# Pattern Recognition and Machine Learning 2022 Winter Semester

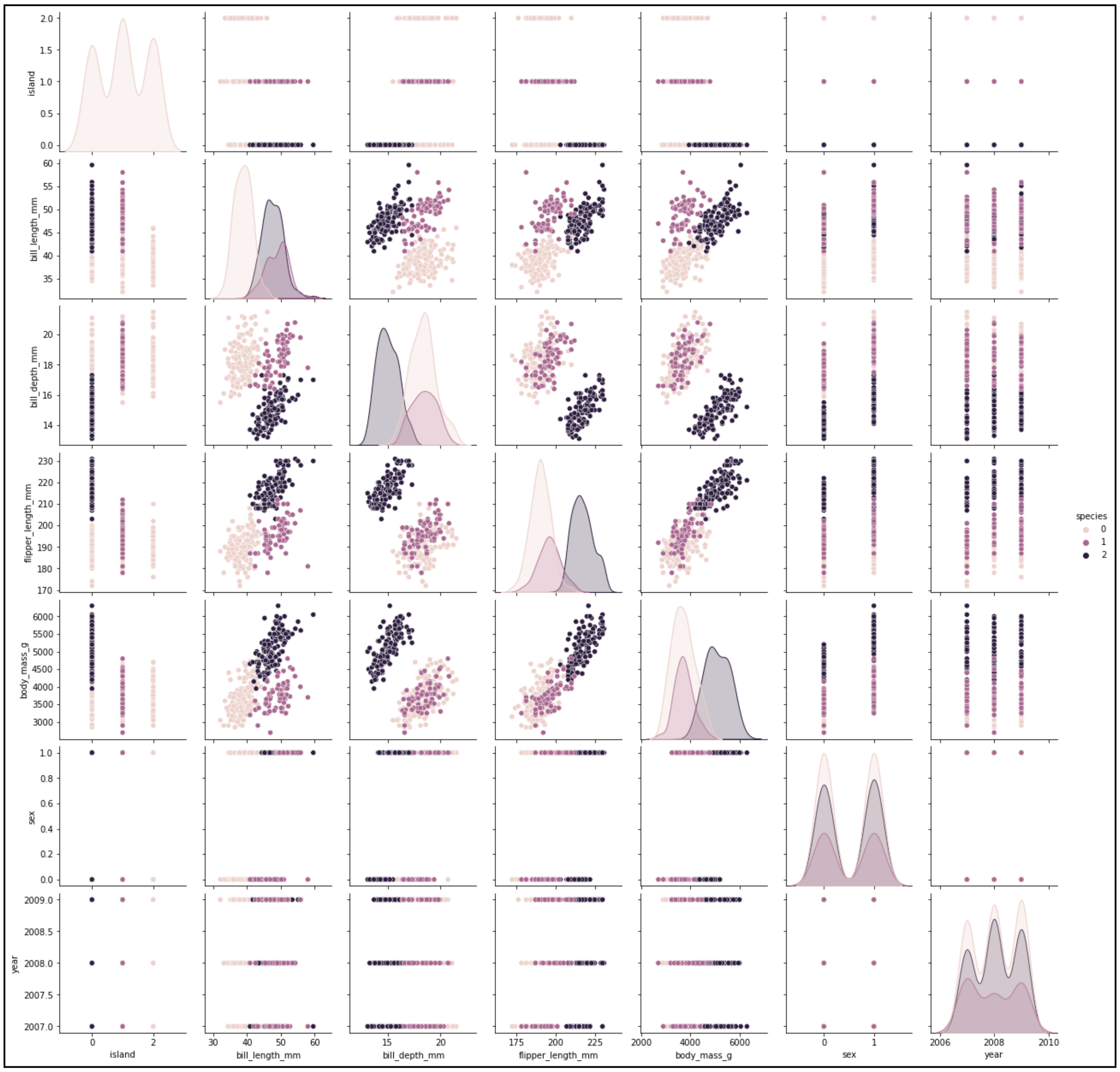
## Report - Lab Assignment - 2

### Question - 1

- Initially, I imported the necessary pandas and numpy libraries for data manipulation and created a Data Frame from the "penguins.csv" dataset given to us.
- Then, I used pd.dropna() function for deleting all the rows that contained NaN values.
- Afterwards, for the encoding part of the string dtype features, I used LabelEncoder provided by sklearn.preprocessing module
- Then for the visualization of the data, I used the seaborn library and created the plots for all possible permutations of any 2 pairs of relevant features.
- Then, using the train_test_split() function of the sklearn.model_selection module, I divided the dataset in a 70:30 ratio.
- Then, I implemented the cost function entropy. I found out the counts of all the unique values of the target class and found out the probabilities and calculated entropy using its formula (Summation: -pi*log(pi)).
- Then, I created the Information Gain function which returned the information gain of the particular feature passed as the argument. The function made use of the entropy function to calculate the total entropy and weighted entropies of all unique values of the feature and return the final answer (IG = Total entropy - weighted entropies)
- Then, for the next part of the question, we had to convert all the continuous format data to categorical format data. For that, I looped from the minimum value of the continuous feature to the maximum value of the feature and found the Information Gain for all the integer data points and the point at which Information Gain was maximum was the Threshold point for making the two-fold split. Thus, the continuous feature was converted into a categorical feature using the threshold given by the information gain.
- Then, I moved on to create the Decision Tree Class and created the helper methods inside it.

- Inside the Decision Tree Class, I implemented the build tree method which builds itself recursively and returns the leaf node at its base case. At each step of the recursive call, the best split function (another method inside Decision Tree Class) is called which returns the attribute with the maximum information at that particular level of the tree.
- If a split value is lesser than the feature values of a particular attribute, then the right branch recursion for the tree occurs and when the split value is greater than or equal to the feature values of a particular attribute, the left branch recursion for the tree occurs.
- Also, when an object is created with the Decision Tree Class, a required argument that is the maximum depth of the tree has to be passed. The tree cannot grow beyond this given maximum depth.
- While the current depth of the tree is less than the maximum depth, splitting keeps on occurring and when the break condition happens, that is current depth = max depth, the recursive call stops and the final unique value of the feature is returned to the previous call.
- In the best split function, the information gain of all the features is compared and the feature with the maximum information gain is selected as the best split attribute for that particular node of the tree.
- When the information gain is completely 0 or completely 1, the node is considered as the leaf node and is returned to the previous call.
- Then, I did the classification of the data and created a predict function inside the Decision Tree class itself which is also based on recursion and returns the predictions based upon the training data provided.
- And finally I created a method named accuracy which calculated the accuracy of the classifier based on the comparison between the predictions returned by our predict function and the class labels provided in the dataset.

