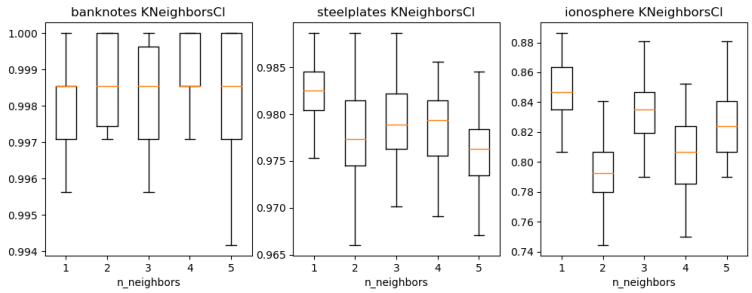
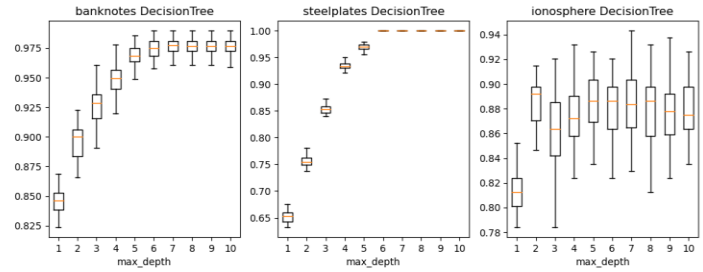
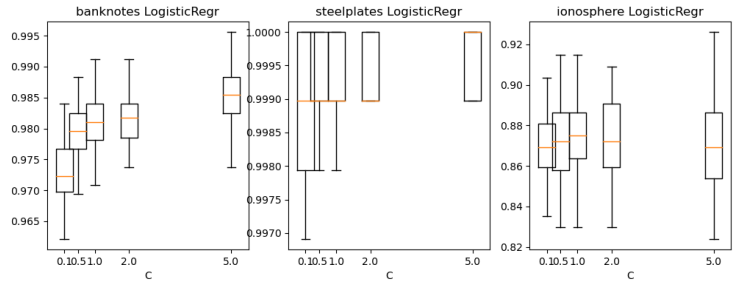
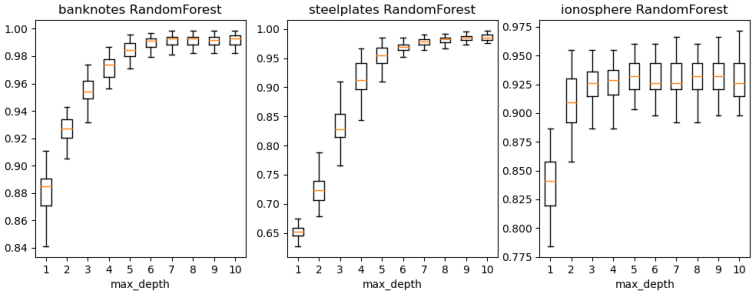
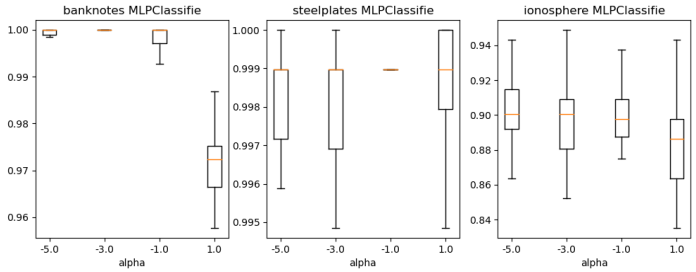
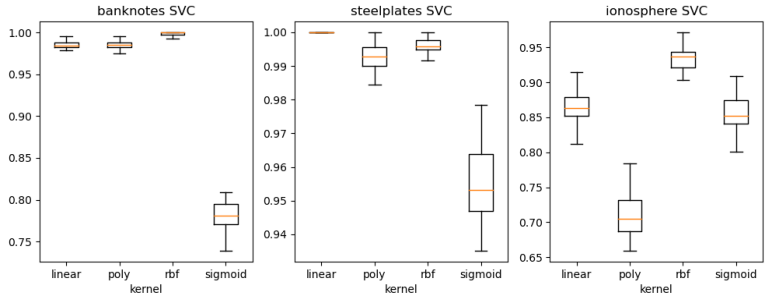
Assignment 1

Part 1:

(i)

	banknotes	steelplates	ionosphere
KNN			
Decision Tree			
Logistic Regression			
Random Forest			

MLP Classifier	
SVC	

(ii) Table containing the lowest mean test error

Classifier	banknotes	Steelplates	ionosphere
KNN	0.00	0.02	0.15
Decision Tree	0.02	0.00	0.12
Logistic Regression	0.01	0.00	0.13
Random Forest	0.01	0.01	0.07
MLP Classifier	0.00	0.00	0.10
SVC	0.00	0.00	0.07

Table containing the corresponding hyperparameter values

Classifier	banknotes	Steelplates	ionosphere
KNN	2	1	1
Decision Tree	7	6, 7, 8, 9, 10	2,5
Logistic Regression	5.0	5.0	2.0
Random Forest	7	10	5,8
MLP Classifier	-3.0	-1, 1.0	-5.0
SVC	rbf	linear	rbf

Hyperparameters values justification:

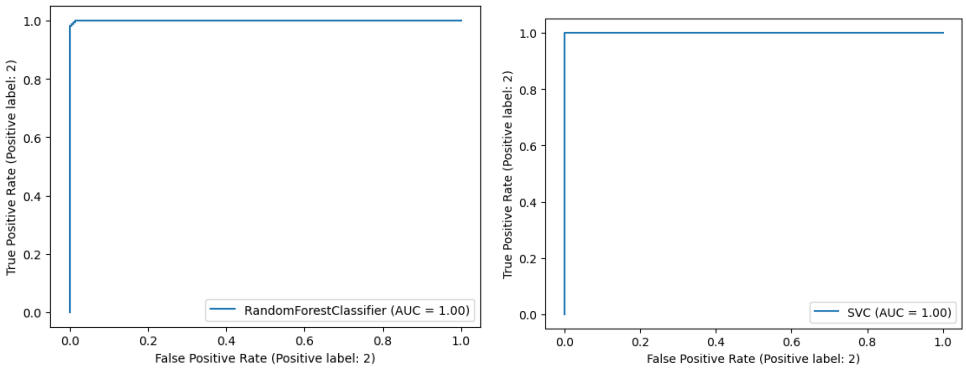
```
# Draw the box plots to visually display the model performance.
if controlName in ['alpha']: controlOptions = np.log10(controlOptions) # for semilogx
elif controlName in ['kernel']: controlOptions = [1, 2, 3, 4]
a = plt.boxplot(scores, positions=controlOptions, showfliers=False, showmeans=True)
```

- I used this “showmeans=true” to make sure where the means exactly are and showed a green arrow of where mean is. This is in the code provided and here is an example on the right on how it looks. Hence how I knew KNN banknotes parameter value was 2. And Logistic Regression ionosphere value was 2.0 as well.

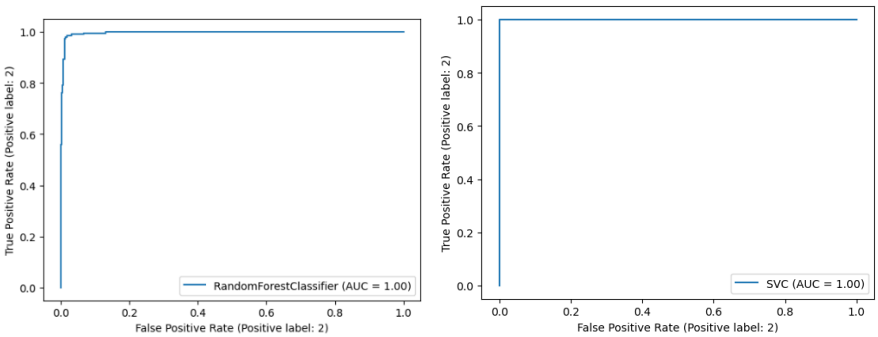
- For the Decision Tree Steelplates data, max_depth of 6,7,8,9,10 are all very close to 1 and can be considered as the values for the lowest mean test error. However, if one value had to be chosen, it would be 6 because it can help prevent overfitting, and lower dept is usually less computationally expensive to build as its simple.
- For Random Forest banknotes data, max_dept 6,7,8,9,10 are also close although mean of 7 value was slightly higher hence why I chose that, but 6,7,8,9,10 are all valid answers for corresponding values as they are all relatively high.
- For Random Forest banknotes data, max_dept 5 and 8 are very similar and hard to distinguish which one is causing the lowest mean test error. But it would be to choose 5 then 8 as the max_dept is less and hence simpler.
- For the MLP steelplates data, the corresponding value could be either -1 or 1. But -1 is less spread so that could be better value to choose.

(iii) ROC curves; SVC and the Random Forest Classifier on the three classification datasets.

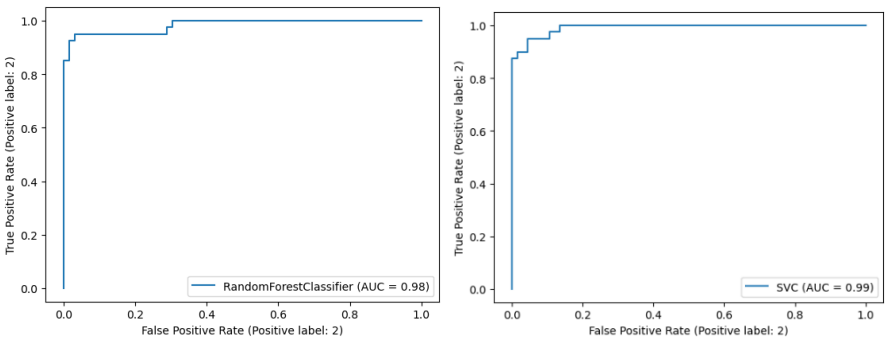
Banknotes:



Steelplates:



Ionosphere:



(iv) Report:

- For the banknote's dataset, KNN, MLP and SVC achieved the lowest mean test error of 0.00 indicating excellent performance. The hyperparameter values were 2 nearest neighbours for KNN, -3.0 alpha for MLP and rbf for SVC.
- For steelplates dataset, decision tree, logistic regression, MLP, SVC achieved the lowest mean error of 0.00. The hyperparameter values were -1.0 and 1.0 alpha for MLP and linear for SVC.
- For ionosphere, there is a variety of different lowest mean test errors. Random forest and SVC achieved the lowest mean test error of 0.07. The hyperparameter values were 5 and 8 max-depth for random forest and rbf for SVC.
- Overall, the best classifier in this case is the SVC because it achieved lowest mean test error of 0.00 for the banknotes and steelplates dataset and for ionosphere it had error of 0.07, which is lowest overall and best performance. MLP would be the second-best.
- The errors could be due to the complexity of the data, noise, and outliers. This means the ionosphere data could be more complex than the others.

- KNN: The choice of the number of neighbours(k) in KNN can significantly impact the model's performance. Small value of k can lead to overfitting and large value of k can result in underfitting. Optimal value of k may vary depending on dataset.
- Decision Tree are sensitive to hyperparameters such as maximum depth and minimum samples split. Higher maximum depth values can lead to more complex trees and increased risk of overfitting, while too low values may result in underfitting. Similarly, the minimum samples split parameter affects the granularity of the tree's decision-making process.
- Logistic Regression is relatively less sensitive to hyperparameters compared to other models. The regularization parameter (C) controls the trade-off between model complexity and overfitting. However, the impact of C on the model's performance is generally moderate, and higher values of C may lead to overfitting.
- Random Forest is sensitive to hyperparameters such as the number of trees in the forest and the maximum depth of trees. Increasing the number of trees may improve the model's performance up to a certain point, beyond which it may lead to diminishing returns or increased computational cost. Similarly, setting an appropriate maximum depth can help balance model complexity and generalization ability.
- MLP Classifier is highly sensitive to hyperparameters like the number of hidden layers, the number of neurons in each layer, and the learning rate. The choice of these parameters significantly impacts the model's convergence, generalization, and ability to capture complex patterns in the data.
- SVC is sensitive to the choice of kernel function (e.g., linear, RBF) and its parameters such as C and gamma. The selection of appropriate kernel parameters is crucial for controlling the trade-off between model complexity and error minimization. Additionally, the choice of kernel function affects the model's ability to capture nonlinear relationships in the data.

- For banknotes data, both curves achieve an AUC of 1.00 but the ROC curve for SVC exhibits an ideal shape ascending steeply and sharply from bottom left corner towards the top right corner. The ROC for Random Forest Classifier has a slight deviation from ideality, as indicated by a curve towards the right at the point where TPR reaches 1.0.

Despite this, it’s only a slight curve and AUC is still 1.00 hence overall performance is still good. The ROC curve for SVC positioned further to the left throughout the y-axis. This also suggests that the classifier achieves better performance in terms of minimizing the false positive rate while maximizing the true positive rate. Furthermore, SVC has the ideal point where the true positive rate is 1.0 and the false positive rate is 0.0. Overall, they are both good classifiers.

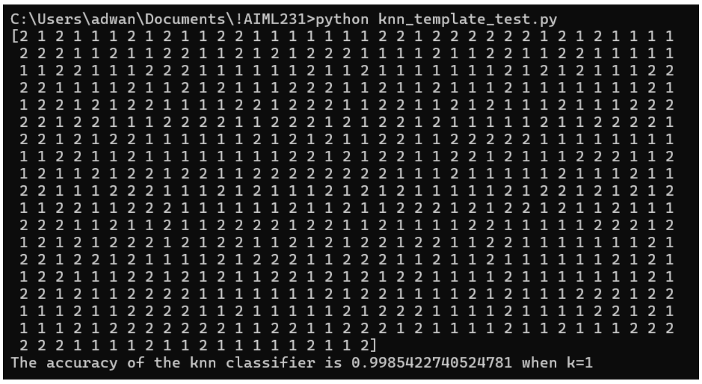
- For steelplates data, both curves achieve an AUC of 1.00, just like banknotes data, the ROC curve for SVC has an ideal shape ascending sharply from bottom left corner towards the top right corner. This classifier for this dataset minimizes the false positive whiles maximizing the true positive rate just like SVC for banknotes. The ROC for Random Forest Classifier has a slightly greater deviation from ideality. When Random Forest Classifier’s TPR increases to 1, the FPR is slightly shifting to the right (y-axis increases). Whereas SVC’s ROC Curve it’s on the y-axis of zer0. This shift to the right is very minimal that’s why the overall performance is still very good. This is also shown by AUC still being equal to 1.00 for both Curves.

- For ionosphere data. The ROC curve for SVC achieves an AUC of 0.99 whereas the ROC curve for Random Forest Classifier achieves an AUC of 0.98. This means that SVC is the better classifier as there is 0.01 more area under curve. Both deliver good performance as it’s only a very minimal difference. The random forest classifier reaches TPR of 1.0 when FPR is shifted to the right until it reaches 0.4 (y-axis). Whereas SVC reaches TPR of 1.0 when FPR is 0.2. This means SVC ROC curve is more to the left which is more ideal for maximizing TPR and minimizing FPR. The shape of the ROC curve are different to the other datasets.

- From this we can see that the best ROC curves come from banknote dataset.

Part 2:

Code can be seen in knn_template.py. Running code from Command Prompt:

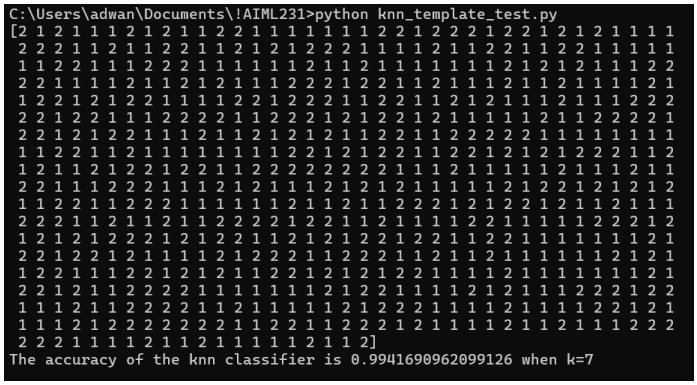


```
C:\Users\adwan\Documents\!AIML231>python knn_template_test.py
[2 1 2 1 1 1 2 1 2 1 1 2 2 1 1 1 1 1 1 1 2 2 1 2 2 2 2 2 2 1 2 1 2 1 1 1 1
 2 2 2 1 1 2 1 1 2 2 1 1 2 1 2 1 2 2 2 1 1 1 1 2 1 1 2 2 1 1 2 2 1 1 1 1 1
 1 1 2 2 1 1 1 2 2 2 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 2 1 2 1 2 1 1 1 2 2
 2 2 1 1 1 1 2 1 1 2 1 2 1 1 1 2 2 2 1 1 2 1 1 2 1 1 1 2 1 1 1 1 1 1 1 2 1
 1 2 2 1 2 1 2 2 1 1 1 1 2 2 1 2 2 2 1 1 2 2 1 1 2 1 2 1 1 1 2 1 1 1 2 2 2
 2 2 1 2 2 1 1 1 2 2 2 2 1 1 2 2 2 1 2 1 2 2 1 2 1 1 1 1 1 2 1 1 2 2 2 2 1
 2 2 1 2 1 2 2 1 1 1 1 1 1 2 1 2 1 2 1 1 2 2 1 1 2 2 2 2 2 1 1 1 1 1 1 1 1
 1 1 2 2 1 1 2 1 1 1 1 1 1 1 1 2 2 1 2 1 2 2 1 1 2 2 1 2 1 1 1 2 2 2 1 1 2
 1 2 1 1 2 1 2 2 1 2 2 1 1 2 2 2 2 2 2 2 2 1 1 1 2 1 1 1 1 1 2 1 1 1 2 1 1
 2 2 1 1 1 2 2 2 1 1 1 2 1 2 1 1 1 2 1 2 1 2 1 1 1 1 1 2 1 2 2 1 2 1 2 1 2
 1 1 2 2 1 1 2 2 2 1 1 1 1 1 1 2 1 1 2 1 1 2 2 2 1 2 1 2 2 1 2 1 1 2 1 1 1
 2 2 2 1 1 2 1 1 2 1 1 2 2 2 2 1 2 2 1 1 2 1 1 1 1 2 2 1 1 1 1 1 2 2 2 1 2
 1 2 1 2 1 2 2 2 1 2 1 2 2 1 1 2 1 2 1 2 2 1 2 2 1 1 2 2 1 1 1 1 1 1 1 2 1
 2 2 1 2 2 1 2 1 1 1 1 2 1 2 2 2 1 1 2 2 2 1 1 2 1 1 2 1 1 1 1 1 1 2 1 2 1
 1 2 2 1 1 1 2 2 1 2 1 2 1 1 2 2 1 2 1 2 1 2 2 1 1 1 1 2 1 1 1 1 1 1 1 2 1
 2 2 1 2 1 1 2 2 2 2 1 1 1 1 1 1 1 2 1 2 2 1 1 1 1 2 1 2 1 1 1 2 2 2 1 2 2
 1 1 1 2 1 1 2 2 2 2 1 1 2 1 1 1 1 1 2 1 2 2 2 1 1 1 1 2 1 1 1 1 2 2 1 2 1
 1 1 1 2 1 2 2 2 2 2 2 2 1 1 2 2 1 1 2 2 2 1 2 1 1 1 1 2 1 1 2 1 1 1 2 2 2
 2 2 2 1 1 1 1 2 1 1 2 1 1 1 1 1 2 1 1 2]
The accuracy of the knn classifier is 0.9985422740524781 when k=1
```

The accuracy of the KNN classifier is 0.9985422740524781 when k=1 for the test set.

The accuracy of the KNN classifier is 1.0 when k=3 for the test set.

The accuracy of the KNN classifier is 1.0 when k=5 for the test set.



The accuracy of the KNN classifier is 0.9941690962099126 when k=7 for the test set.

Each number represents the predicted class label (1 or 2) assigned by the KNN classifier to a particular instance in the test set. When k = 3 or k =5 the accuracy is 1.0 which means perfect accuracy. When k =1, accuracy is 0.9985 (4dp) which is still highly accurate but is less than 1.0. This decrease in accuracy could be due to overfitting. With k=1, the classifier relies heavily on the nearest neighbour, potentially capturing noise or outliers in the training data, leading to reduced generalization ability.

When k =7, accuracy is 0.9942(4dp) which is still highly accurate but is less than 1.0. This decrease in accuracy could be due to underfitting. With a larger value of k, the classifier considers more neighbours, resulting in a smoother decision boundary that may not capture the intricacies of the data well enough.

k = 3 and k = 5 both have perfect accuracy but if one has to be chosen, that would be k=3 because of high accuracy and potentially it could reduce computational cost as less nearest

neighbours may be involved.
Overall, all these different give high accuracy as they all over 0.99 hence differences across these k nearest neighbours is very minimal.

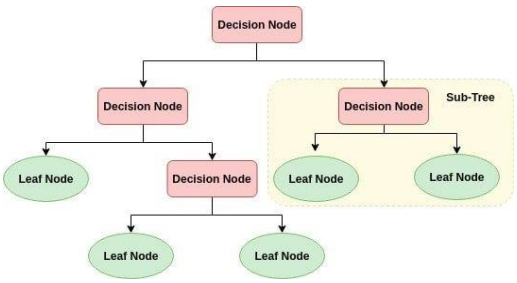
Part 3:

```
C:\Users\adwan\Documents\!AIML231>python decision_tree_template.py
[1, 0]
Humidity=h&depth=0
->left    Outlook=o&depth=1
->left      leaf:None=None&label=1&depth=2
->right     Wind=s&depth=2
->left        leaf:None=None&label=0&depth=3
->right       leaf:None=None&label=1&depth=3
->right   leaf:None=None&label=1&depth=1
```

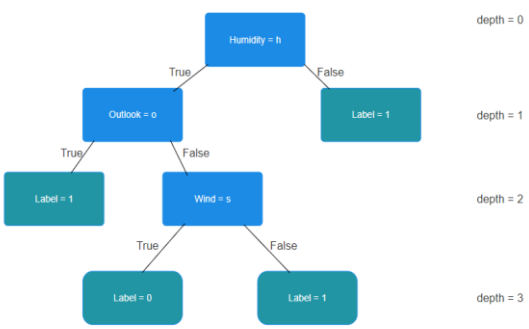
The decision tree learning algorithm starts by considering the entire dataset at the root node. It then selects the best feature to split the data, aiming to maximize information gain or minimize impurity. This splitting process is recursive, continuing until a stopping criterion is met, such as reaching a maximum tree depth or having minimum samples per leaf node. At each node, the dataset is partitioned based on the chosen feature's values, forming child nodes. Once a stopping criterion is reached, the node becomes a leaf node, assigned with a class label, often the majority class of its samples. Optionally, pruning may be applied to remove unnecessary branches, improving the model's generalization. For classification, the decision tree traverses from the root to the leaf nodes, applying splitting conditions until reaching a leaf, where the class label is assigned as the prediction for new instances.

```
function DecisionTreeLearning(data):
    if stopping_criterion(data):  # Check if stopping criterion is met
        return create_leaf_node(data)
    else:
        best_split_feature = select_best_split_feature(data)
        node = create_node(best_split_feature)
        for value in unique_values(best_split_feature):
            partitioned_data = partition_data(data, best_split_feature, value)
            if is_empty(partitioned_data):
                child_node = create_leaf_node(data)
            else:
                child_node = DecisionTreeLearning(partitioned_data)
            node.add_child(value, child_node)
        return node
```

Here is also pseudo-code which supports the explanation above. For example, if stopping criteria is met, we make the current node a leaf. If not then, it selects the best feature to split the data and creates a decision node for it. For each partitioned subset of the data, it calls the DecisionTreeLearning function to construct the subtree.



Here is a diagram from which we can see visually. The decision tree will have conditions for going into the left node or right node. These nodes can be a decision node which further splits or a leaf node/



Predicted class labels for the two data instances were [1,0]. For the first instance, there are 3 possibilities for the class label being assigned class 1:
Humidity = h (true) so follows the left branch, then Outlook = o(true), then left leaf node labelled as class 1.
Humidity=h(true, so follows the left branch, then Outlook does not equal o (false), follows the right branch, then wind does not equal s (false), so continues

to the right leaf node labelled as class 1.
Lastly, Humidity does not equal h (false), and it continues to the right leaf node labelled class 1.
For the second instance, there is only one way to be assigned the class label 0 which is humidity=h(true), then Outlook does not equal o (false) , then wind=s(true), so it continues to the left node labelled as class 0.

The root node of your decision tree is based on the condition "Humidity = h" (high). Instances where the "Humidity" attribute equals "h" go to the left branch, while instances with different humidity values go to the right branch. Humidity feature has 14 instances, 7 of them are h (high) humidity and the other 7 are n (normal) humidity shown in the code. Hence 7 high humidity instances go the left node and 7 normal humidity for the right node.

Entropy (E) of Play Tennis (class label) is: / / $E = -(p \cdot \log_2(p) + q \cdot \log_2(q))$ formula used

$E = -(3/14 \log_2(3/14) + 11/14 \log_2(11/14))$ = 0.74959525725 = 0.75

//shown in decision_tree_template.py code
h = 1,0,1,1,0,1,0 // high humidity has 3 'class 0' labels and 4 'class 1' label
n = 1,1,1,1,1,1,1 // normal humidity has 7 'class 1' label

$-(4/7 \times \log_2(4/7) + 3/7 \times \log_2(3/7))$ = 0.98525714285 // entropy of high humidity partition
entropy for n (normal) humidity partition = 0 // because we are very certain (all 1 class label)

7/14 x 0.98525714285 = 0.492628571 //7/14 instances were h (high)
7/14 x 0.0 = 0.0 //7/14 instances were n (normal)

Conditional entropy = 0.492628571 = 0.49 // Conditional entropy involves humidity feature.

Information gain (IG) = Entropy of Play Tennis (class label) - Conditional Entropy
IG = 0.74959525725 - 0.492628571 = 0.256966686 = 0.257
This feature (humidity) has the highest information gained out all the other features hence why is chosen as the root node.