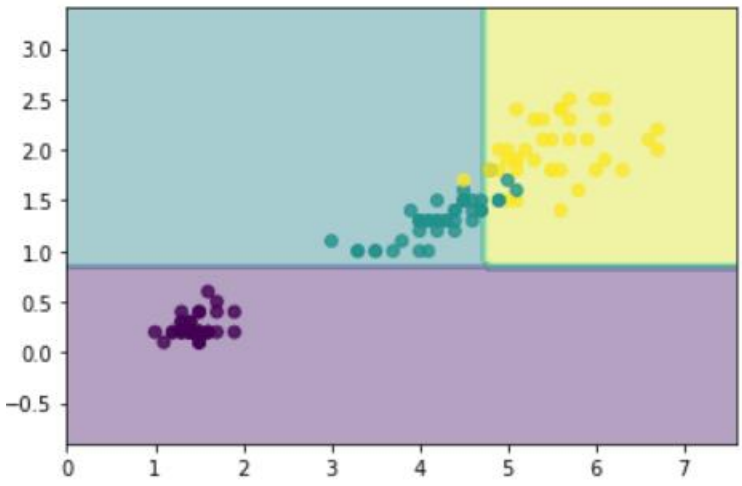
- Splitting the dataset to train and test set in the ratio of 80:20

Subpart 1:

- Now creating a decision tree classifier with max_depth=2 and fitting the training dataset to the model
- After training the model, we are using the trained model to predict the classes for the testing dataset
- Now, plotting the decision tree and indicating the depth at which each split made



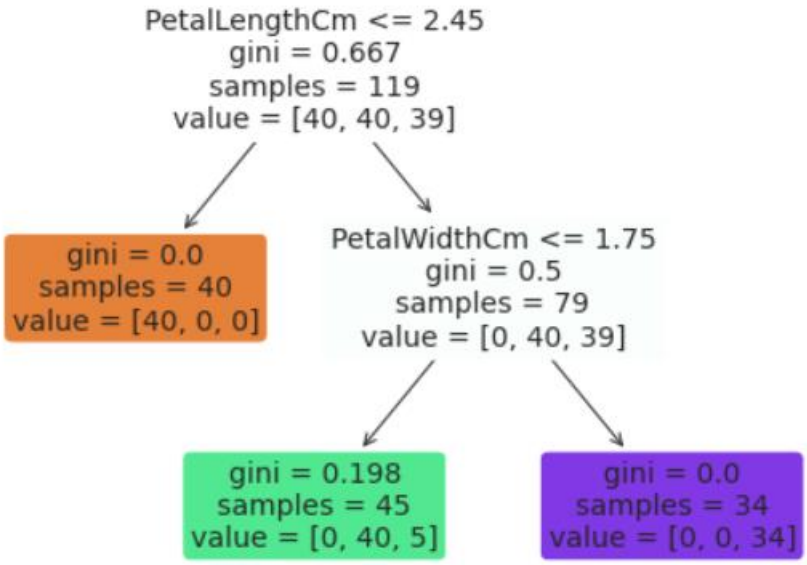
- Now plotting the decision boundary for the tree by creating a grid of points in the feature space and plotting the decision boundary



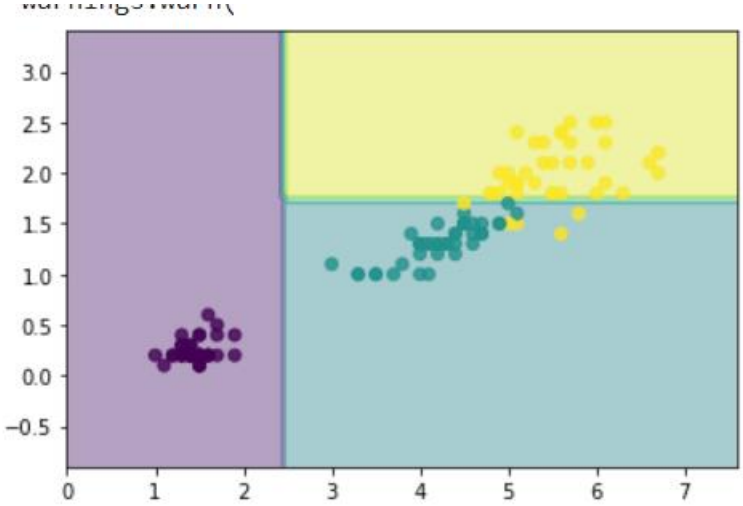
Subpart 2:

- First, identifying the widest Iris-versicolor and removing it from the training and testing sets
- Now, we are creating a decision tree classifier with max_depth=2
- And fitting the training data to the model

- Now we are using the trained model to predict the classes for the testing data
- After predicting now we are plotting the decision tree indicating the depth at which each split was made

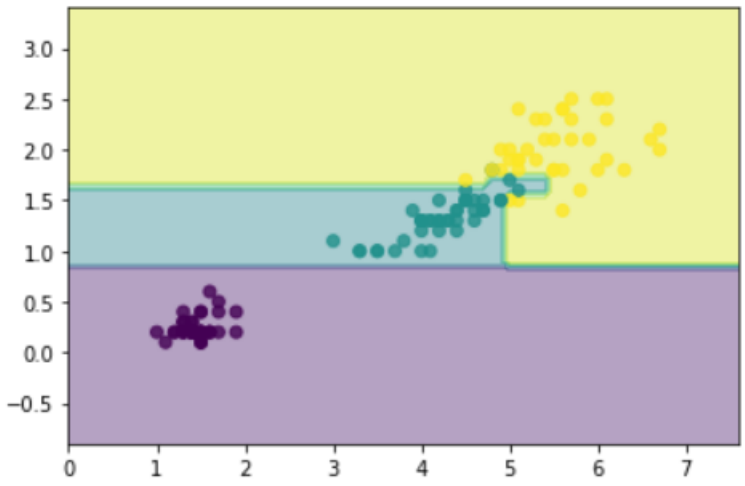


- Now, plotting the decision boundary by creating a grid of points in the feature space and plotting the decision boundary



Subpart 3:

- Now training a decision tree with max_depth=None on the original dataset
- After training we are plotting the decision boundary by ths same method followed for the previous part



- Now comparing the analysis for subpart 1 and subpart 3
- For comparing we are finding accuracy score, precision, recall and f1 score for both the decision tree classifiers one with max_depth=2 and the other with max_depth=None

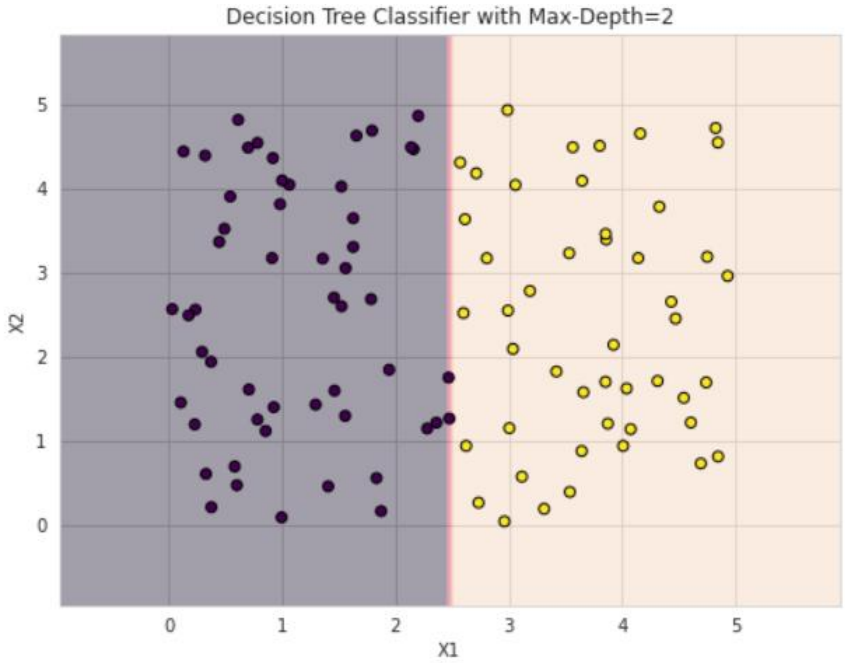
```
Decision tree classifier with max depth=2:
Accuracy:  0.9666666666666667
Precision:  0.9722222222222222
Recall:  0.9629629629629629
F1 score:  0.9658994032395567


Decision tree classifier with max depth=None:
Accuracy:  1.0
Precision:  1.0
Recall:  1.0
F1 score:  1.0
```

- From the above we can say that decision tree classifier with max_depth=None has higher accuracy than the other

Subpart 4:

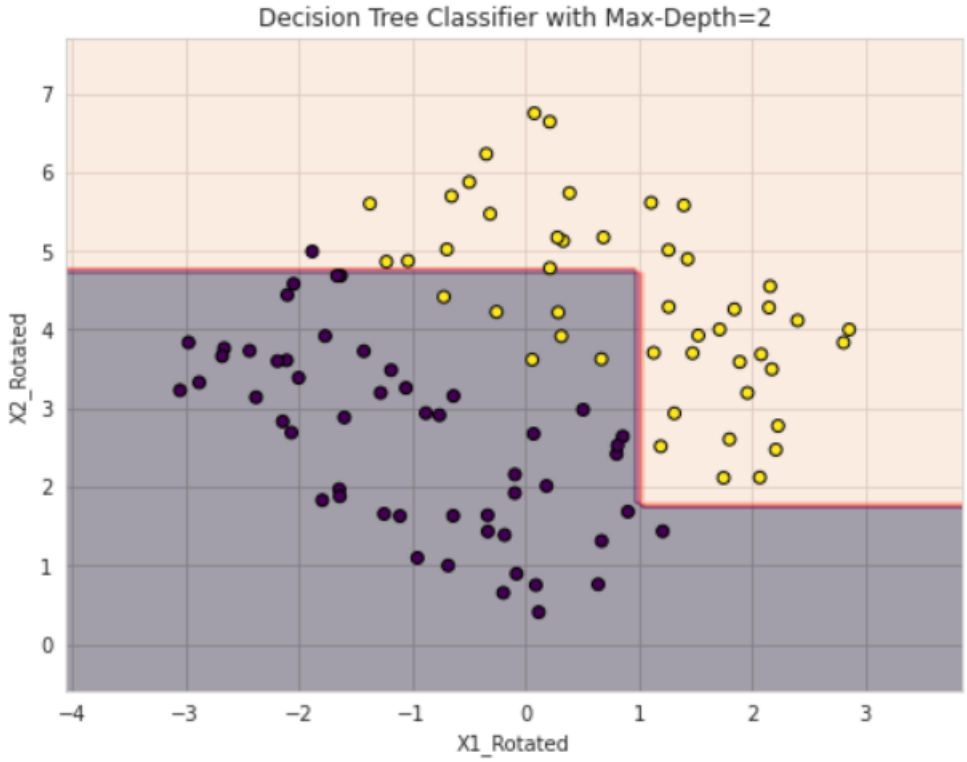
- Creating the random dataset having 2 attributes and 2 classes
- After creating the dataset we are training the decision tree classifier with max_depth=2
- Now plotting the decision boundaries with the data points



- Now rotating the datapoints by 45 degrees clockwise about origin

```
theta = np.radians(-45)
X1_rotated = (X1 * np.cos(theta)) + (X2 * np.sin(theta))
X2_rotated = (X2 * np.cos(theta)) - (X1 * np.sin(theta))
```

- After rotating the datapoints now we are training a decision tree classifier with max_depth=2 on the rotated datapoints
- Plotting the decision boundaries



- Now comparing the plots obtained when the datapoints are normal and plots obtained when datapoints are rotated by 45 degrees
- The decision boundary obtained in the first method (without rotation) will be a straight line passing through (2.5, 0) and separating the two classes.
- The decision boundaries obtained in the second method (with rotation) will be a line separating the two classes but not aat the exact half as the previous one
- Rotating the dataset can change the orientation and distribution of the data points and affects the decision boundary obtained

Subpart 5:

- In part 2 the decision tree classifier has given the max_depth=2 and it is trained on the training set and the decision boundary is plotted.
- In part 3 the decision tree classifier has given the max_depth=None and it is trained on the training set and the decision boundary is plotted
- In part 4 the decision tree classifier has given the max_depth=2 and it is trained on the training set and the decision boundary is plotted
- By doing all these we can observe that the decision tree classifier gives better accuracy for part 4 and then part 3 and finally part 2

Regression

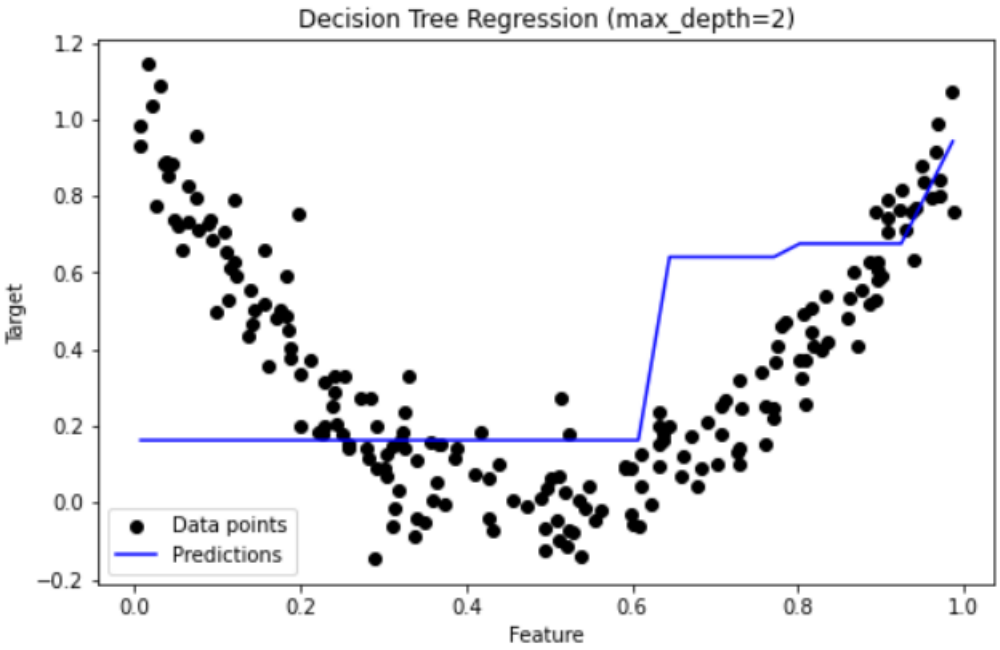
Subpart 1:

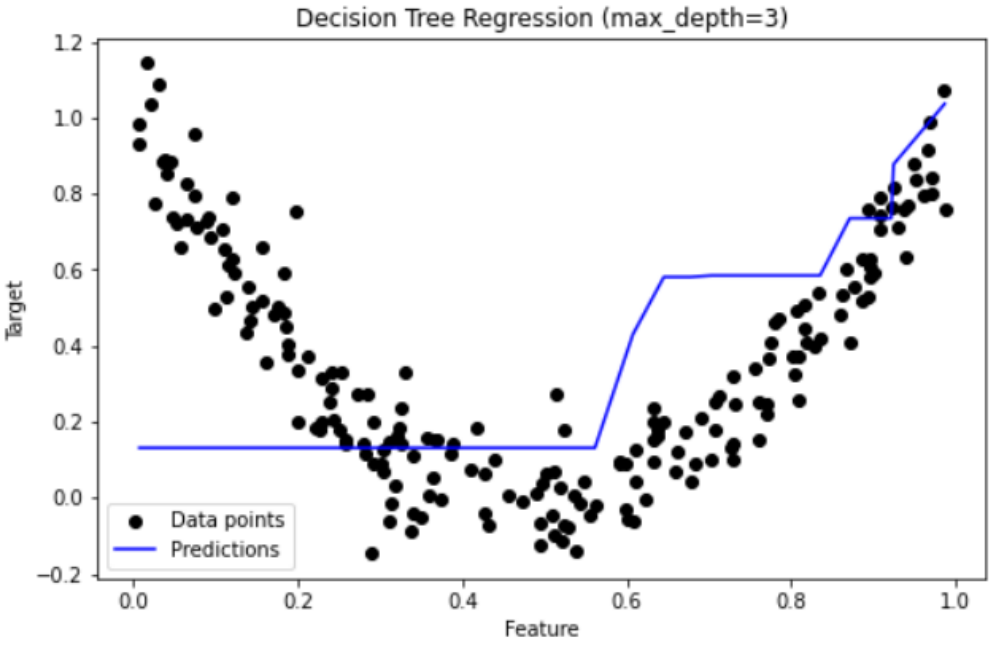
- First getting the dataset by the read_csv()

	X	Y
0	0.374540	-0.005042
1	0.950714	0.835799
2	0.731994	0.244592
3	0.598658	-0.032501
4	0.156019	0.659870

- After reading the data we now split the data into train and test set
- We are now training two decision tree models one with max_depth=2 and the other with max_depth=3 using the DecisionTreeRegressor()

- Now we are plotting the regression predictions for each max_depth using a line plot and scattering the Datapoints on the same plots



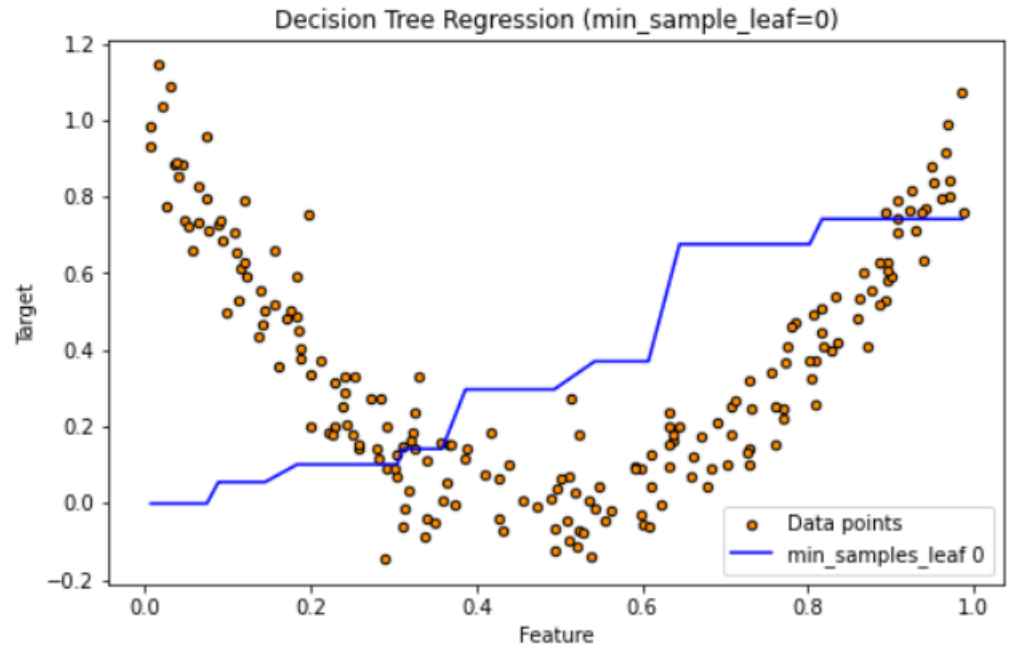


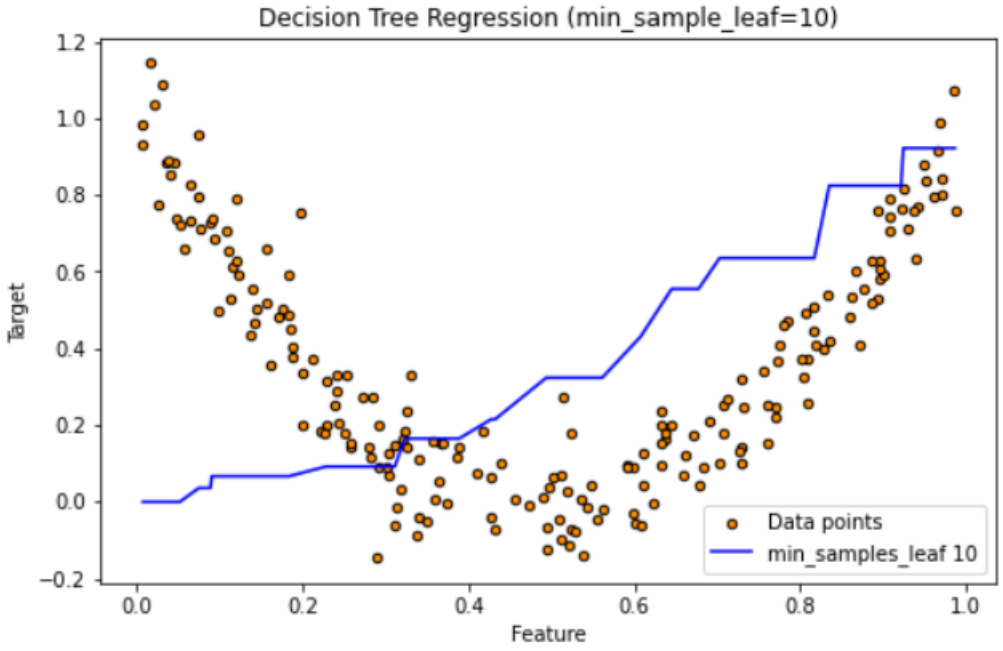
- When max_depth=2 this means that the tree will be limited to making only one split in data. This results in simple model with limited capacity. As a result, the predictions may be biased and have less variance
- When max_depth=3 this means that the tree will be able to make two splits in data. This results in a more complex model compared to

previous one. The predictions may be less biased and have lower variance compared to previous model.

Subpart 2:

- Now we are training our model by varying the min_samples_leaf
- We are taking two cases one has min_samples_leaf=0 and the other has min_samples_leaf=10
- We are looping and plotting the regression predictions for each min_samples_leaf using a line plot and scattering the Datapoints on the same plots





- A smaller min_samples_split allows the decision tree to fit data more closely. A larger min_samples_leaf can prevent overfitting by creating a simpler model with less variance

Question 3

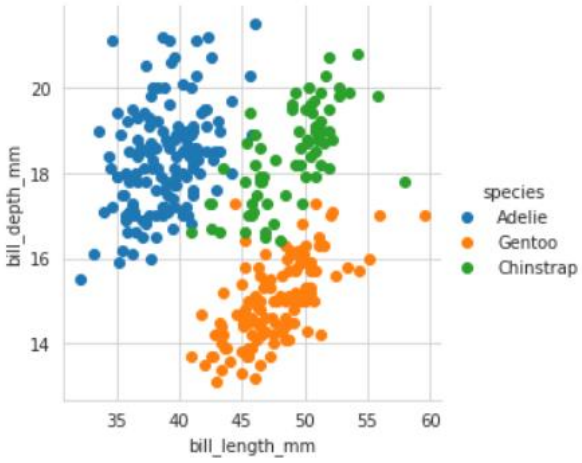
Cost function to be implemented is Gini Index

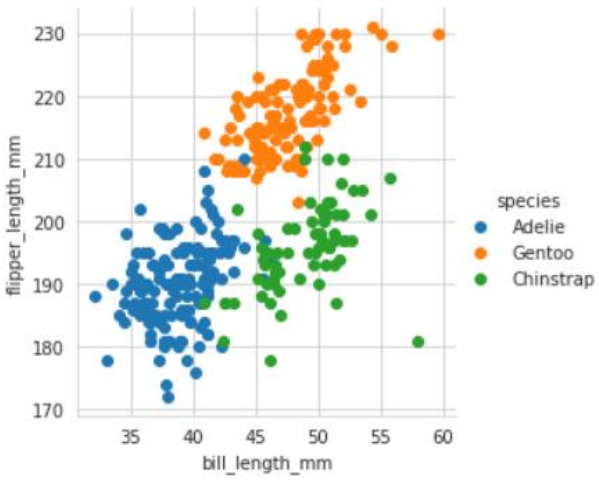
Subpart 1:

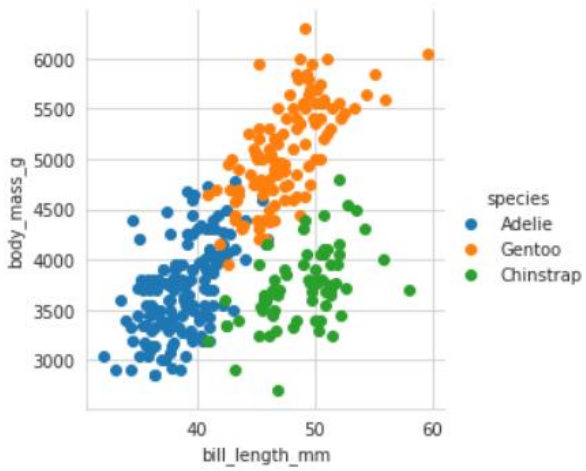
- First installing the dataset and loading it and performing the head()

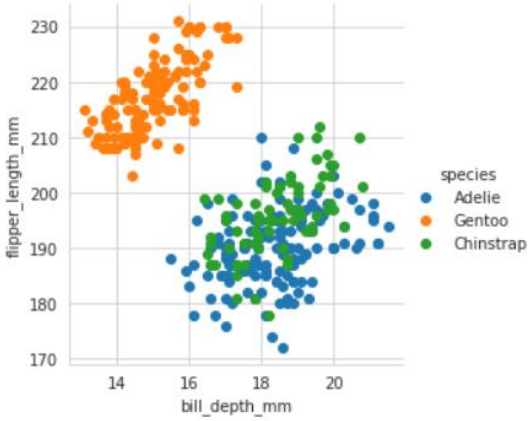
	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	male	2007
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	female	2007
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	female	2007
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN	2007
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	female	2007

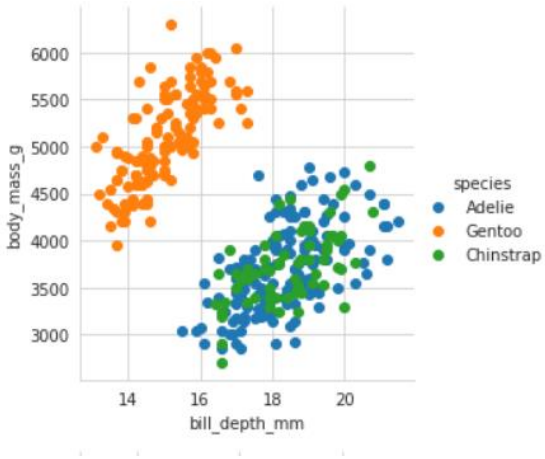
- Some missing values are present in bill_length, bill_depth, flipper_length, body_mass, sex. So dropping the missing values from these columns using dropna()
- Now performing the visualisation of the dataset

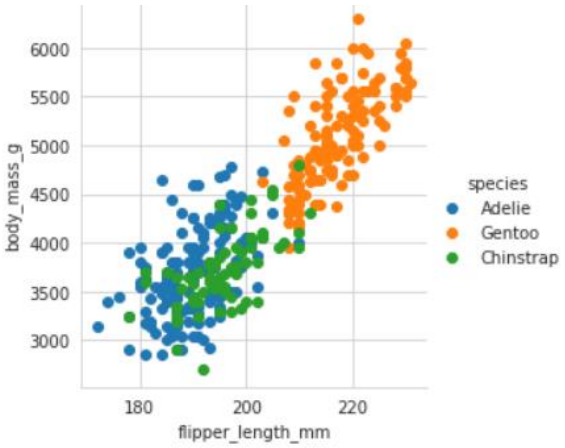






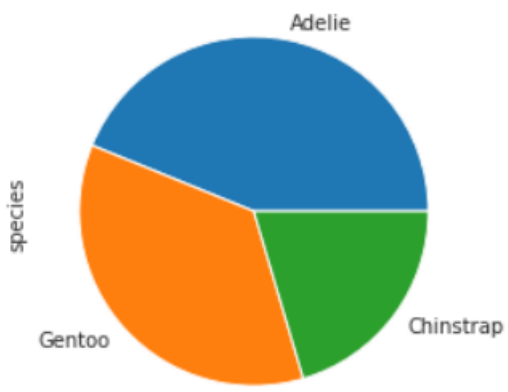




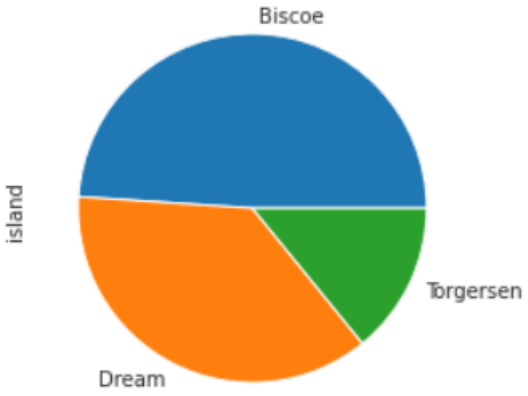


- Now plotting the pie charts

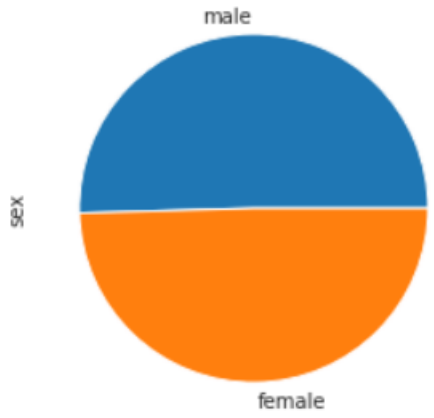
```
Adelie       0.438438
Gentoo       0.357357
Chinstrap    0.204204
Name: species, dtype: float64
```



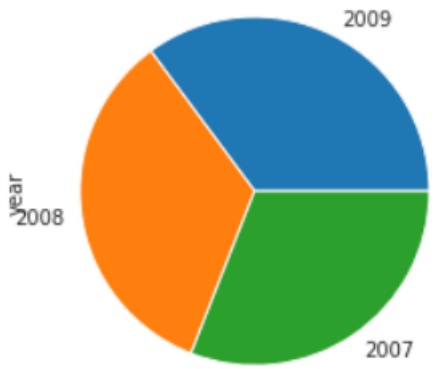

```
Biscoe       0.489489
Dream        0.369369
Torgersen    0.141141
Name: island, dtype: float64
```



```
male      0.504505
female    0.495495
Name: sex, dtype: float64
```




```
2009    0.351351
2008    0.339339
2007    0.309309
Name: year, dtype: float64
```



- Now performing categorical encoding
- Converting the string values of a column into dummy values of 0 and 1 for the columns Sex, Island, Year
- Replacing the species column values with 0, 1 and 2
- Finally, the dataset looks like

	species	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	male	Dream	Torgersen	2008	2009
0	0	39.1	18.7	181.0	3750.0	1	0	1	0	0
1	0	39.5	17.4	186.0	3800.0	0	0	1	0	0
2	0	40.3	18.0	195.0	3250.0	0	0	1	0	0
4	0	36.7	19.3	193.0	3450.0	0	0	1	0	0
5	0	39.3	20.6	190.0	3650.0	1	0	1	0	0

- Now splitting the dataset into train and test sets

Subpart 2:

- Performing the gini index as the cost function

```
def gini_index(data):
  classes = np.unique(data)
  n_samples=len(data)

  if n_samples==0:
    return 0

  gini=1

  for c in classes:
    p=len(data[data==c])/len(data)
    gini-=p**2
  return gini
```

- Gini index = 1 – summation(pi)2

Subpart 3:

- **Cont_to_cat()** function was declared and the pseudocode of the same is
- Split is a dictionary which is used
    - To store the left and right data after the split
    - To store the features which was analysed for the split
    - To store the value of the feature (threshold) used for splitting
    - To store the resulting gini index for the split
- Now we are running a nested loop through all the features and all the values of the features, calculating all possible splits and returning the case with the best split possible

```
def cont_to_cat(data):
  max_gini=-99999999999
  split={}

  for feature in range(data.shape[1]):
    feature_values=data[:,feature]
    unique_values=np.unique(feature_values)
    for threshold in unique_values:
      # left_data=np.array(feature_values<=threshold)
      for row in data:
        if(row[feature]<= threshold):
          left_data=np.array([row])
      for row in data:
        if(row[feature]> threshold):
          right_data=np.array([row])

      if(len(left_data)>0 and len(right_data)>0):
        y, left_y, right_y = data[:,-1], left_data[:,-1], right_data[:,-1]

        # calculate the gini index for the split
        left_gini = gini_index(left_y)
        right_gini = gini_index(right_y)
        split_gini = gini_index(y) - ((left_gini*len(left_y) + right_gini*len(right_y))/len(y))

        if split_gini>max_gini and feature!=12:
          max_gini=split_gini
          split['Gini']=max_gini
          split['Threshold']=threshold
          split['Left']=left_data
          split['Right']=right_data
          split['Feature']=feature

  return split
```

Subparts 4 and 5:

- Cont_to_cat() function itself gets the attribute for the best split and then make the split
- In the code cont_to_cat() function is continuously called recursively and splits are made till
    - Data to be split > min_samples_leaf
    - Depth < max_depth
    - Information gain for the split > 0
- This was called out by the tree() function which gets automatically called on calling fit()
- The tree() implements the cont_to_cat() and gini() functions and makes the decision tree as per give conditions
- If split is not possible, value of the leaf is decided using the voting majority

Subpart 6

- Function was made inside the DTC() class which takes the parameters of X_new and makes predictions based on previous learnings

```
def make_prediction(self, x, tree):
  if tree.value!=None:
    return tree.value

  feature=x[tree.feature]
  if feature<tree.threshold:
    return self.make_prediction(x, tree.left)
  else:
    return self.make_prediction(x, tree.right)
```

Subpart 7

- Finally, our model was evaluated using the test set.
- Model was fitted using hyperparameters: max_depth=7; min_samples_split=2.
- Overall accuracies were calculated