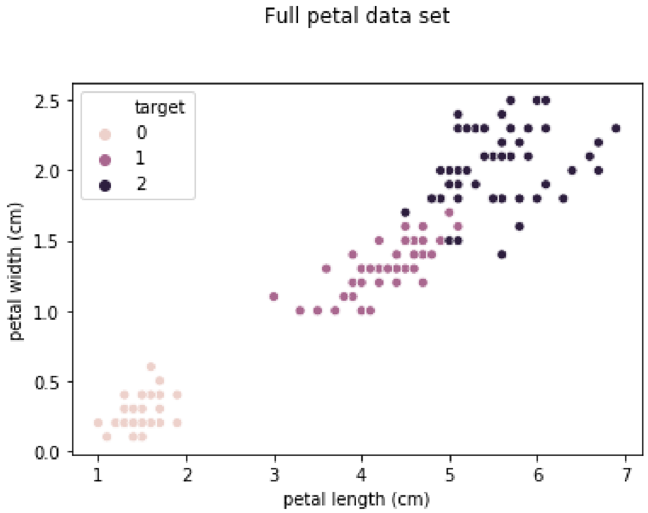
Simple example of decision-tree classification, using the Iris dataset

In [1]:
```
import graphviz
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
```

To simplify things, we'll use just the data on petal length and width, and ignore data on sepal length and width for now.

In [2]:
```
r = load_iris()
raw = pd.DataFrame(r['data'])
raw.columns = r['feature_names']
raw['target'] = pd.Series(r['target'])
petals=raw.loc[:,['petal length (cm)','petal width (cm)','target']]
petals
sns.scatterplot(x=petals['petal length (cm)'],y=petals['petal width (cm)'], hue=petals[
'target']).set_title('\n\nFull petal data set\n\n')
```

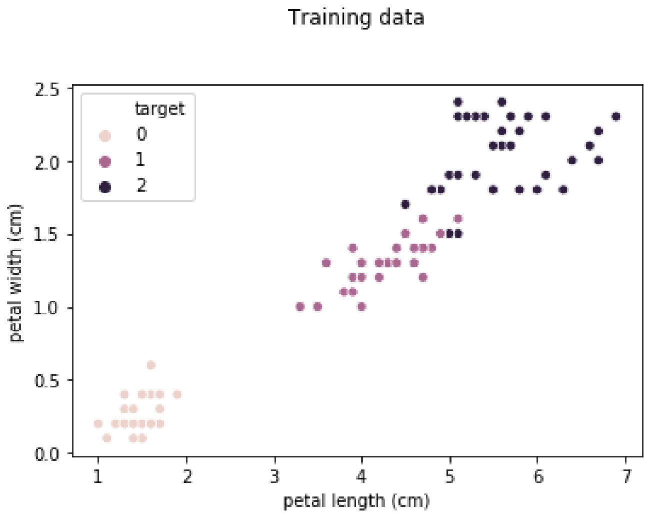
Out[2]:
```
Text(0.5, 1.0, '\n\nFull petal data set\n\n')
```



Create random training and test sets from the full dataset, using 2/3 of the dataset to train and 1/3 to test:

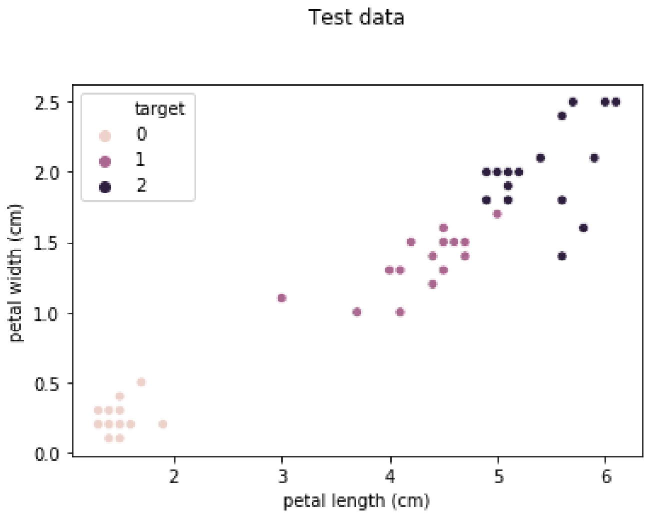
In [8]:
```
data_train, data_test, target_train, target_test = train_test_split(
                                    petals[['petal length (cm)' ,'petal width (cm)']]
                                    ,petals['target']
                                    ,stratify=petals['target']
                                    ,test_size = 0.33
                                    ,random_state = 0)

sns.scatterplot(x=data_train['petal length (cm)'],y=data_train['petal width (cm)'],hue=t
arget_train).set_title('\n\nTraining data\n\n')
```

Out[8]:
```
Text(0.5, 1.0, '\n\nTraining data\n\n')
```



In [4]:
```
sns.scatterplot(x=data_test['petal length (cm)'],y=data_test['petal width (cm)'],hue=tar
get_test).set_title('\n\nTest data\n\n')
```

Out[4]:
```
Text(0.5, 1.0, '\n\nTest data\n\n')
```



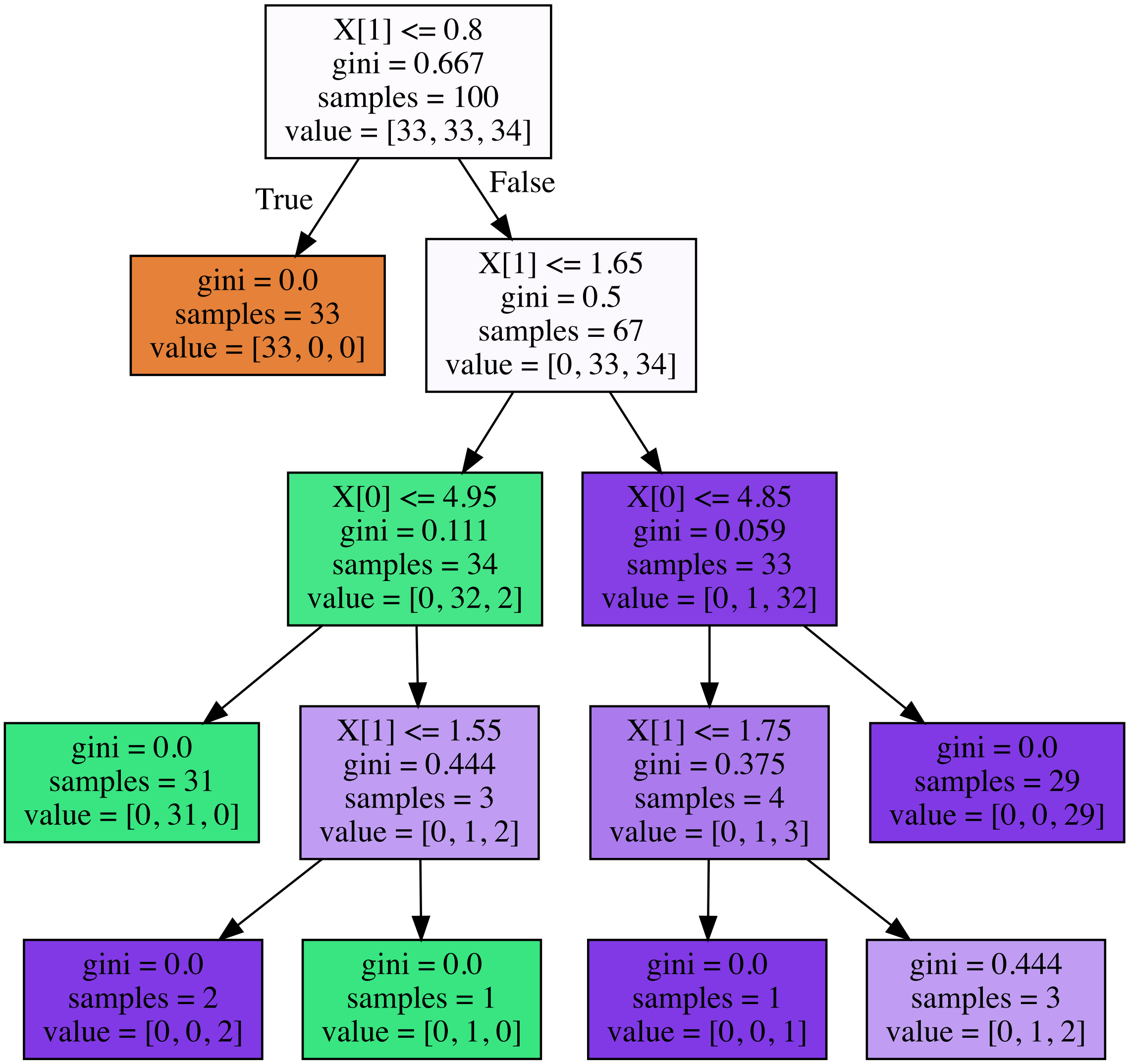
Create a decision tree classifier:

In [5]:

```
iris_tree = DecisionTreeClassifier(random_state=0)
iris_tree.fit(data_train,target_train)
test_score = iris_tree.score(data_test,target_test)
training_score = iris_tree.score(data_train,target_train)
print(f"""\n\nIf we allow the tree to grow as complex as possible (no regularization), we can see that it develops several layers
and displays a modest amount of overfitting, with the model fitting the training data very well with a score of {training_score},
and the model fitting the test data somewhat less well, with a score of {test_score}:\n\n""")
export_graphviz(iris_tree,out_file = 'iris_tree.dot',filled=True)
with open('iris_tree.dot') as f:
    iris_tree_graph = f.read()
print("""Here's what the top of the dot file looks like, by the way-- not too hard to read:\n\n""" +iris_tree_graph[:200]+'\n\n\n')
graphviz.Source(iris_tree_graph)
```

```
If we allow the tree to grow as complex as possible (no regularization), we can see tha
t it develops several layers
and displays a modest amount of overfitting, with the model fitting the training data v
ery well with a score of 0.99,
and the model fitting the test data somewhat less well, with a score of 0.96:


Here's what the top of the dot file looks like, by the way-- not too hard to read:

digraph Tree {
node [shape=box, style="filled", color="black"] ;
0 [label="X[1] <= 0.8\ngini = 0.667\nsamples = 100\nvalue = [33, 33, 34]", fillcolor="#
8139e504"] ;
1 [label="gini = 0.0\nsamples = 33\
```

Out[5]:

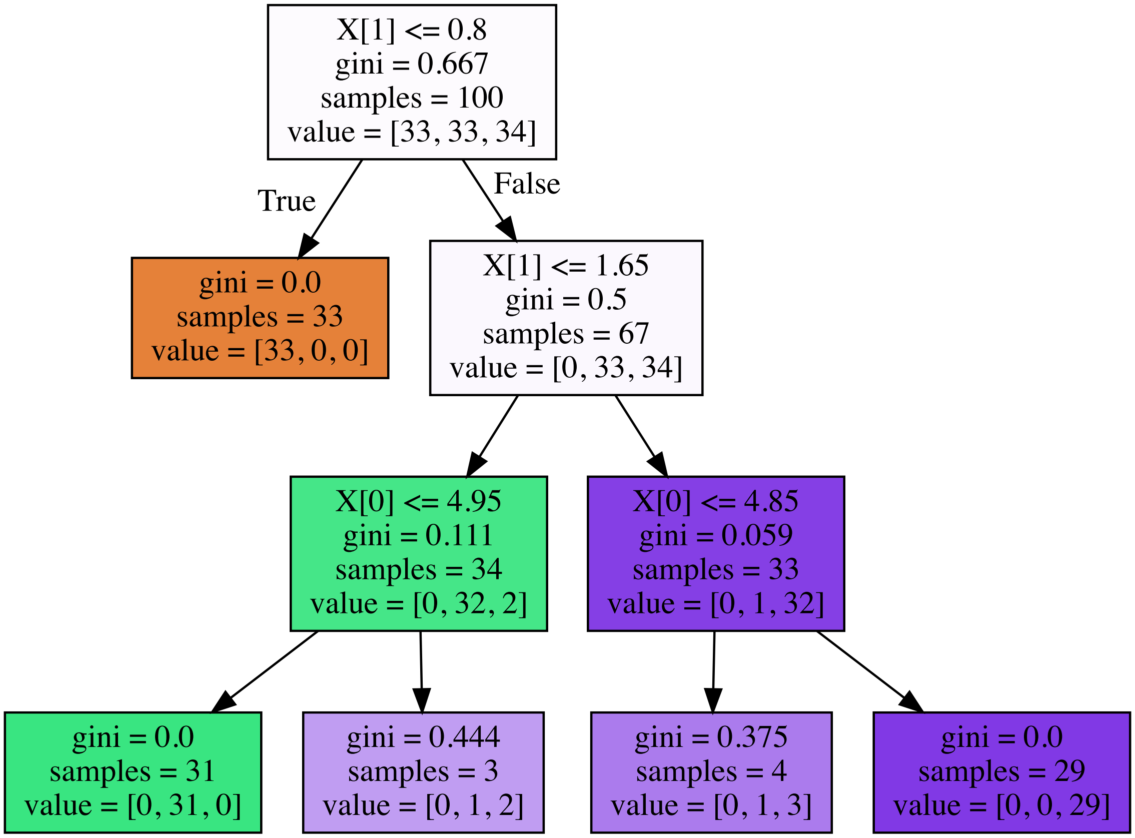


In [7]:

```
# Now with some regularization, restricting the depth of the tree to a certain number of
decision points:

iris_tree = DecisionTreeClassifier(random_state=0,max_depth=3)
iris_tree.fit(data_train,target_train)
test_score = iris_tree.score(data_test,target_test)
training_score = iris_tree.score(data_train,target_train)
export_graphviz(iris_tree,out_file = 'iris_tree.dot',filled=True)
with open('iris_tree.dot') as f:
    iris_tree_graph = f.read()
print(f"""And here's the model with its maximum depth constrained, so that only certain
 number of decision points are allowed.
In this case, the training score is {training_score} and the test score is {test_score}.
""")
graphviz.Source(iris_tree_graph)
```

```
And here's the model with its maximum depth constrained, so that only certain number of
decision points are allowed.
In this case, the training score is 0.98 and the test score is 0.98.
```

Out[7]:



In [ ]:

In [ ]: