Decision Trees

- DecisionTreeClassifier (multi-class calssification)

- !pip install --upgrade scikit-learn
- !pip install --upgrade graphviz

```
In [1]: from sklearn.tree import DecisionTreeClassifier
        from sklearn.tree import export_graphviz, plot_tree
```

```
In [2]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import matplotlib.image as img
        import seaborn as sns
```

```
In [3]: X = [[0,0], [1,1]]
        y = [0, 1]
        clf = DecisionTreeClassifier()
        clf = clf.fit(X, y)
```

```
In [4]: clf.predict([[-1,0],[0,1],[2,2], [3,3]])
```

```
Out[4]: array([0, 1, 1, 1])
```

```
In [5]: clf.predict_proba([[-1,0],[0,1],[2,2], [3,3]])
```

```
Out[5]: array([[1., 0.],
               [0., 1.],
               [0., 1.],
               [0., 1.]])
```

Iris Dataset

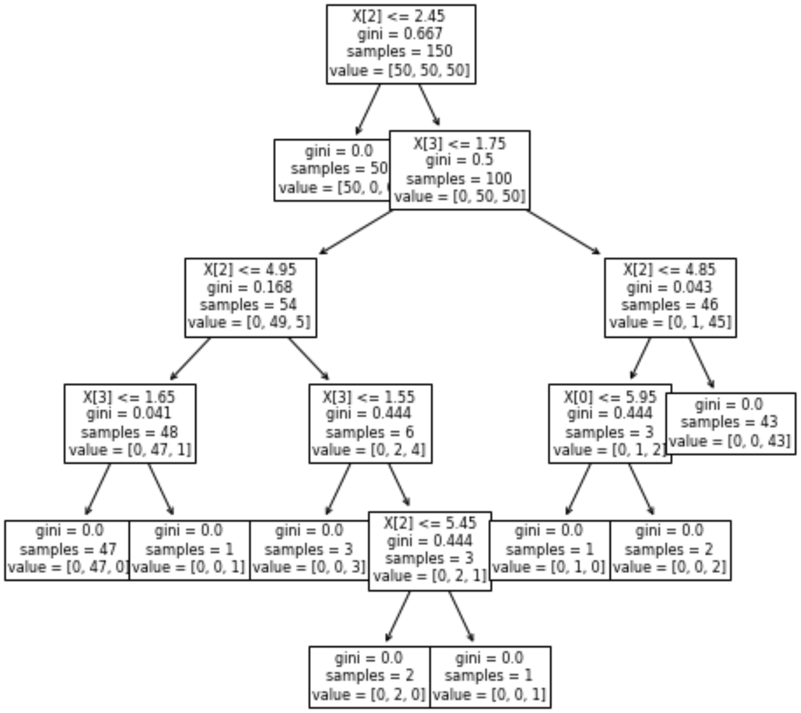
```
In [6]: from sklearn.datasets import load_iris
```

In [7]:
```
iris = load_iris()
iris.data[:10]
```

Out[7]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1]])
```

In [8]:
```
clf1 = DecisionTreeClassifier()
clf1 = clf.fit(iris.data, iris.target)
```

In [9]:
```
plt.figure(figsize=(8,8))
plot_tree(clf1, fontsize=8);
```



In [10]:
```
import graphviz
```

In [11]:
```
dot_data = export_graphviz(clf1, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("iris")
```

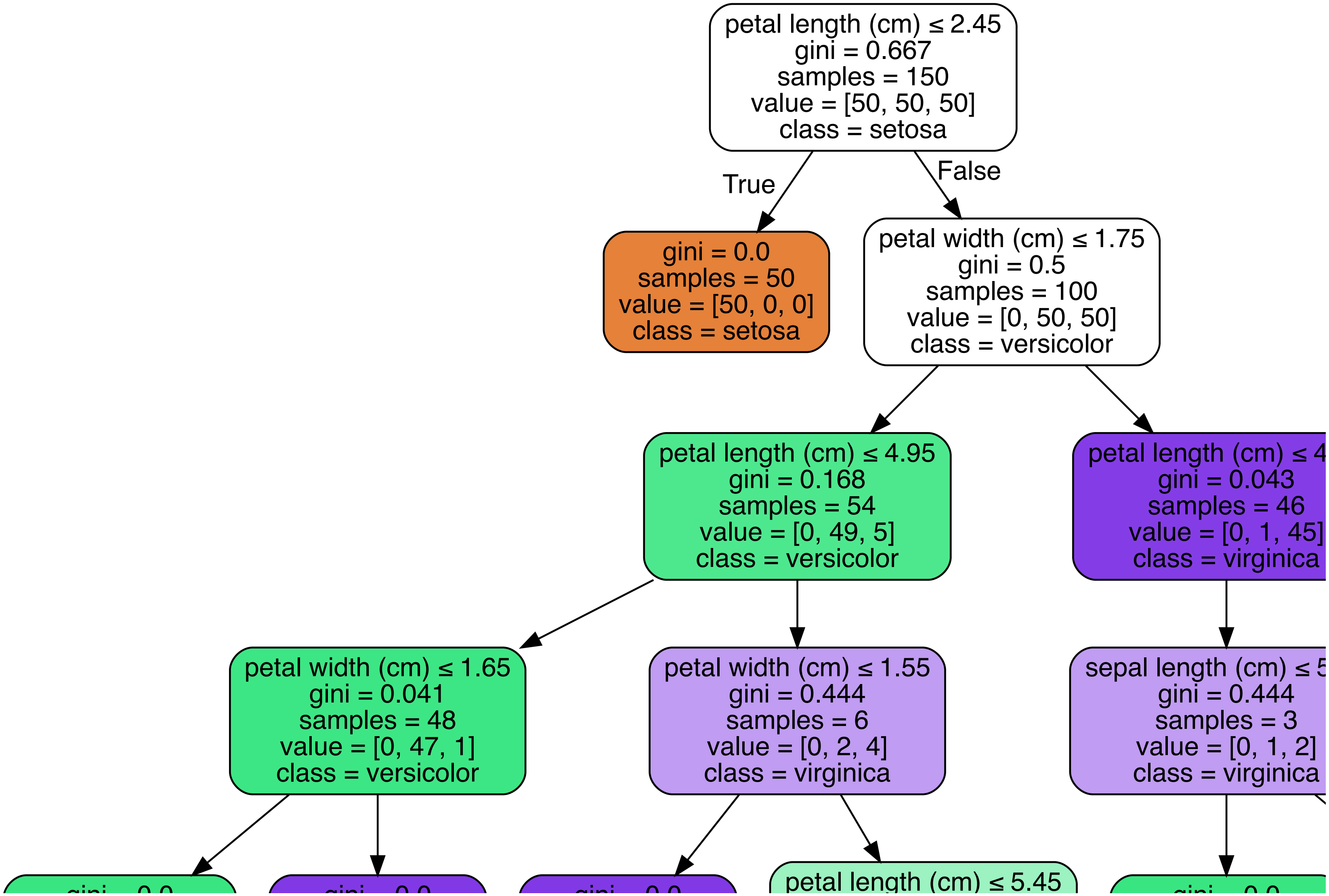
Out[11]:
```
'iris.pdf'
```

In [12]:
```
!open iris.pdf
```

In [ ]:

In [13]:
```
dot_data = export_graphviz(clf1, out_file=None,
                           feature_names=iris.feature_names,
                            class_names=iris.target_names,
                            filled=True, rounded=True,
                            special_characters=True)

graph = graphviz.Source(dot_data, format="png")
graph
```

Out[13]:



Model Evaluation

In [14]:
```
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

In [15]:
```
X = iris.data
y = iris.target
```

In [16]:
```
# 70% training set, 30% testing set

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

In [17]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[17]:
```
((105, 4), (45, 4), (105,), (45,))
```

In [18]:
```
clf = DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
```

In [19]:
```
# Predict the response

y_pred = clf.predict(X_test)
y_pred
```

Out[19]:
```
array([0, 1, 1, 0, 2, 1, 2, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       2, 0, 2, 1, 0, 0, 1, 2, 1, 2, 1, 2, 2, 0, 1, 0, 1, 2, 2, 0, 1, 2,
       1])
```

In [20]:
```
clf.predict_proba(X_test)
```

Out[20]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
```

In [21]:

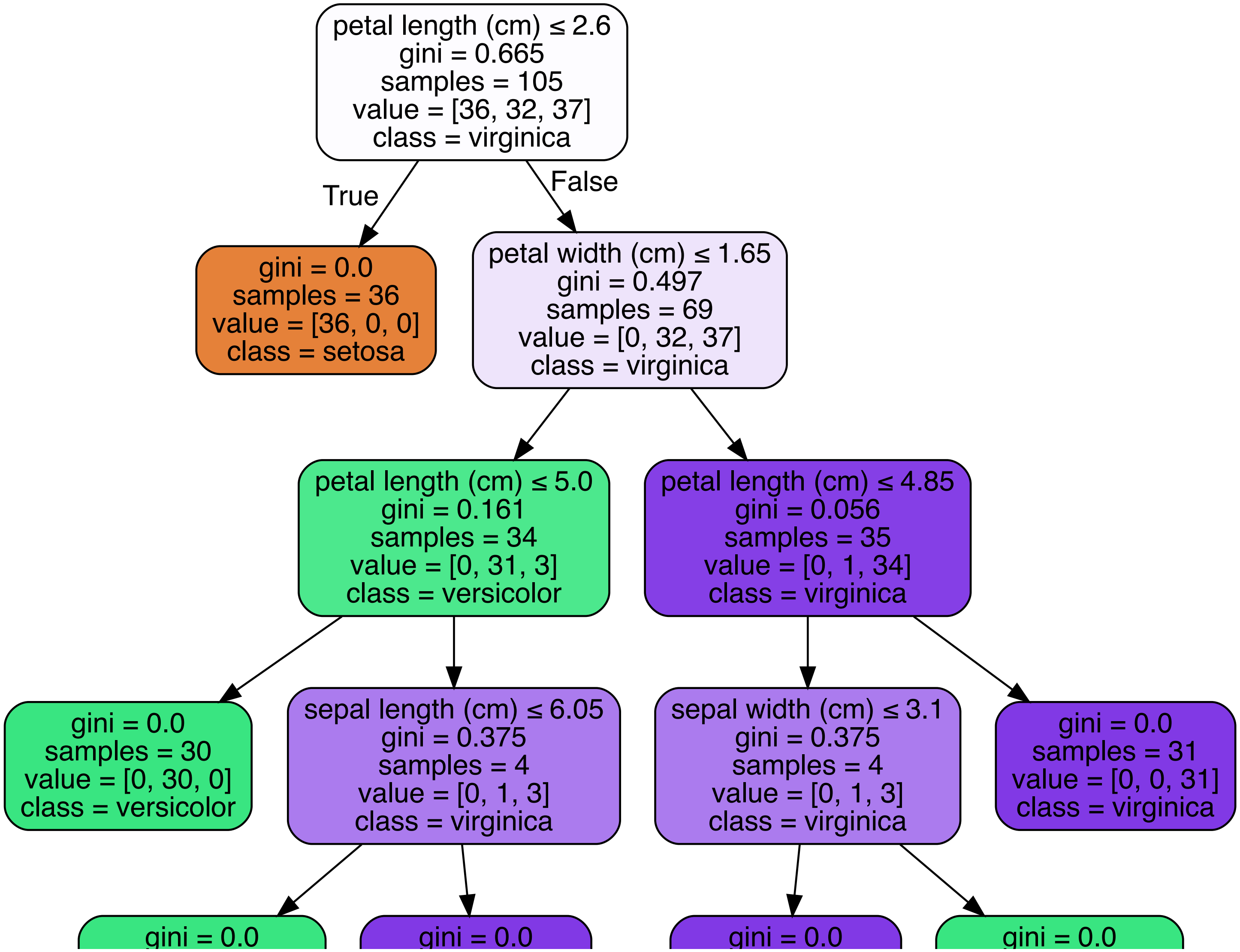
```
# Model accuracy

metrics.accuracy_score(y_test, y_pred)
```

Out[21]:

```
0.9555555555555556
```

In [22]:
```
dot_data = export_graphviz(clf, out_file=None,
                          feature_names=iris.feature_names,
                           class_names=iris.target_names,
                           filled=True, rounded=True,
                           special_characters=True)
graph = graphviz.Source(dot_data)

graph
```

Out[22]:



Pandas Data Frame

In [23]:

```python
iris = sns.load_dataset("iris")
iris.head()
```

Out[23]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [24]:

```python
feature_names = iris.columns[:-1]
feature_names
```

Out[24]:

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width'], dtype='object')
```

In [25]:

```python
class_column = iris.columns[-1]
class_column
```

Out[25]:

```
'species'
```

In [26]:

```python
class_names = iris[class_column].unique()
class_names
```

Out[26]:

```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

In [27]:

```python
X = iris[feature_names]
X.head()
```

Out[27]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [28]:

```
y = iris[class_column]
y.head()
```

Out[28]:

```
0    setosa
1    setosa
2    setosa
3    setosa
4    setosa
Name: species, dtype: object
```

In [29]:

```
# 70% training set, 30% testing set

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

In [30]:

```
X_train.head()
```

Out[30]:

	sepal_length	sepal_width	petal_length	petal_width
118	7.7	2.6	6.9	2.3
18	5.7	3.8	1.7	0.3
4	5.0	3.6	1.4	0.2
45	4.8	3.0	1.4	0.3
59	5.2	2.7	3.9	1.4

In [31]:

```
X_test.head()
```

Out[31]:

	sepal_length	sepal_width	petal_length	petal_width
14	5.8	4.0	1.2	0.2
98	5.1	2.5	3.0	1.1
75	6.6	3.0	4.4	1.4
16	5.4	3.9	1.3	0.4
131	7.9	3.8	6.4	2.0

In [32]:

```
clf = DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
```

In [33]:
```
# Predict the response

y_pred = clf.predict(X_test)
y_pred
```

Out[33]:
```
array(['setosa', 'versicolor', 'versicolor', 'setosa', 'virginica',
       'versicolor', 'virginica', 'setosa', 'setosa', 'virginica',
       'versicolor', 'setosa', 'virginica', 'versicolor', 'versicolor',
       'setosa', 'versicolor', 'versicolor', 'setosa', 'setosa',
       'versicolor', 'versicolor', 'virginica', 'setosa', 'virginica',
       'versicolor', 'setosa', 'setosa', 'versicolor', 'virginica',
       'versicolor', 'virginica', 'versicolor', 'virginica', 'virginica',
       'setosa', 'versicolor', 'setosa', 'versicolor', 'virginica',
       'virginica', 'setosa', 'versicolor', 'virginica', 'versicolor'],
      dtype=object)
```

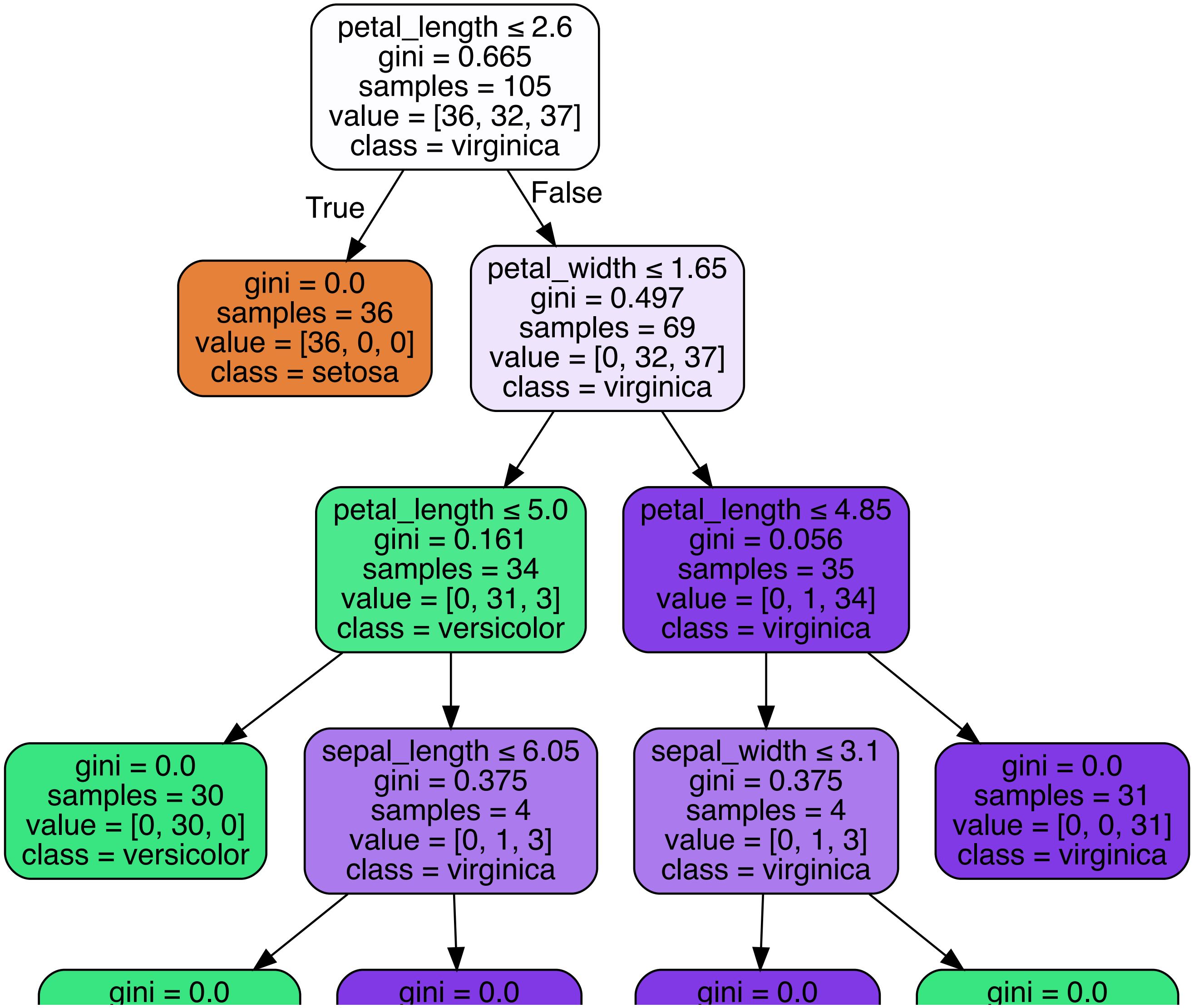
In [34]:
```
# Model accuracy

metrics.accuracy_score(y_test, y_pred)
```

Out[34]:
```
0.9555555555555556
```

In [35]:

```
dot_data = export_graphviz(clf, out_file=None,
                          feature_names=feature_names,
                           class_names=class_names,
                           filled=True, rounded=True,
                           special_characters=True)
graph = graphviz.Source(dot_data)

graph
```

Out[35]:



- Default splitting criteria is Gini
- Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.

In [36]:

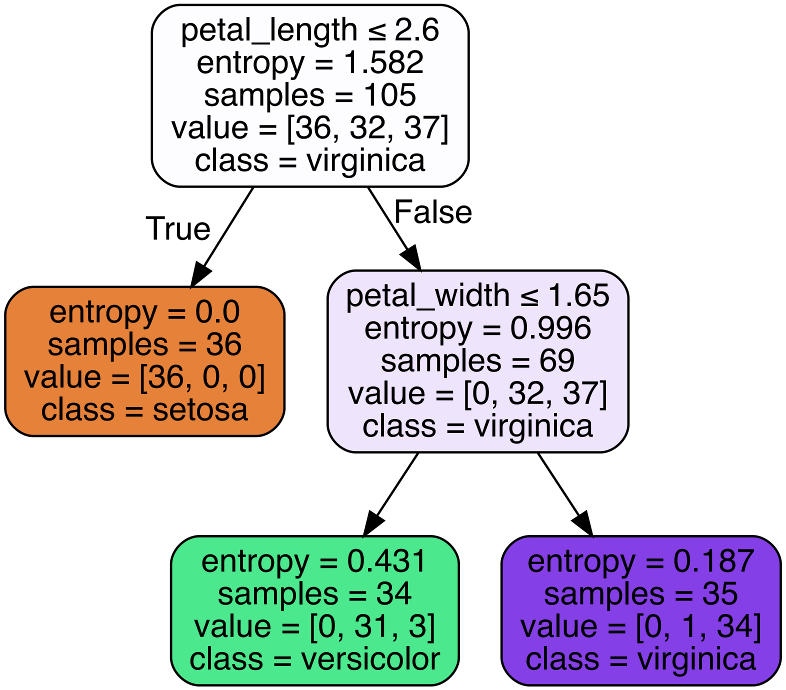
```
clf = DecisionTreeClassifier(criterion="entropy", max_depth=2)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9555555555555556
```

In [37]:

```
dot_data = export_graphviz(clf, out_file=None,
                           feature_names=feature_names,
                            class_names=class_names,
                            filled=True, rounded=True,
                            special_characters=True)
graph = graphviz.Source(dot_data)

graph
```

Out[37]:



In [ ]:

Decision Surface

In [38]:

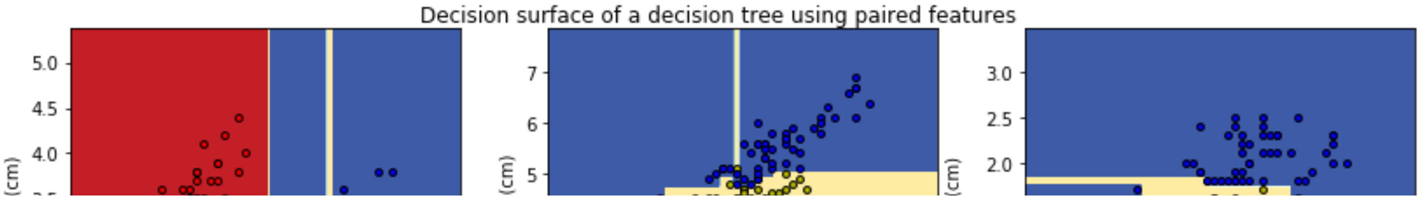
```
iris = load_iris()
```

In [39]:

```
n_classes = 3
plot_colors = "ryb"
plot_step = 0.02
```

In [40]:

```
fig, ax = plt.subplots(2, 3, figsize=(12,8))

for pairidx, pair in enumerate([[0, 1], [0, 2], [0, 3],
                                [1, 2], [1, 3], [2, 3]]):
    # We only take the two corresponding features
    X = iris.data[:, pair]
    y = iris.target

    # Train
    clf = DecisionTreeClassifier().fit(X, y)

    # Plot the decision boundary
    plt.subplot(2, 3, pairidx + 1)

    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                         np.arange(y_min, y_max, plot_step))
    plt.tight_layout(h_pad=0.5, w_pad=0.5, pad=2.5)

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    cs = plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)

    plt.xlabel(iris.feature_names[pair[0]])
    plt.ylabel(iris.feature_names[pair[1]])

    # Plot the training points
    for i, color in zip(range(n_classes), plot_colors):
        idx = np.where(y == i)
        plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i],
                    cmap=plt.cm.RdYlBu, edgecolor='black', s=15)

plt.suptitle("Decision surface of a decision tree using paired features")
plt.legend(loc='lower right', borderpad=0, handletextpad=0)
plt.axis("tight")
plt.show()
```


Decision surface of a decision tree using paired features
(cm)
(cm)
(cm)