Task 1 - Decision Tree Classifier

Create the Decision Tree classifer and visualize it graphically

In [2]:
```
# Importing the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [3]:
```
# Importing and analyzing the dataset
iris = pd.read_csv(r'C:\Users\maniz\Downloads\Iris.csv')
iris.info()
print(iris.head())
```

```
<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
   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
```

In [4]:
```
iris.shape
```

Out[4]:
```
(150, 6)
```

In [5]:
```
iris.describe()
```

Out[5]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [6]:
```
iris.isnull().sum()
```

Out[6]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

In [7]:
```
iris['Species'].value_counts()
```

Out[7]:
```
Iris-virginica     50
Iris-versicolor    50
Iris-setosa        50
Name: Species, dtype: int64
```

In [8]:
```
species = {
    'Iris-setosa': 0,
    'Iris-versicolor':1,
    'Iris-virginica':2
}
```

In [9]:
```
iris['Species'] = iris['Species'].map(species)
```

In [10]:
```
iris['Species'].unique()
```

Out[10]:
```
array([0, 1, 2], dtype=int64)
```

To build any model using sklearn we have to specifically define the target variable and all the independent variables.

In [11]:
```
# Separating dependent and independent variables
y = iris['Species']
X = iris.drop(['Id','Species'], axis=1)
```

In [12]:
```
X.shape, y.shape
```

Out[12]:
```
((150, 4), (150,))
```

In [13]:
```
# Now to validate how our model will perform on unseen data, we need to create a validation set
# and to create this we will use train test split vaidation set of sklearn modelselection module

from sklearn.model_selection import train_test_split
```

In [14]:
```
# Creating the train and validation set

X_train, X_valid, y_train, y_valid = train_test_split(X, y, random_state = 101, stratify=y, test_size=0.25)

#stratify=y will make similar distribution of classes in both training and validation set and test_size=0.25 means
# that X_train will contain 75% of the data and test data will contain only 25% of the data.
```

In [15]:
```
# Distribution in training set
y_train.value_counts(normalize=True) #value_counts will return the counts of all classes.
# and normalize=True will return the %
```

Out[15]:
```
2    0.339286
1    0.330357
0    0.330357
Name: Species, dtype: float64
```

In [16]:
```
# Similarly check in validation set
y_valid.value_counts(normalize = True)
```

Out[16]:
```
1    0.342105
0    0.342105
2    0.315789
Name: Species, dtype: float64
```

In [17]:
```
# Its almost same in training and validation set
```

In [18]:
```
X_valid.shape, y_valid.shape
```

Out[18]:
```
((38, 4), (38,))
```

In [19]:
```
#importing decision tree classifier
from sklearn.tree import DecisionTreeClassifier
```

In [20]:
```
# creating the decision tree function
dt_model = DecisionTreeClassifier(random_state=10)
```

In [21]:
```
# fitting the model
dt_model.fit(X_train, y_train)
```

Out[21]:
```
DecisionTreeClassifier(random_state=10)
```

In [22]:
```
#Checking the training score
dt_model.score(X_train, y_train) # using gini criterion
```

Out[22]:
```
1.0
```

Our model is 100% accurate on the training set, wow great ! but wait check accuracy on validation set.

In [23]:
```
# checking validation score
dt_model.score(X_valid, y_valid)
```

Out[23]:
```
0.9210526315789473
```

Accuracy drops to 92% means accuracies of training and validation set are not in sink.

Lets try another criterion i.e. - 'Entropy'

In [24]:
```
dt_model_entropy = DecisionTreeClassifier(criterion = 'entropy', random_state = 10)
```

In [25]:
```
dt_model_entropy.fit(X_train, y_train)
```

Out[25]:
```
DecisionTreeClassifier(criterion='entropy', random_state=10)
```

In [26]:
```
# checking the training score
dt_model_entropy.score(X_train, y_train)
```

Out[26]:
```
1.0
```

In [27]:
```
# checking validation score
dt_model_entropy.score(X_valid, y_valid)
```

Out[27]:
```
0.9210526315789473
```

There is no such difference in the score in using 'gini' or 'entropy'.

So to optimise our model lets tune the parameters of the decision tree.

Tuning min_samples_split

In [28]:
```
dt_model_entropy2 = DecisionTreeClassifier(criterion = 'entropy', min_samples_split= 50, random_state = 10)
dt_model_entropy2.fit(X_train, y_train)
# checking the training score
print('Accuracy of training set:', dt_model_entropy2.score(X_train, y_train))
# checking validation score
print('Accuracy of validation set: ', dt_model_entropy2.score(X_valid, y_valid))
```

```
Accuracy of training set: 0.9732142857142857
Accuracy of validation set:  0.9210526315789473
```

Tuning maximum_depth

In [29]:
```
dt_model_entropy3 = DecisionTreeClassifier(criterion = 'entropy', max_depth = 2, random_state = 10)
dt_model_entropy3.fit(X_train, y_train)
# checking the training score
print('Accuracy of training set:', dt_model_entropy3.score(X_train, y_train))
# checking validation score
print('Accuracy of validation set: ', dt_model_entropy3.score(X_valid, y_valid))
```

```
Accuracy of training set: 0.9732142857142857
Accuracy of validation set:  0.9210526315789473
```

Tuning max_leaf_node

In [30]:
```
dt_model_entropy4 = DecisionTreeClassifier(criterion = 'entropy', max_leaf_nodes = 3, random_state = 10)
dt_model_entropy4.fit(X_train, y_train)
# checking the training score
print('Accuracy of validation set: ', dt_model_entropy4.score(X_train, y_train))
# checking validation score
print('Accuracy of validation set: ', dt_model_entropy4.score(X_valid, y_valid))
```

```
Accuracy of validation set:  0.9732142857142857
Accuracy of validation set:  0.9210526315789473
```

Now we can see that the accuracies of training dataset and validation dataset are in sink as compare to the model we got in beginning.

Lets visualize our model (decision tree)

In [31]:
```
from sklearn import tree
```

In [32]:
```
!pip install graphviz
```

```
Requirement already satisfied: graphviz in c:\users\maniz\anaconda3\lib\site-packages (0.16)
```

In [33]:
```
conda install graphviz
```

```
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... done

# All requested packages already installed.


Note: you may need to restart the kernel to use updated packages.
```

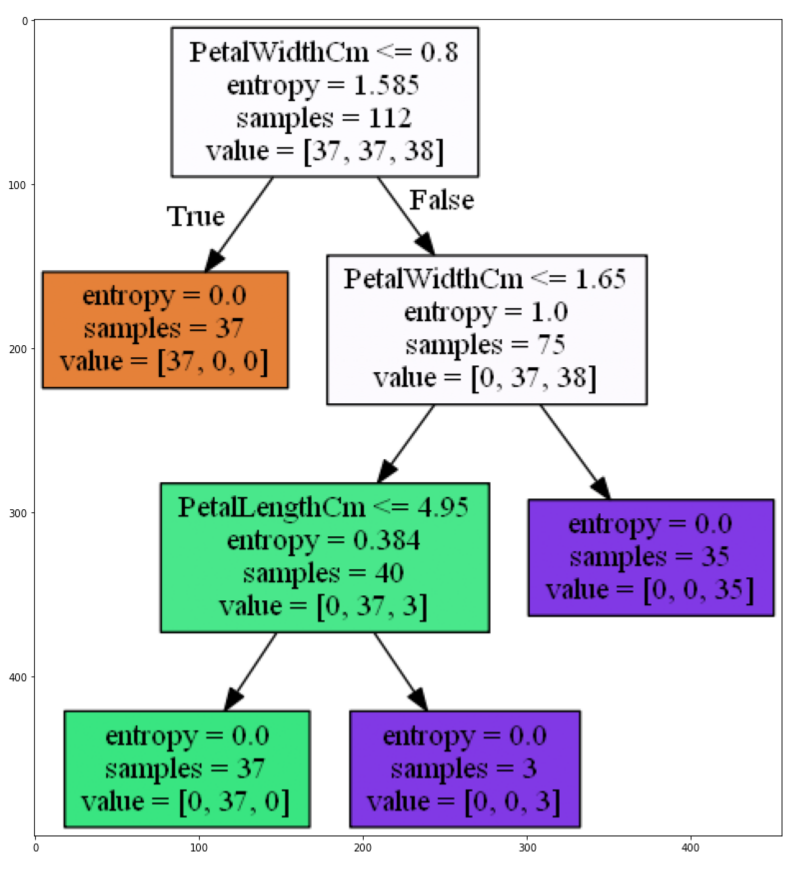
In [34]:
```
our_model_1 = tree.export_graphviz(dt_model_entropy, out_file= 'tree.dot', feature_names = X_train.columns, filled =True)
```

In [35]:
```
!dot -Tpng tree.dot -o tree.png
```

In [36]:
```
image = plt.imread('tree.png')
plt.figure(figsize=(15,15))
plt.imshow(image)

# General model without tuning any its feature
```

Out[36]:
```
<matplotlib.image.AxesImage at 0x2888a44e640>
```



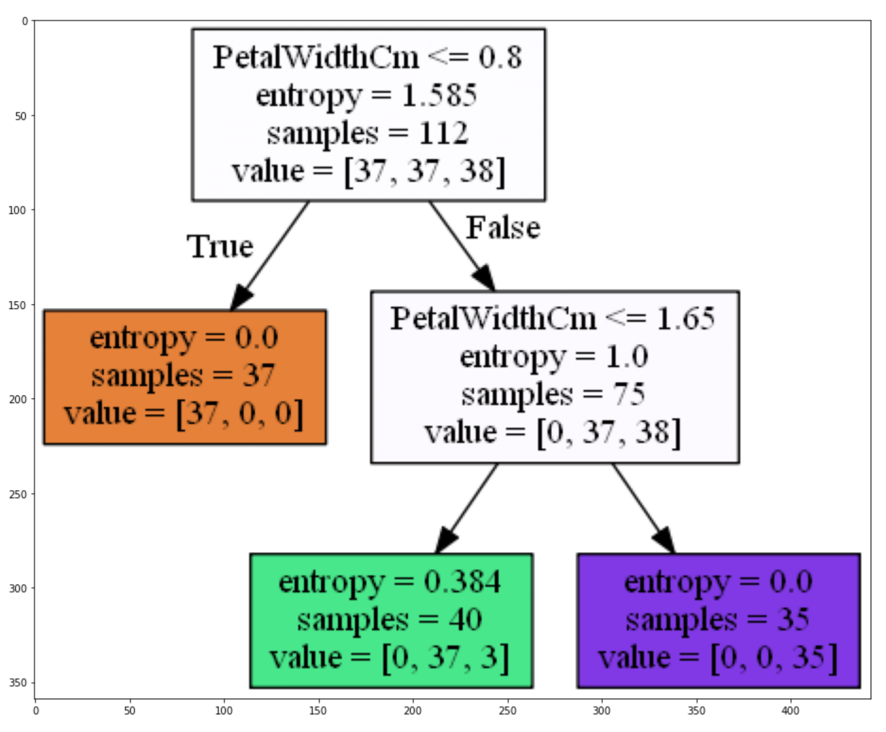
With min_samples_split= 50

In [99]:
```
our_model_2 = tree.export_graphviz(dt_model_entropy2, out_file= 'tree2.dot', feature_names = X_train.columns, filled =True)

!dot -Tpng tree2.dot -o tree2.png

image2 = plt.imread('tree2.png')
plt.figure(figsize=(15,15))
plt.imshow(image2)
```

Out[99]:
```
<matplotlib.image.AxesImage at 0x1c96d306430>
```



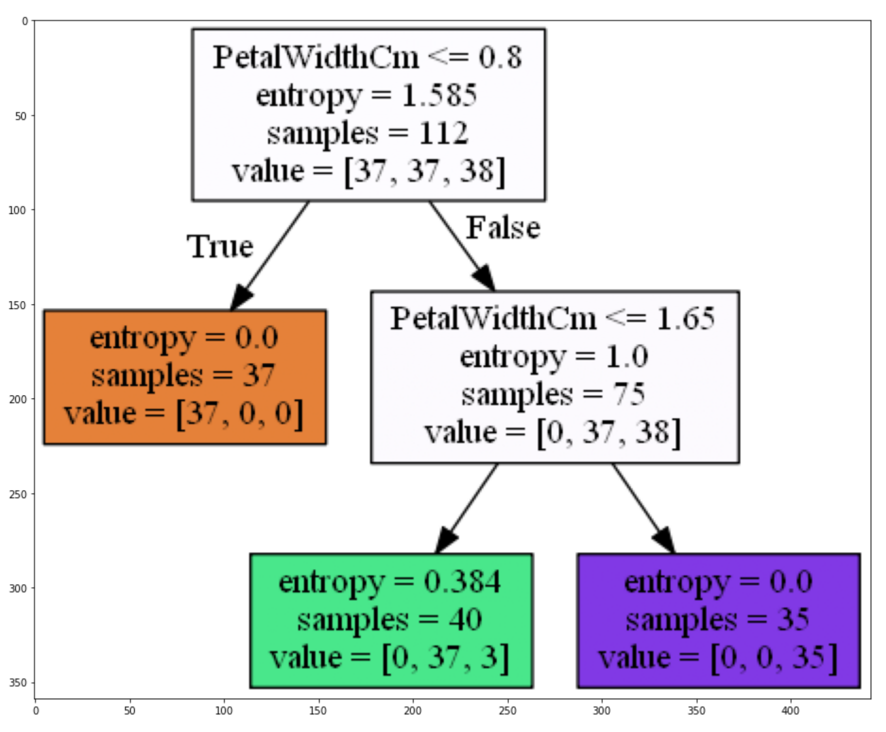
With max_depth = 2

In [100…
```
our_model_3 = tree.export_graphviz(dt_model_entropy3, out_file= 'tree3.dot', feature_names = X_train.columns, filled =True)

!dot -Tpng tree3.dot -o tree3.png

image3 = plt.imread('tree3.png')
plt.figure(figsize=(15,15))
plt.imshow(image3)
```

Out[100…
```
<matplotlib.image.AxesImage at 0x1c96d3a5ac0>
```



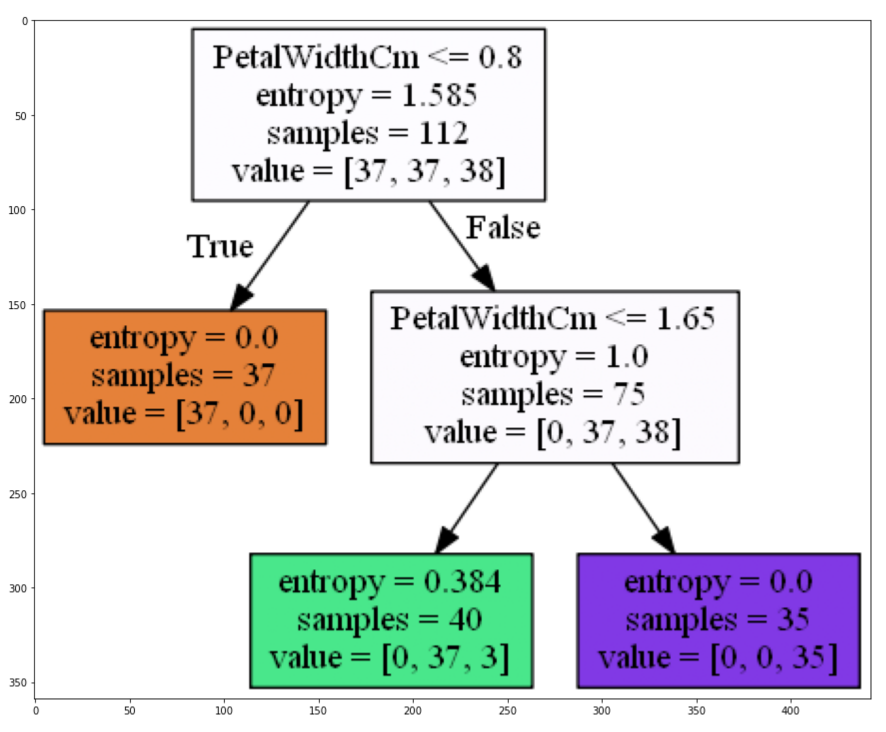
With max_leaf_nodes = 3

In [102…
```
our_model_4 = tree.export_graphviz(dt_model_entropy4, out_file= 'tree4.dot', feature_names = X_train.columns, filled =True)

!dot -Tpng tree4.dot -o tree4.png

image4 = plt.imread('tree4.png')
plt.figure(figsize=(15,15))
plt.imshow(image4)
```

Out[102…
```
<matplotlib.image.AxesImage at 0x1c96e057eb0>
```



In [ ]: