In [1]:
```
'''
    Author: A.Shrikant
'''
```
Out[1]:
```
'\n    Author: A.Shrikant\n'
```
In [2]:
```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.tree import export_text
```
In [3]:
```
# Load the iris dataset.

iris = load_iris()
iris
```
Out[3]:
```
{'data': 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],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
```
In [4]:
```
type(iris)
```
Out[4]:
```
sklearn.utils.Bunch
```
In [5]:
```
# Separating the features and the dependent variables.
X1 = iris.data[:,:]
y = iris.target
```
In [6]:
```
X1
```
Out[6]:
```
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],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
```
In [7]:
```
y
```
Out[7]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [8]:
```
X1.shape
```
Out[8]:
```
(150, 4)
```
In [9]:
```
iris.feature_names
```
Out[9]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```
In [10]:
```
iris.target_names
```
Out[10]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```
In [11]:
```
df = pd.concat([pd.DataFrame(X1, columns=iris.feature_names), pd.DataFrame(y, columns=['species'])], axis=1)
df.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   species            150 non-null    int32  
dtypes: float64(4), int32(1)
memory usage: 5.4 KB
```

No missing values found in the dataset.

In [12]:
```
df.head()
```
Out[12]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [13]:
```
# The dataset is balanced since the different classes are present in equal proportion.

df['species'].value_counts()
```
Out[13]:
```
0    50
1    50
2    50
Name: species, dtype: int64
```
In [14]:
```
df.describe()
```
Out[14]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

Not treating outliers because the range of each feature variable is very narrow.

Splitting the data into train and test:

In [15]:
```
X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(X1, y, test_size=0.25, stratify=y, random_state=1234)
```

Building the Decision Tree based model:

A Decision Tree is a supervised machine-learning algorithm that can be used for both the Classification and Regression types of tasks. The topmost node in a Decision Tree is the **root node**, the nodes that don't have any children are the **leaf nodes**, and the remaining nodes are called the **internal nodes**.

When Decision Tree is used for Classification it is called a **Classification Tree**. In Classification Tree at each node the best question is asked based on the training dataset attributes/features such that it leads to the best segregation of the data points according to their class labels.

To measure the quality of splits at each node there are several impurity measuring techniques:

- **Gini Index**
- **Entropy**
- **Information Gain**

If all the data points falling at a node are of the same class then the impurity is minimum for that node, instead if all datapoints of different class labels are present in equal amounts then the impurity is maximum for that node.

When Decision Tree is used for Regression it is called a **Regression Tree**. In Regression Tree at each node, the best feature and a threshold associated with that feature are selected which gives minimum **SSR(Sum of Squared Residuals)** for the samples in the left and right child nodes of that node.

To control the height/depth of the tree we have several hyperparameters for pruning the tree like:

- **max_depth** - Specifying the upper limit for the tree depth.
- **min_samples_split** - Specifying the minimum number of training samples to be there in a node to consider it for further splitting.
- **min_samples_leaf** - Specifying the minimum number of training samples to be there in the left and right child nodes of an internal node after applying the split on that internal node. If the split results in less samples in left and right child nodes than the one specified by **min_samples_leaf** then this split will not be considered.

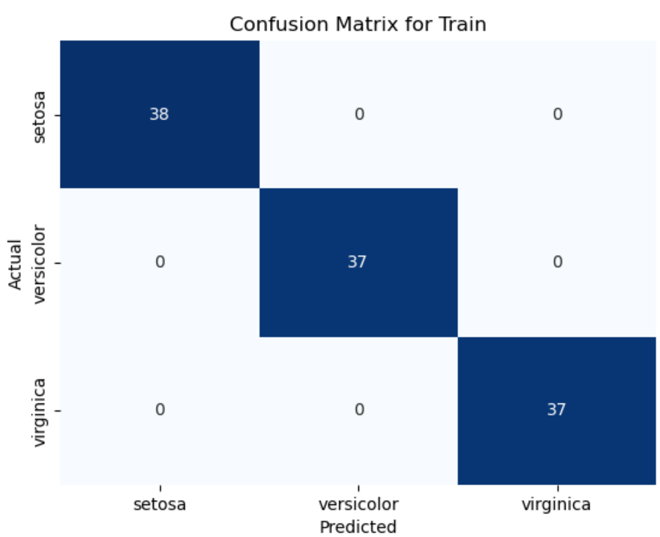
In [16]:
```
dt_classifier_1 = DecisionTreeClassifier()
dt_classifier_1.fit(X_train_1, y_train_1)
```

Out[16]:
```
DecisionTreeClassifier()
```

In [17]:
```
# Making predictions.

y_pred_train_1 = dt_classifier_1.predict(X_train_1)
y_pred_test_1 = dt_classifier_1.predict(X_test_1)
```

In [18]:
```
# https://stackoverflow.com/questions/54506626/how-to-understand-seaborns-heatmap-annotation-format

def draw_confusion_matrix(y_true, y_pred, c_matrix_for):
    labels = ['setosa', 'versicolor', 'virginica']
    sns.heatmap(confusion_matrix(y_true, y_pred), annot=True, fmt='.3g', xticklabels=labels,
            yticklabels=labels, cmap='Blues', cbar=False)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'Confusion Matrix for {c_matrix_for}')
    plt.show()
```

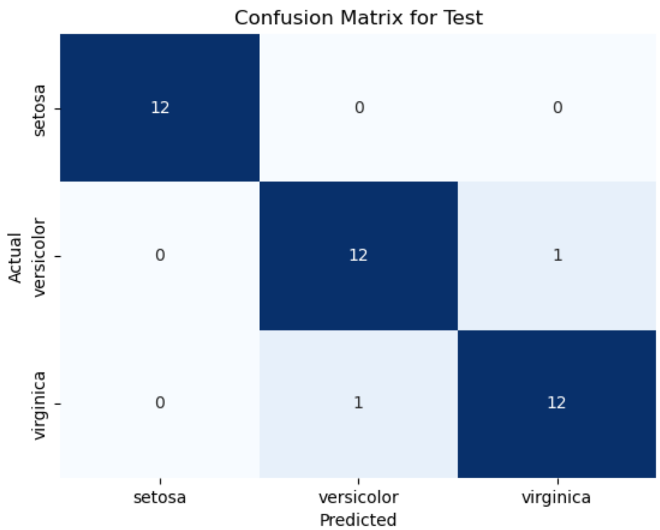
In [19]:
```
draw_confusion_matrix(y_train_1, y_pred_train_1, c_matrix_for='Train')
```



In [20]:
```python
print('For Train data:')
print(classification_report(y_train_1, y_pred_train_1))
```

```
For Train data:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        38
           1       1.00      1.00      1.00        37
           2       1.00      1.00      1.00        37

    accuracy                           1.00       112
   macro avg       1.00      1.00      1.00       112
weighted avg       1.00      1.00      1.00       112
```

In [21]:
```python
draw_confusion_matrix(y_test_1, y_pred_test_1, c_matrix_for='Test')
```



In [22]:
```python
print('For Test data:')
print(classification_report(y_test_1, y_pred_test_1))
```

```
For Test data:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.92      0.92      0.92        13
           2       0.92      0.92      0.92        13

    accuracy                           0.95        38
   macro avg       0.95      0.95      0.95        38
weighted avg       0.95      0.95      0.95        38
```

In [23]:
```python
print(f'Train Accuracy: {accuracy_score(y_train_1, y_pred_train_1)}')
print(f'Test Accuracy: {accuracy_score(y_test_1, y_pred_test_1)}')
```

```
Train Accuracy: 1.0
Test Accuracy: 0.9473684210526315
```

In [24]:
```python
from sklearn.model_selection import cross_val_score
```

In [25]:
```python
accuracy_decision_tree_1 = cross_val_score(dt_classifier_1, X_train_1, y_train_1, cv=10)
accuracy_decision_tree_1
```

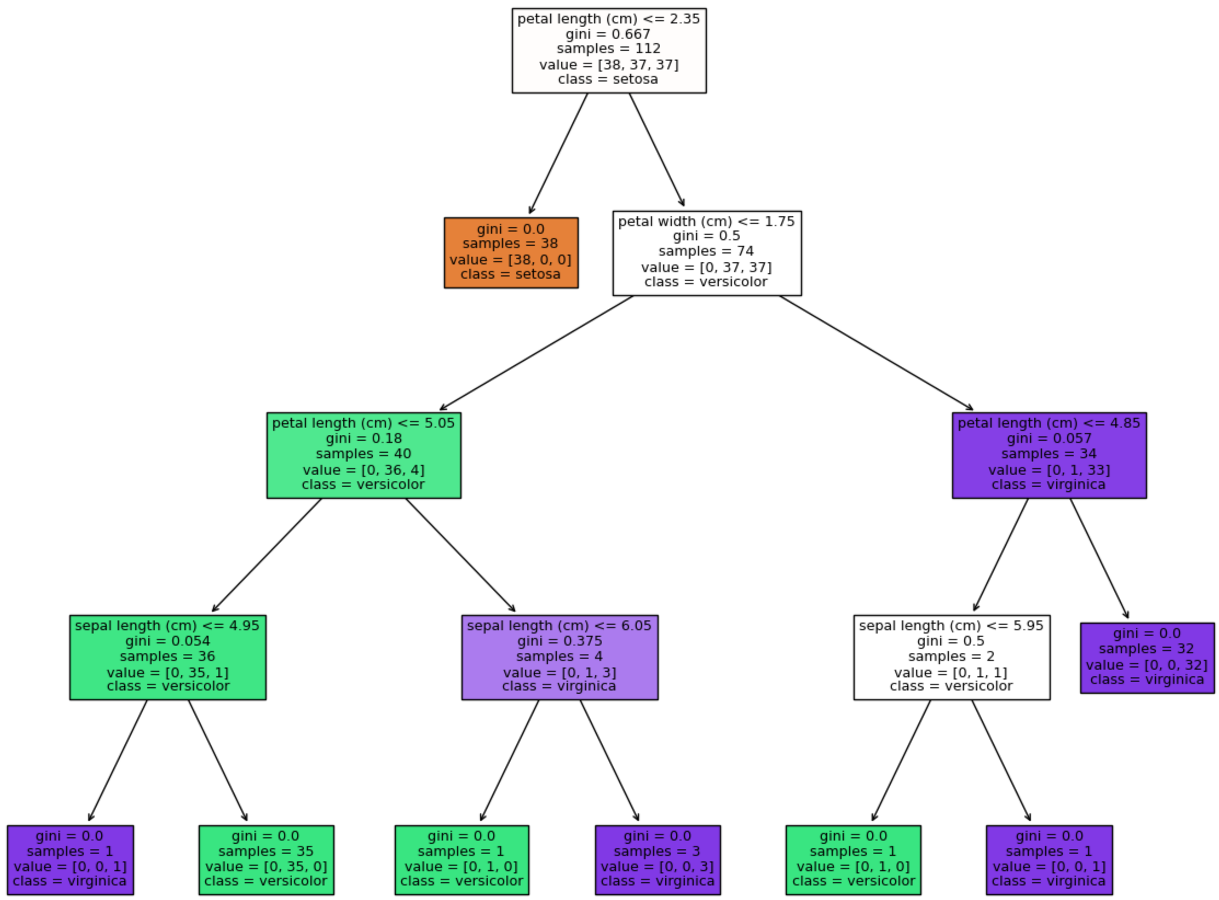
Out[25]:
```
array([0.91666667, 0.91666667, 1.        , 1.        , 1.        ,
       1.        , 0.90909091, 0.90909091, 0.81818182, 1.        ])
```

In [26]:
```python
accuracy_decision_tree_1.mean()
```

Out[26]:
```
0.9469696969696969
```

Plotting the Decision Tree Classifier:

In [27]:
```
plt.figure(figsize=(15,12))
plot_tree(dt_classifier_1, filled=True, feature_names=iris.feature_names[:],
          class_names=iris.target_names)
plt.show()
```



Post Pruning the Decision Tree:

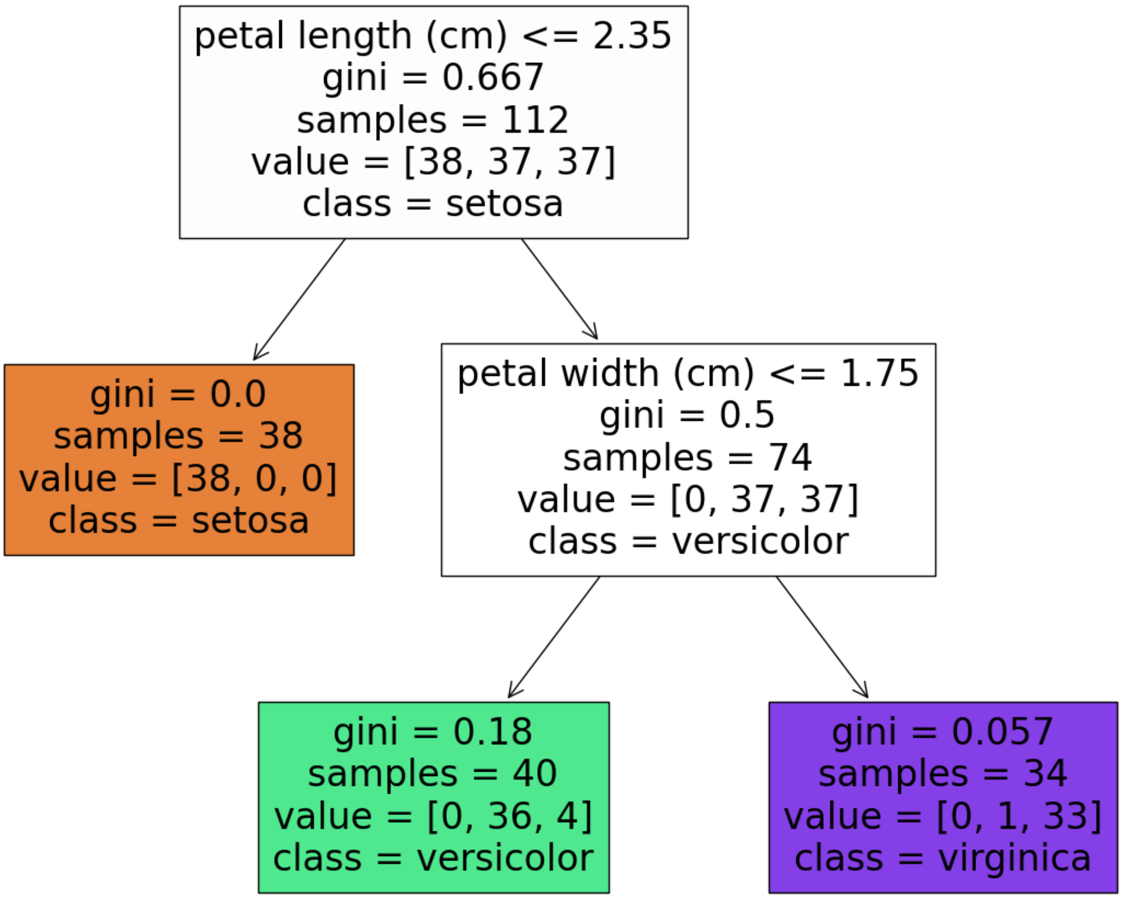
In [28]:
```
# From the Decison Tree we plotted above we see that only 'petal length (cm)' and 'petal width (cm)' are the most important
# features so we move them into a new array.
X2 = iris.data[:, 2:]
```

In [29]:
```
X2
```

```
       [1.3, 0.2],
       [1.5, 0.2],
       [1.3, 0.3],
       [1.3, 0.3],
       [1.3, 0.2],
       [1.6, 0.6],
       [1.9, 0.4],
       [1.4, 0.3],
       [1.6, 0.2],
       [1.4, 0.2],
       [1.5, 0.2],
       [1.4, 0.2],
       [4.7, 1.4],
       [4.5, 1.5],
       [4.9, 1.5],
       [4. , 1.3],
       [4.6, 1.5],
       [4.5, 1.3],
       [4.7, 1.6],
       [3.3, 1. ],
```

In [38]:
```
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(X2, y, test_size=0.25, stratify=y, random_state=1234)
```

In [39]:
```
# Plotting the Decision Tree Classifier:

dt_classifier_2 = DecisionTreeClassifier(max_depth=2)
dt_classifier_2.fit(X_train_2, y_train_2)

y_pred_train_2 = dt_classifier_2.predict(X_train_2)
y_pred_test_2 = dt_classifier_2.predict(X_test_2)

plt.figure(figsize=(15,12))

# filled=True keyword argument in plot_tree() is used to paint the leaf nodes which may be pure or impure.
plot_tree(dt_classifier_2, filled=True, feature_names=iris.feature_names[2:],
          class_names=iris.target_names)
plt.show()
```



In [40]:
```
print(f'Train Accuracy: {accuracy_score(y_train_2, y_pred_train_2)}')
print(f'Test Accuracy: {accuracy_score(y_test_2, y_pred_test_2)}')
```

```
Train Accuracy: 0.9553571428571429
Test Accuracy: 0.9736842105263158
```

In [41]:
```
print('For Train Data:')
print(classification_report(y_train_2, y_pred_train_2))
```

```
For Train Data:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        38
           1       0.90      0.97      0.94        37
           2       0.97      0.89      0.93        37

    accuracy                           0.96       112
   macro avg       0.96      0.95      0.95       112
weighted avg       0.96      0.96      0.96       112
```

In [42]:
```
print('For Test Data:')
print(classification_report(y_test_2, y_pred_test_2))
```

```
For Test Data:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.93      1.00      0.96        13
           2       1.00      0.92      0.96        13

    accuracy                           0.97        38
   macro avg       0.98      0.97      0.97        38
weighted avg       0.98      0.97      0.97        38
```

In [43]:
```
accuracy_decision_tree_2 = cross_val_score(dt_classifier_2, X_train_2, y_train_2, cv=10)
accuracy_decision_tree_2
```

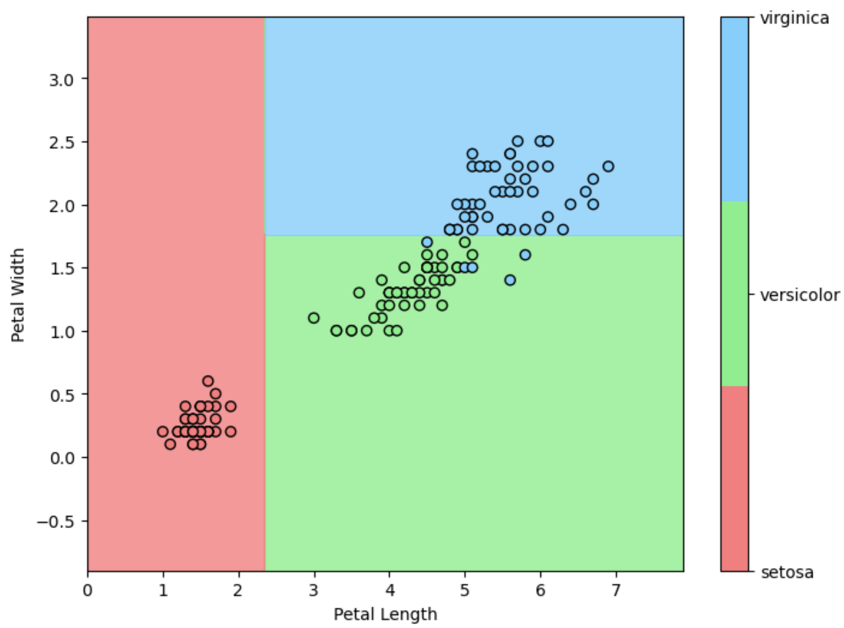
Out[43]:
```
array([0.91666667, 0.83333333, 1.        , 0.81818182, 1.        ,
       1.        , 0.90909091, 0.90909091, 0.81818182, 1.        ])
```

In [44]:
```
accuracy_decision_tree_2.mean()
```

Out[44]:
```
0.9204545454545455
```

Plotting the decision regions after selecting the best two attributes 'petal width' and 'petal length':

In [45]:
```
plt.figure(figsize=(8,6))
x_min, x_max = X2[:, 0].min() - 1, X2[:, 0].max()+1
y_min, y_max = X2[:, 1].min() - 1, X2[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                     np.arange(y_min, y_max, 0.01))

# np.c_[xx.ravel(), yy.ravel()] is same as np.r_['1,2,0', xx2.ravel(),yy2.ravel()] where we are concatenating the flattened
# arrays xx.ravel() and yy.ravel() along the axis 2.
z = dt_classifier_2.predict(np.c_[xx.ravel(), yy.ravel()])
z = z.reshape(xx.shape)

# Creating a ListedColormap object using custom colors.
colors = ['lightcoral', 'lightgreen', 'lightskyblue']
cmap = matplotlib.colors.ListedColormap(colors)

# plt.contourf() plots filled contours which has 3 levels because there are 3 classes in the output/target variable.
contour = plt.contourf(xx, yy, z, alpha=0.8, cmap=cmap)

# We then plot a scatter plot to represent the 2-D datapoints represented by the coordinates in X2[:,0], X2[:,1].
plt.scatter(X2[:,0], X2[:,1], c=y, edgecolor='k', cmap=cmap)

plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
cbar = plt.colorbar()

# cbar.set_ticks() sets the ticks/markers on the axes.
cbar.set_ticks([0, 1, 2])

# cbar.set_ticklabels() sets the tick labels for the ticks.
cbar.set_ticklabels(iris.target_names)

plt.show()
```



Creating a text report showing the rules of the Decision Tree:

In [51]:
```
explain = export_text(dt_classifier_2, feature_names=['petal_length', 'petal_width'])

# class 0 -> 'setosa'
# class 1 -> 'versicolor'
# class 2 -> 'virginica'
print(explain)
```

```
|--- petal_length <= 2.35
|   |--- class: 0
|--- petal_length >  2.35
|   |--- petal_width <= 1.75
|   |   |--- class: 1
|   |--- petal_width >  1.75
|   |   |--- class: 2
```

Conclusion:

The most important features for classifying the samples according to **'species'** are: **'petal width'** and **'petal length'**.

Max depth for the tree = 2 gives us a reasonably good model.

These information have been found using post pruning.

Before pruning the tree:

- Train Accuracy: **100%**
- Test Accuracy: **94.74%**
- Train Accuracy with 10 fold stratified cross validation: **94.7%**

After post pruning the tree:

- Train Accuracy: **95.53%**
- Test Accuracy: **97.37%**
- Train Accuracy with 10 fold stratified cross validation: **92.05%**