Decision Tree Classifier on IRIS Dataset

In [12]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [10]:
```
from sklearn.datasets import load_iris
```

In [55]:
```
import matplotlib.pyplot as plt
```

In [22]:
```
import pandas as pd
```

In [ ]:
```
iris = load_iris()
#in array form
iris
```

In [34]:
```
#converting array into a dataset using panda
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['target'] = iris.target
iris_df.sample(10)
```

Out[34]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
93	5.0	2.3	3.3	1.0	1
115	6.4	3.2	5.3	2.3	2
143	6.8	3.2	5.9	2.3	2
10	5.4	3.7	1.5	0.2	0
101	5.8	2.7	5.1	1.9	2
64	5.6	2.9	3.6	1.3	1
44	5.1	3.8	1.9	0.4	0
87	6.3	2.3	4.4	1.3	1
42	4.4	3.2	1.3	0.2	0
138	6.0	3.0	4.8	1.8	2

In [40]:
```
X=iris.data[:,2:]
y=iris.target
```

[:,:] comma se pehle ':' means seslect all rows, if we have 2:6, this mean select from the 3rd row (index 2) to the 6th row (just count rows, not try to go with the digit. mean (2+1) th row to (6-1) th row,since indexing starts at 0 and same for the columns.in this way [:,:-1] means all rows and all columns except the last column. i.e. IN GENERAL [a:b,c:d] MEANS FROM a th INDEX TO b-1 th INDEX AND FOR COLUMN FROM c th INDEX TO d-1 th INDEX

In [116…
```
#all parameters in Scikit Learn are max_depth,min_samples_leaf,min_samples_split
# Increasuing min_* hyperparameters and reducing max_* hyperparameters will regu
# https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeCla
```

```
tree_clf = DecisionTreeClassifier(max_depth = 2,)
tree_clf.fit(X,y)
```

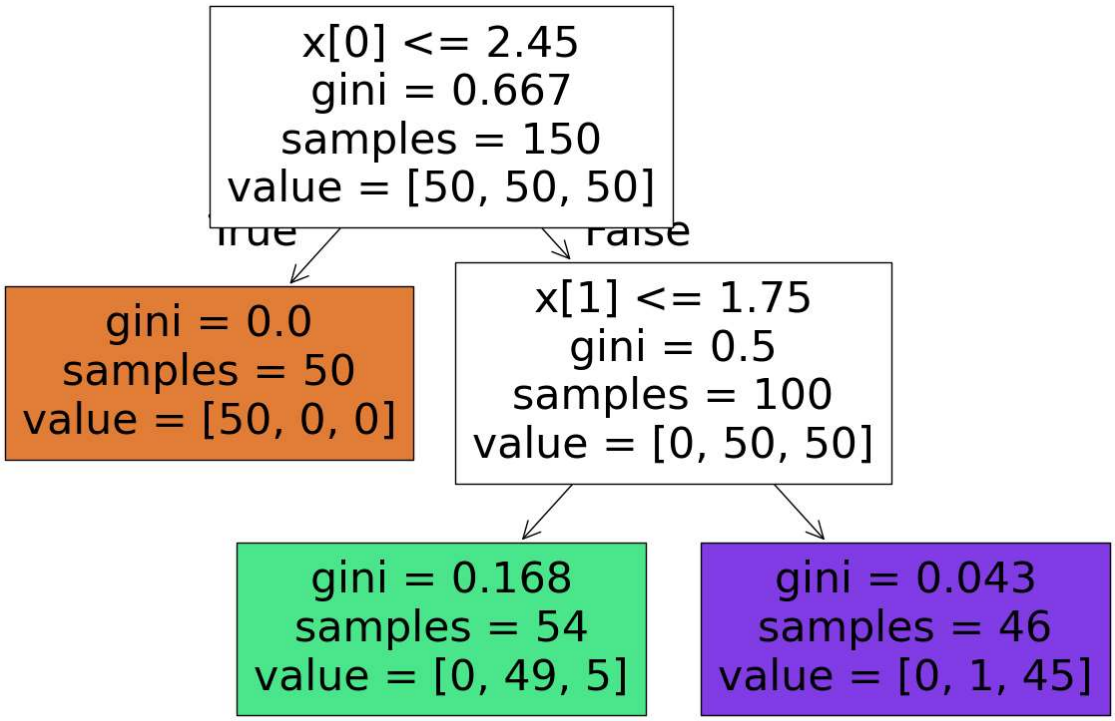
Out[116…

DecisionTreeClassifier

DecisionTreeClassifier(max_depth=2)

In [57]:
```
from sklearn import tree
plt.figure(figsize=(15,10))
tree.plot_tree(tree_clf,filled=True)
```

Out[57]:
```
[Text(0.4, 0.8333333333333334, 'x[0] <= 2.45\ngini = 0.667\nsamples = 150\nvalu
e = [50, 50, 50]'),
 Text(0.2, 0.5, 'gini = 0.0\nsamples = 50\nvalue = [50, 0, 0]'),
 Text(0.30000000000000004, 0.6666666666666667, 'True  '),
 Text(0.6, 0.5, 'x[1] <= 1.75\ngini = 0.5\nsamples = 100\nvalue = [0, 50, 5
0]'),
 Text(0.5, 0.6666666666666667, '  False'),
 Text(0.4, 0.16666666666666666, 'gini = 0.168\nsamples = 54\nvalue = [0, 49,
5]'),
 Text(0.8, 0.16666666666666666, 'gini = 0.043\nsamples = 46\nvalue = [0, 1, 4
5]')]
```



In [61]:
```
tree_clf.predict_proba([[5,1.5]])
```

Out[61]:
```
array([[0.        , 0.90740741, 0.09259259]])
```

In [63]:
```
tree_clf.predict([[5,1.5]])
```

Out[63]:
```
array([1])
```

Regularization

Regularization in machine learning is a technique used to prevent overfitting and improve the generalization of machine learning models. Overfitting occurs when a model learns not only the underlying pattern in the training data but also noise and random fluctuations. This causes the model to perform well on the training data but poorly on unseen data (i.e., it doesn't generalize well).

Purpose of Regularization: The primary goal of regularization is to impose constraints on a machine learning model so that it doesn't become too complex, thereby reducing the risk of overfitting. Regularization techniques penalize the complexity of the model, encouraging it to favor simpler models that generalize better.

Common Regularization Techniques:

1. L2 Regularization (Ridge Regression):

   > Involves adding a penalty equivalent to the square of the magnitude of coefficients to the loss function. This encourages the model to keep the coefficients small, effectively reducing model complexity. Helps in preventing large coefficients that can lead to overfitting.

2. L1 Regularization (Lasso Regression):

   > Adds a penalty equivalent to the absolute value of the magnitude of coefficients to the loss function. Encourages sparse models by pushing some coefficients to zero, effectively performing feature selection. Useful when you suspect that many of the features are irrelevant.

3. Dropout (Neural Networks):

   > A regularization technique specifically used in neural networks. Randomly drops neurons (along with their connections) during training to prevent neurons from co-adapting too much to each other. Helps in improving the generalization of neural networks.

Regularization in machine learning is important because it helps models to be more reliable and work better on new data. It does this by encouraging simpler models and preventing them from memorizing too much detail from the training data, which can cause problems when trying to predict on data it hasn't seen before.

Overfitting

Overfitting happens when a model tries too hard to fit every detail in its training data, including the noise or random variations that aren't actually part of the real pattern.

Example: Imagine you're trying to recognize different types of dogs based on pictures. If you have a very detailed model, it might learn specific features like the exact shapes of spots or hair patterns that are unique to each dog in your training set. However, some of these features might be random and not actually helpful for recognizing dog breeds in

general. As a result, your model might do great on the dogs it's seen before (training data), but it may struggle with new dogs (test data) because it's too focused on small details that don't matter.

Noise

Noise or random variations refer to unpredictable and irregular fluctuations or disturbances in data that do not represent meaningful patterns or relationships. Here's a brief example:

Example: Imagine you're measuring the height of students in a classroom. Due to various factors like measurement errors, different shoes, or slight posture changes, the recorded heights may vary slightly from the true heights. These variations are considered noise or random fluctuations because they don't reflect any systematic or meaningful differences in height among the students.

Gini Index

The Gini index is a measure of impurity or uncertainty used in decision trees to decide where to split the data.

Entropy

Entropy measures the impurity or randomness in the dataset. In decision trees, it helps to find the best feature to split the data based on the information gain.

Differences between Gini Index & Entropy

Definition:

Gini Index: Measures the impurity of a dataset. A lower Gini index indicates a purer split. Entropy: Measures the uncertainty or disorder of a dataset. A lower entropy value indicates less randomness and more certainty in the data. **Range:**

Gini Index: Ranges from 0 (perfect purity) to 0.5 (maximum impurity for a binary classification). Entropy: Ranges from 0 (perfect purity) to 1 (maximum disorder for a binary classification). **Calculation Complexity:**

Gini Index: Simpler to calculate since it only involves squaring probabilities. Entropy: Slightly more complex as it involves logarithmic calculations. **Which is Better to Use?** No Clear Winner: There isn't a definitive answer to which one is better, as it often depends on the specific problem and dataset. **Common Choice:** Gini Index is often preferred because it is simpler and faster to compute. Entropy can sometimes give more nuanced splits, especially when there is a significant imbalance in class distribution.

In [ ]: