In [59]:
```
from sklearn.datasets import load_iris
iris = load_iris()
```
executed in 16ms, finished 12:39:38 2023-10-26

In [60]:
```
iris
```
executed in 16ms, finished 12:39:38 2023-10-26

Out[60]:
```
{'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],
        [5.1, 3.8, 1.5, 0.3],
```

In [61]:
```
#테스트 데이터의 비율을 0.25로 나누기
from sklearn.model_selection import train_test_split
train_input , test_input , train_target , test_target = train_test_split(iris.dat
```
executed in 15ms, finished 12:39:38 2023-10-26

In [62]:
```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
```
executed in 15ms, finished 12:39:38 2023-10-26

In [63]:
```
ss.fit(train_input)
```
executed in 16ms, finished 12:39:38 2023-10-26

Out[63]:
```
StandardScaler
StandardScaler()
```

In [64]:
```
train_scaled = ss.transform(train_input)
test_scaled = ss.transform(test_input)
```
executed in 15ms, finished 12:39:38 2023-10-26

4.1 KNN

In [65]:
```
from sklearn.neighbors import KNeighborsClassifier
kn = KNeighborsClassifier()
kn.fit(train_scaled , train_target)
print(kn.score(train_scaled , train_target))
print(kn.score(test_scaled , test_target))
```

executed in 16ms, finished 12:39:38 2023-10-26

```
0.9642857142857143
1.0
```

- 과소적합이 일어났다고 판단해서 , 이웃 수를 조정해보기

In [66]:
```
for i in [1,3,5,7,9,11,13,15,17,19]:
    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(train_scaled , train_target)
    print('이웃 수 : ' , i)
    print(knn.score(train_scaled , train_target))
    print(knn.score(test_scaled , test_target))
    print()
```

executed in 66ms, finished 12:39:38 2023-10-26

```
이웃 수 :  1
1.0
1.0

이웃 수 :  3
0.9464285714285714
1.0

이웃 수 :  5
0.9642857142857143
1.0

이웃 수 :  7
0.9642857142857143
1.0

이웃 수 :  9
0.9464285714285714
1.0

이웃 수 :  11
0.9553571428571429
1.0

이웃 수 :  13
0.9464285714285714
1.0

이웃 수 :  15
0.9375
1.0

이웃 수 :  17
0.9375
1.0

이웃 수 :  19
0.9375
1.0
```

In [67]:
```
from sklearn.metrics import accuracy_score , precision_score , recall_score , r
```

executed in 15ms, finished 12:39:38 2023-10-26

In [68]:
```
pred = kn.predict(test_scaled)
```

executed in 7ms, finished 12:39:38 2023-10-26

In [69]:
```
confusion_matrix(test_target , pred)
```
executed in 13ms, finished 12:39:38 2023-10-26

Out[69]:
```
array([[15,  0,  0],
       [ 0, 11,  0],
       [ 0,  0, 12]], dtype=int64)
```

- 잘못 분류한게 하나도 없다...

4.2 Logistic Regression

In [70]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(train_scaled , train_target)
print(lr.score(train_scaled , train_target))
print(lr.score(test_scaled , test_target))
```
executed in 15ms, finished 12:39:38 2023-10-26

```
0.9642857142857143
1.0
```

In [71]:
```
pred_proba = lr.predict_proba(test_scaled)
```
executed in 17ms, finished 12:39:38 2023-10-26

In [72]:
```
pred = lr.predict(test_scaled)
```
executed in 15ms, finished 12:39:38 2023-10-26

In [73]:
```
confusion_matrix(test_target , pred)
```
executed in 15ms, finished 12:39:38 2023-10-26

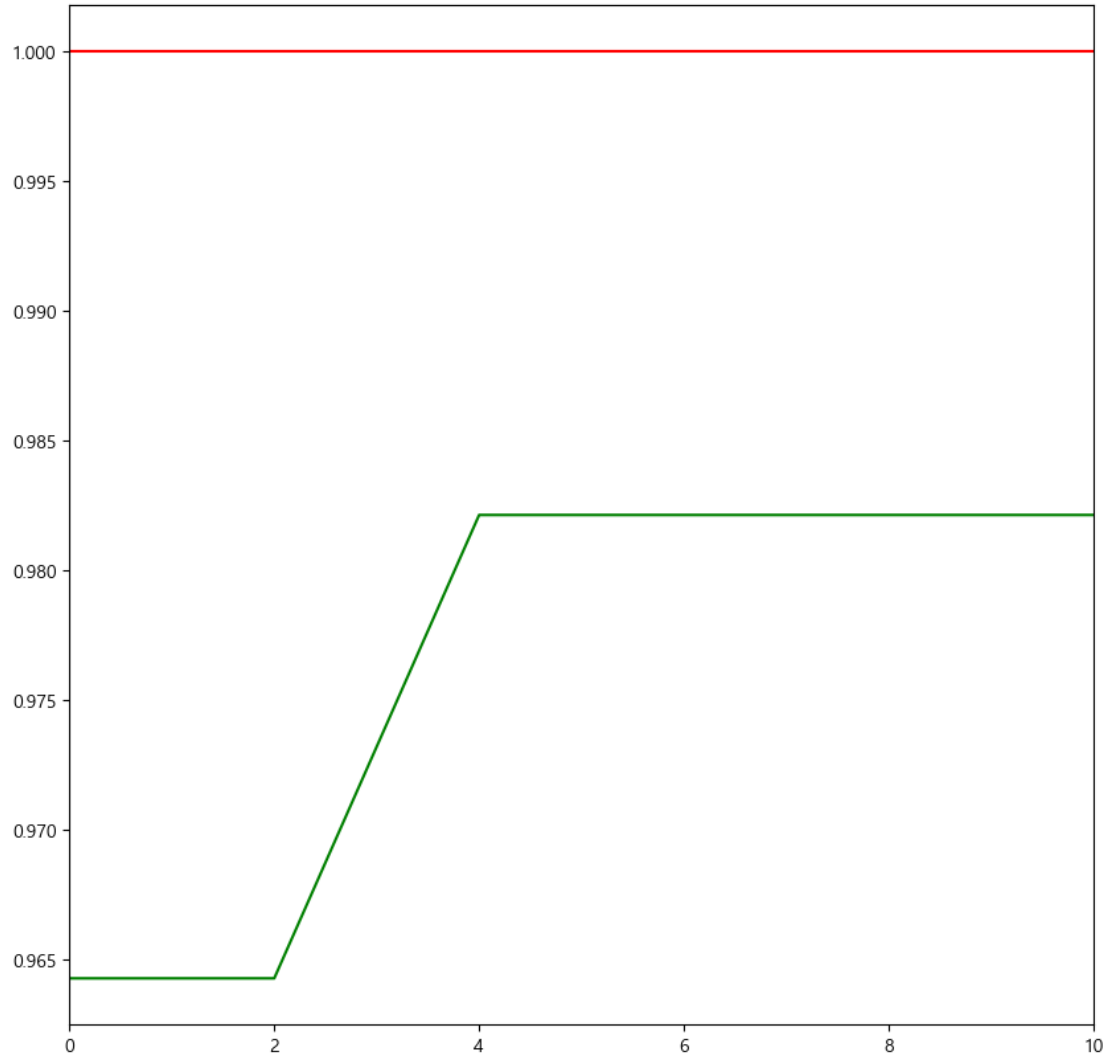
Out[73]:
```
array([[15,  0,  0],
       [ 0, 11,  0],
       [ 0,  0, 12]], dtype=int64)
```

- KNN 때와 똑같다.

In [74]:
```
train = []
test = []
for i in range(1,100):
    lrr = LogisticRegression(C = i)
    lrr.fit(train_scaled , train_target)
    train.append(lrr.score(train_scaled , train_target))
    test.append(lrr.score(test_scaled , test_target))
```
executed in 593ms, finished 12:39:39 2023-10-26

In [75]:
```
plt.figure(figsize = (10,10))
plt.plot(train , color = 'g' , label = 'train')
plt.plot(test , color = 'r' , label = 'test')
plt.xlim(0,10)
plt.show()
```

executed in 121ms, finished 12:39:39 2023-10-26



In [76]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(C=5)
lr.fit(train_scaled , train_target)
print(lr.score(train_scaled , train_target))
print(lr.score(test_scaled , test_target))
```

executed in 12ms, finished 12:39:39 2023-10-26

```
0.9821428571428571
1.0
```

In [77]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(C=5)
lr.fit(train_scaled , train_target)
print(lr.score(train_scaled , train_target))
print(lr.score(test_scaled , test_target))
```

executed in 17ms, finished 12:39:39 2023-10-26

```
0.9821428571428571
1.0
```

- 규제를 C = 5인 단계로 줄 때부터 , train 데이터의 점수가 0.982로 KNN보다 높게 나왔다.
- KNN의 최대값 : 0.9642
- LR 의 최대값 : 0.9821

4.3 의사결정나무

- 의사결정나무는 scale 안해도 됨.

In [78]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(train_input , train_target)
print(dt.score(train_input , train_target))
print(dt.score(test_input , test_target))
```

executed in 17ms, finished 12:39:39 2023-10-26

```
1.0
1.0
```

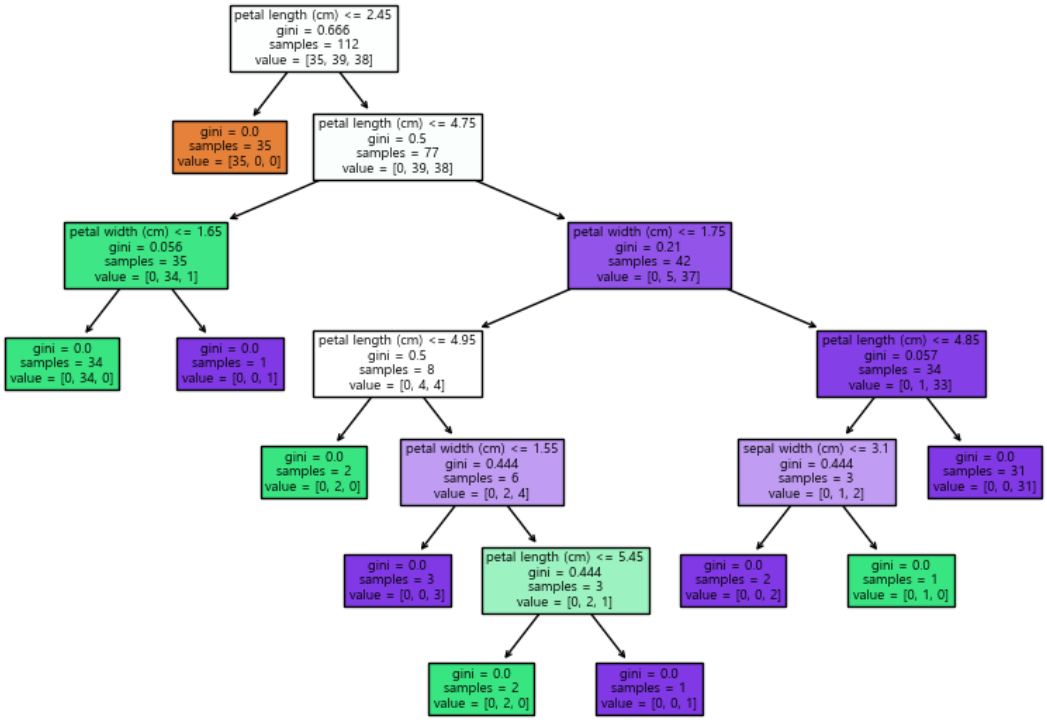
In [79]:
```
iris.feature_names
```

executed in 15ms, finished 12:39:39 2023-10-26

Out[79]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [80]:
```
plt.figure(figsize = (10,7))
plot_tree(dt , filled = True , feature_names = ['sepal length (cm)','sepal width
plt.show()
```

executed in 473ms, finished 12:39:39 2023-10-26



In [81]:
```
from sklearn.tree import export_graphviz
```

executed in 4ms, finished 12:39:39 2023-10-26

In [82]:
```
export_graphviz(dt , out_file = 'tree.dot' , class_names = iris.target_names , fe
```

executed in 14ms, finished 12:39:39 2023-10-26

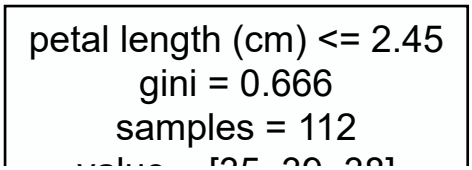
In [83]:
```
import graphviz
```

executed in 32ms, finished 12:39:39 2023-10-26

In [84]:
```
with open('tree.dot') as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

executed in 319ms, finished 12:39:40 2023-10-26

Out[84]:



- 예측이 너무 잘되는데 , 데이터의 분포를 확인해보기

In [85]:
```
df = pd.DataFrame(iris.data , columns = iris.feature_names)
```
executed in 13ms, finished 12:39:40 2023-10-26

In [86]:
```
df['species'] = iris.target
```
executed in 5ms, finished 12:39:40 2023-10-26

In [87]:
```
df
```
executed in 15ms, finished 12:39:40 2023-10-26

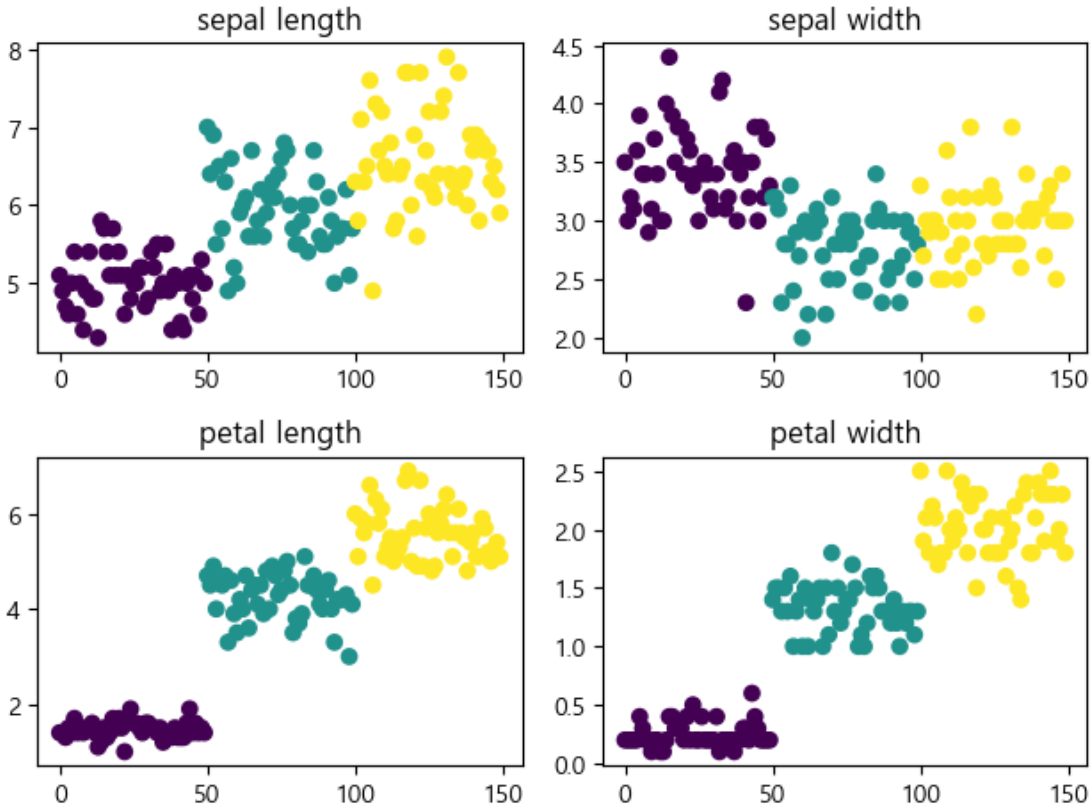
Out[87]:

	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
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

In [88]:
```
plt.subplot(2,2,1)
plt.scatter(x = np.arange(150) , y = df['sepal length (cm)'] , c = df['species']
plt.title('sepal length')
plt.subplot(2,2,2)
plt.scatter(x = np.arange(150) , y = df['sepal width (cm)'] , c = df['species'])
plt.title('sepal width')
plt.subplot(2,2,3)
plt.scatter(x = np.arange(150) , y = df['petal length (cm)'] , c = df['species']
plt.title('petal length')
plt.subplot(2,2,4)
plt.scatter(x = np.arange(150) , y = df['petal width (cm)'] , c = df['species'])
plt.title('petal width')
plt.tight_layout()
plt.show()
```

executed in 444ms, finished 12:39:40 2023-10-26



- 4개의 변수별로 분류가 잘 되어 있어서 잘되는것같다.