# Visualization of the Dataset (Ques -1 )

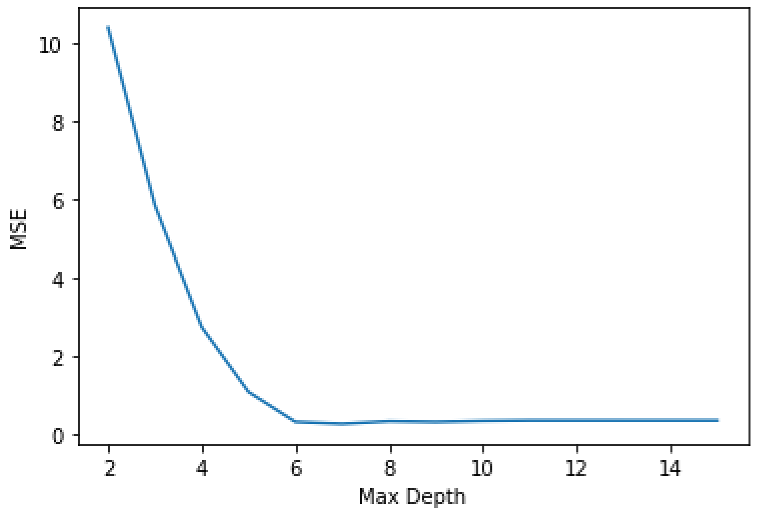


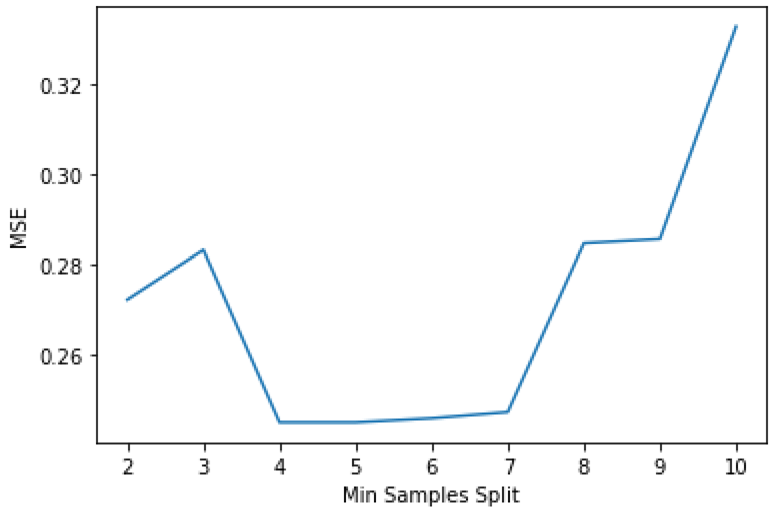
## **Question - 2**

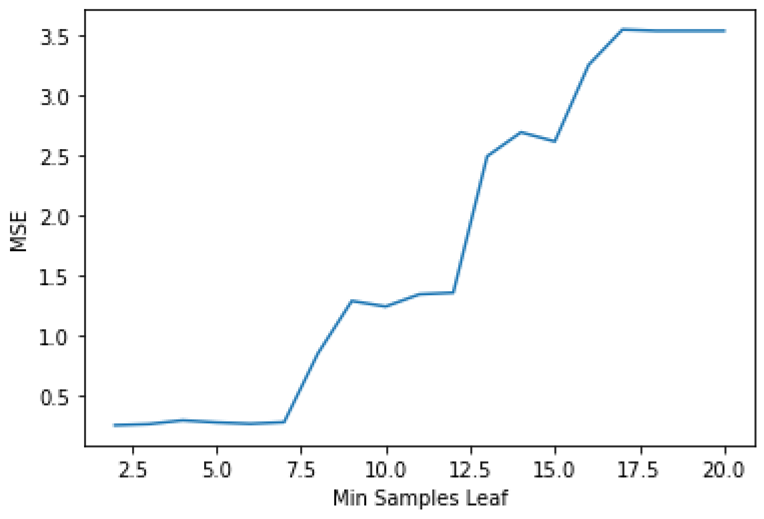
- Initially, I imported all the necessary libraries for data manipulation like pandas and numpy and matplotlib for data visualization purposes.
- Then, I created a DataFrame from the dataset provided.
- Then, for the preprocessing purposes, I dropped all the null values of the dataset using the pd.dropna() function of the pandas library.
- Then, using the train_test_split() function of the sklearn.model_selection module, I divided the data in the ratio 70:10:20 in training, validation and test sets.
- Further, we needed to apply a Decision Tree Regressor model on our dataset of the sklearn.tree module and that Model has many hyper parameters whose values we needed to tune in such a way that generalizes our tree best and has the best performance on the validation set.
- Best performance on the dataset was determined by the different values of the Mean Squared Error that we got by checking different values of a particular hyper parameter.
- I created a function hyper_parameter_tuning(), which took X_train, y_train, X_validation and y_validation data as the arguments and inside the function, I used the following four parameters:
    - ➔ max_depth
    - ➔ min_samples_split
    - ➔ min_samples_leaf
    - ➔ max_leaf_nodes
- Then, I firstly went on to tune the first parameter, maximum depth, I created an array of integers from 2 to 15 as different depths for the model.
- Then, I looped through the array and I applied the DecisionTreeRegressor model at every point and passed that particular depth value to the model as its max_depth parameter and also calculated the mean squared error at every point along with storing those errors in a separate array. The point at which I got the minimum mean squared error was the point of the best maximum depth.
- Then, I plotted the max_depth(x - axis) vs mean_squared_error(y - axis) graph for different values of max_depth and their corresponding mean squared errors.
- I repeated the process 3 more times for the other three hyper parameters as well keeping the previously obtained parameters as the best we got. For example: while calculating the best possible value of min_samples_split, I kept the max_depth parameter as the best_max_depth that I got previously.

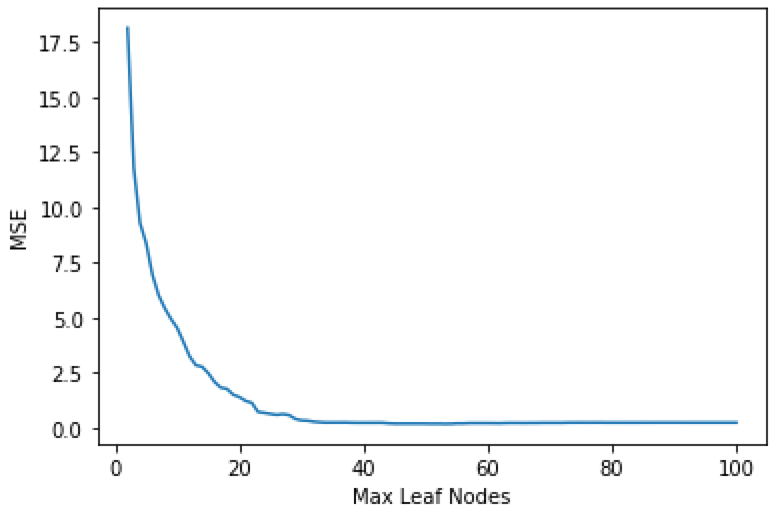
- This way, I got all 4 of the best parameters as:
    - ➔ max_depth - 7
    - ➔ min_samples_split - 4
    - ➔ min_samples_leaf - 2
    - ➔ max_leaf_nodes - 53

- Plots that I got were as:









- Then, from the sklearn.model_selection() module, I imported the cross_val_score function for performing the 5-fold-cross-validation on the model applied using the hyper parameters that I got previously.
- The cross validation scores were like:
```
[0.99727899 0.99620728 0.99708826 0.99748273 0.99254399]
```
- Then, using the sklearn.metrics module, I calculated the mean_squared_error on the testing data and the value as `0.30278892912643374`

- After that, I plotted the Decision Tree created and it looked something like: