

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
In [2]: from sklearn.datasets import load_iris
```

```
In [3]: iris=load_iris()
```

```
In [6]: iris.data
```

```
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],
 [5.1, 3.8, 1.5, 0.3],
 [5.4, 3.4, 1.7, 0.2],
 [5.1, 3.7, 1.5, 0.4],
 [4.6, 3.6, 1. , 0.2],
 [5.1, 3.3, 1.7, 0.5],
 [4.8, 3.4, 1.9, 0.2],
 [5. , 3. , 1.6, 0.2],
 [5. , 3.4, 1.6, 0.4],
 [5.2, 3.5, 1.5, 0.2],
 [5.2, 3.4, 1.4, 0.2],
 [4.7, 3.2, 1.6, 0.2],
 [4.8, 3.1, 1.6, 0.2],
 [5.4, 3.4, 1.5, 0.4],
 [5.2, 4.1, 1.5, 0.1],
 [5.5, 4.2, 1.4, 0.2],
 [4.9, 3.1, 1.5, 0.2],
 [5. , 3.2, 1.2, 0.2],
 [5.5, 3.5, 1.3, 0.2],
 [4.9, 3.6, 1.4, 0.1],
 [4.4, 3. , 1.3, 0.2],
 [5.1, 3.4, 1.5, 0.2],
 [5. , 3.5, 1.3, 0.3],
 [4.5, 2.3, 1.3, 0.3],
 [4.4, 3.2, 1.3, 0.2],
 [5. , 3.5, 1.6, 0.6],
 [5.1, 3.8, 1.9, 0.4],
 [4.8, 3. , 1.4, 0.3],
```

[5.1, 3.8, 1.6, 0.2],  
[4.6, 3.2, 1.4, 0.2],  
[5.3, 3.7, 1.5, 0.2],  
[5. , 3.3, 1.4, 0.2],  
[7. , 3.2, 4.7, 1.4],  
[6.4, 3.2, 4.5, 1.5],  
[6.9, 3.1, 4.9, 1.5],  
[5.5, 2.3, 4. , 1.3],  
[6.5, 2.8, 4.6, 1.5],  
[5.7, 2.8, 4.5, 1.3],  
[6.3, 3.3, 4.7, 1.6],  
[4.9, 2.4, 3.3, 1. ],  
[6.6, 2.9, 4.6, 1.3],  
[5.2, 2.7, 3.9, 1.4],  
[5. , 2. , 3.5, 1. ],  
[5.9, 3. , 4.2, 1.5],  
[6. , 2.2, 4. , 1. ],  
[6.1, 2.9, 4.7, 1.4],  
[5.6, 2.9, 3.6, 1.3],  
[6.7, 3.1, 4.4, 1.4],  
[5.6, 3. , 4.5, 1.5],  
[5.8, 2.7, 4.1, 1. ],  
[6.2, 2.2, 4.5, 1.5],  
[5.6, 2.5, 3.9, 1.1],  
[5.9, 3.2, 4.8, 1.8],  
[6.1, 2.8, 4. , 1.3],  
[6.3, 2.5, 4.9, 1.5],  
[6.1, 2.8, 4.7, 1.2],  
[6.4, 2.9, 4.3, 1.3],  
[6.6, 3. , 4.4, 1.4],  
[6.8, 2.8, 4.8, 1.4],  
[6.7, 3. , 5. , 1.7],  
[6. , 2.9, 4.5, 1.5],  
[5.7, 2.6, 3.5, 1. ],  
[5.5, 2.4, 3.8, 1.1],  
[5.5, 2.4, 3.7, 1. ],  
[5.8, 2.7, 3.9, 1.2],  
[6. , 2.7, 5.1, 1.6],  
[5.4, 3. , 4.5, 1.5],  
[6. , 3.4, 4.5, 1.6],  
[6.7, 3.1, 4.7, 1.5],  
[6.3, 2.3, 4.4, 1.3],  
[5.6, 3. , 4.1, 1.3],  
[5.5, 2.5, 4. , 1.3],  
[5.5, 2.6, 4.4, 1.2],  
[6.1, 3. , 4.6, 1.4],  
[5.8, 2.6, 4. , 1.2],  
[5. , 2.3, 3.3, 1. ],  
[5.6, 2.7, 4.2, 1.3],  
[5.7, 3. , 4.2, 1.2],  
[5.7, 2.9, 4.2, 1.3],  
[6.2, 2.9, 4.3, 1.3],  
[5.1, 2.5, 3. , 1.1],  
[5.7, 2.8, 4.1, 1.3],  
[6.3, 3.3, 6. , 2.5],  
[5.8, 2.7, 5.1, 1.9],  
[7.1, 3. , 5.9, 2.1],  
[6.3, 2.9, 5.6, 1.8],  
[6.5, 3. , 5.8, 2.2],  
[7.6, 3. , 6.6, 2.1],  
[4.9, 2.5, 4.5, 1.7],  
[7.3, 2.9, 6.3, 1.8],  
[6.7, 2.5, 5.8, 1.8],  
[7.2, 3.6, 6.1, 2.5],  
[6.5, 3.2, 5.1, 2. ],  
[6.4, 2.7, 5.3, 1.9],

```
[6.8, 3. , 5.5, 2.1],
[5.7, 2.5, 5. , 2. ],
[5.8, 2.8, 5.1, 2.4],
[6.4, 3.2, 5.3, 2.3],
[6.5, 3. , 5.5, 1.8],
[7.7, 3.8, 6.7, 2.2],
[7.7, 2.6, 6.9, 2.3],
[6. , 2.2, 5. , 1.5],
[6.9, 3.2, 5.7, 2.3],
[5.6, 2.8, 4.9, 2. ],
[7.7, 2.8, 6.7, 2. ],
[6.3, 2.7, 4.9, 1.8],
[6.7, 3.3, 5.7, 2.1],
[7.2, 3.2, 6. , 1.8],
[6.2, 2.8, 4.8, 1.8],
[6.1, 3. , 4.9, 1.8],
[6.4, 2.8, 5.6, 2.1],
[7.2, 3. , 5.8, 1.6],
[7.4, 2.8, 6.1, 1.9],
[7.9, 3.8, 6.4, 2. ],
[6.4, 2.8, 5.6, 2.2],
[6.3, 2.8, 5.1, 1.5],
[6.1, 2.6, 5.6, 1.4],
[7.7, 3. , 6.1, 2.3],
[6.3, 3.4, 5.6, 2.4],
[6.4, 3.1, 5.5, 1.8],
[6. , 3. , 4.8, 1.8],
[6.9, 3.1, 5.4, 2.1],
[6.7, 3.1, 5.6, 2.4],
[6.9, 3.1, 5.1, 2.3],
[5.8, 2.7, 5.1, 1.9],
[6.8, 3.2, 5.9, 2.3],
[6.7, 3.3, 5.7, 2.5],
[6.7, 3. , 5.2, 2.3],
[6.3, 2.5, 5. , 1.9],
[6.5, 3. , 5.2, 2. ],
[6.2, 3.4, 5.4, 2.3],
[5.9, 3. , 5.1, 1.8]])
```

```
In [8]: iris.feature_names
```

```
Out[8]: ['sepal length (cm)',
'sepal width (cm)',
'petal length (cm)',
'petal width (cm)']
```

```
In [9]: iris.target
```

```
Out[9]: 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, 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, 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, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
In [10]: df=pd.DataFrame(iris.data,columns=iris.feature_names)
```

```
In [11]: df.head()
```

```
Out[11]:      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
```

---

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

```
In [12]: df['class']=iris.target
```

```
In [13]: df.head()
```

Out[13]:

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

```
In [14]: df.isna().sum()
```

```
Out[14]: sepal length (cm)    0
sepal width (cm)         0
petal length (cm)        0
petal width (cm)         0
class                    0
dtype: int64
```

```
In [15]: 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   class                  150 non-null   int32
dtypes: float64(4), int32(1)
memory usage: 5.4 KB
```

```
In [16]: df.describe()
```

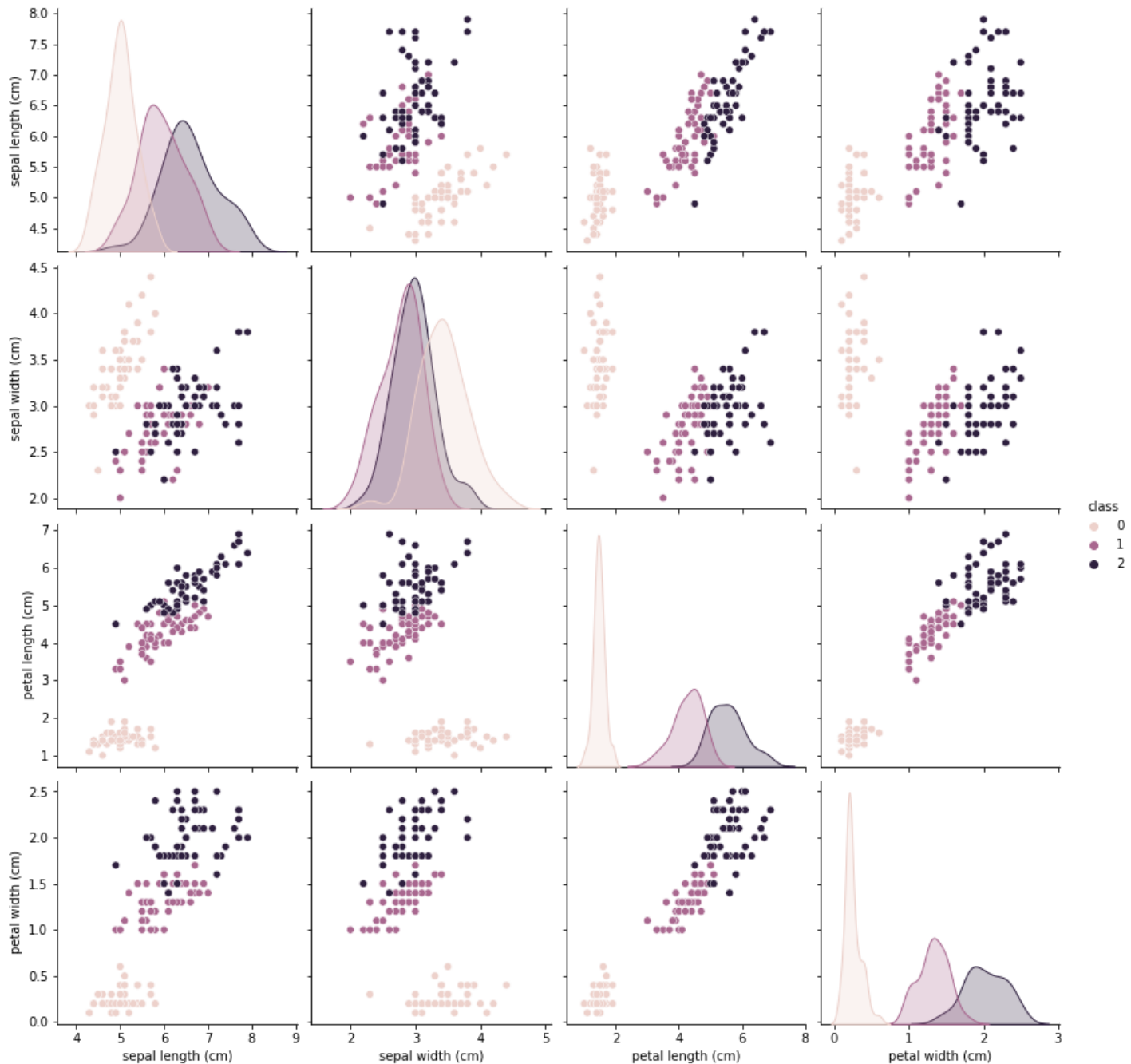
Out[16]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	class
<b>count</b>	150.000000	150.000000	150.000000	150.000000	150.000000
<b>mean</b>	5.843333	3.057333	3.758000	1.199333	1.000000
<b>std</b>	0.828066	0.435866	1.765298	0.762238	0.819232

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	class
<b>min</b>	4.300000	2.000000	1.000000	0.100000	0.000000
<b>25%</b>	5.100000	2.800000	1.600000	0.300000	0.000000
<b>50%</b>	5.800000	3.000000	4.350000	1.300000	1.000000
<b>75%</b>	6.400000	3.300000	5.100000	1.800000	2.000000
<b>max</b>	7.900000	4.400000	6.900000	2.500000	2.000000

```
In [48]: sns.pairplot(data=df, hue='class', size=3, diag_kind='kde')
```

```
Out[48]: <seaborn.axisgrid.PairGrid at 0x1dca435c310>
```



```
In [17]: colname=df.select_dtypes('float64').columns
```

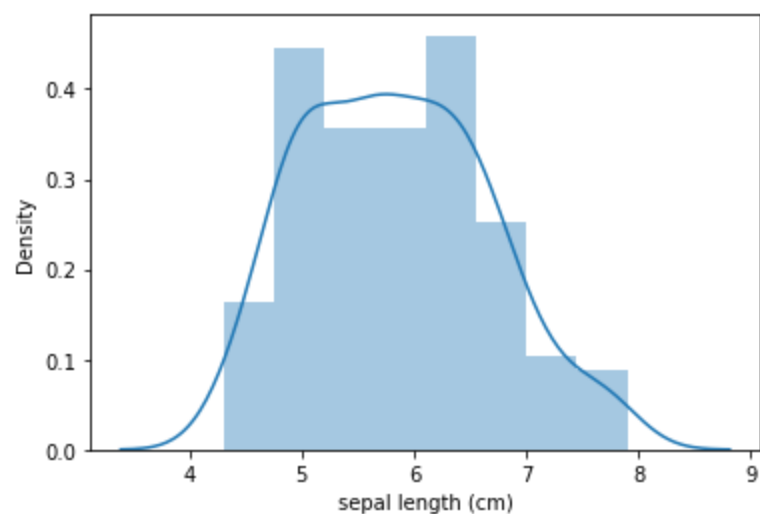
```
In [18]: colname
```

```
Out[18]: Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',  
        'petal width (cm)'],  
        dtype='object')
```

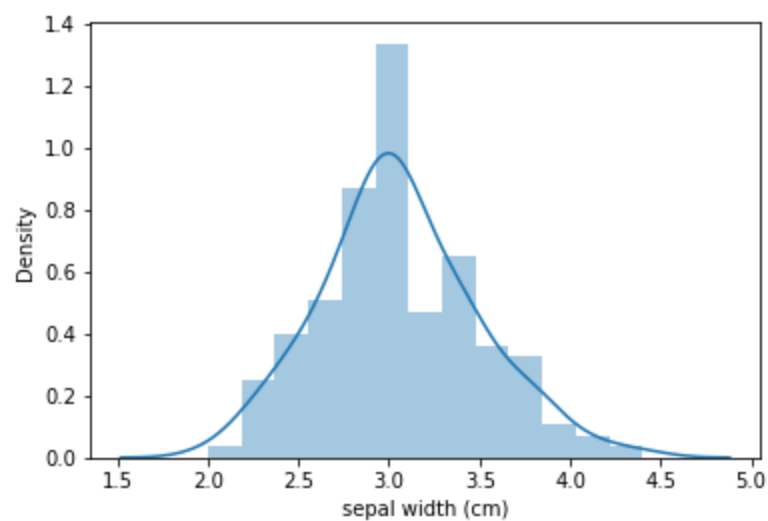
```
In [20]: from scipy.stats import skew
```

```
In [21]: for i in df[colname]:  
        print(i)  
        print(skew(df[i]))  
        sns.distplot(df[i])  
        plt.show()
```

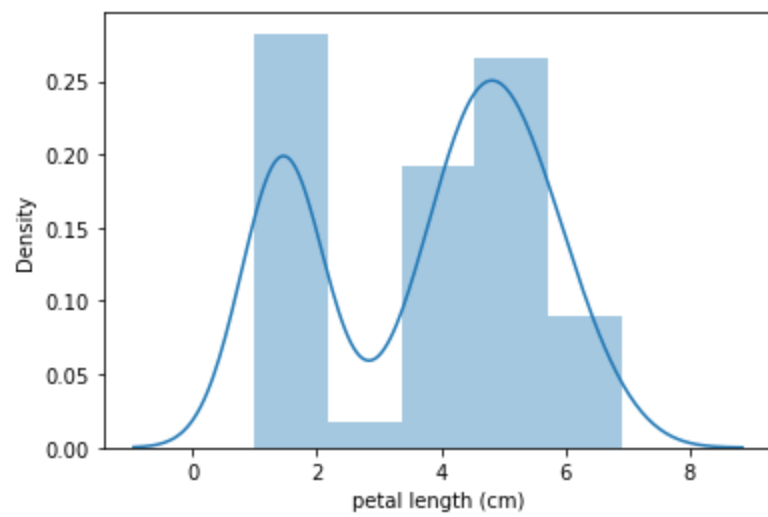
```
sepal length (cm)  
0.3117530585022963
```



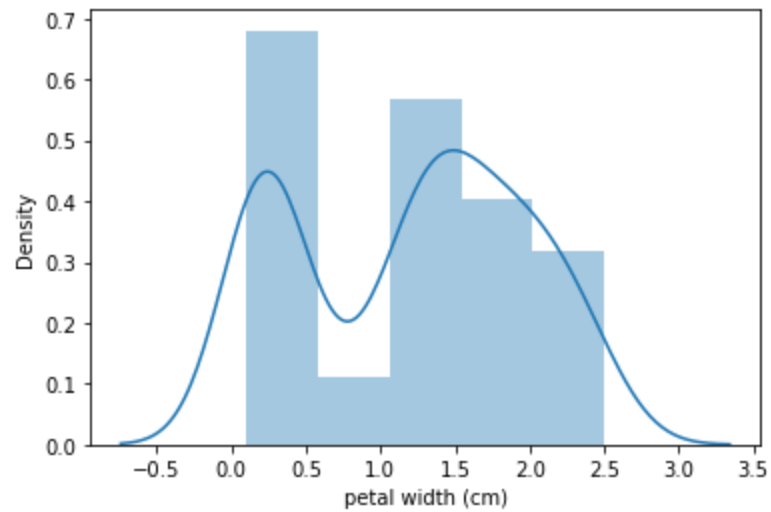
```
sepal width (cm)  
0.31576710633893473
```



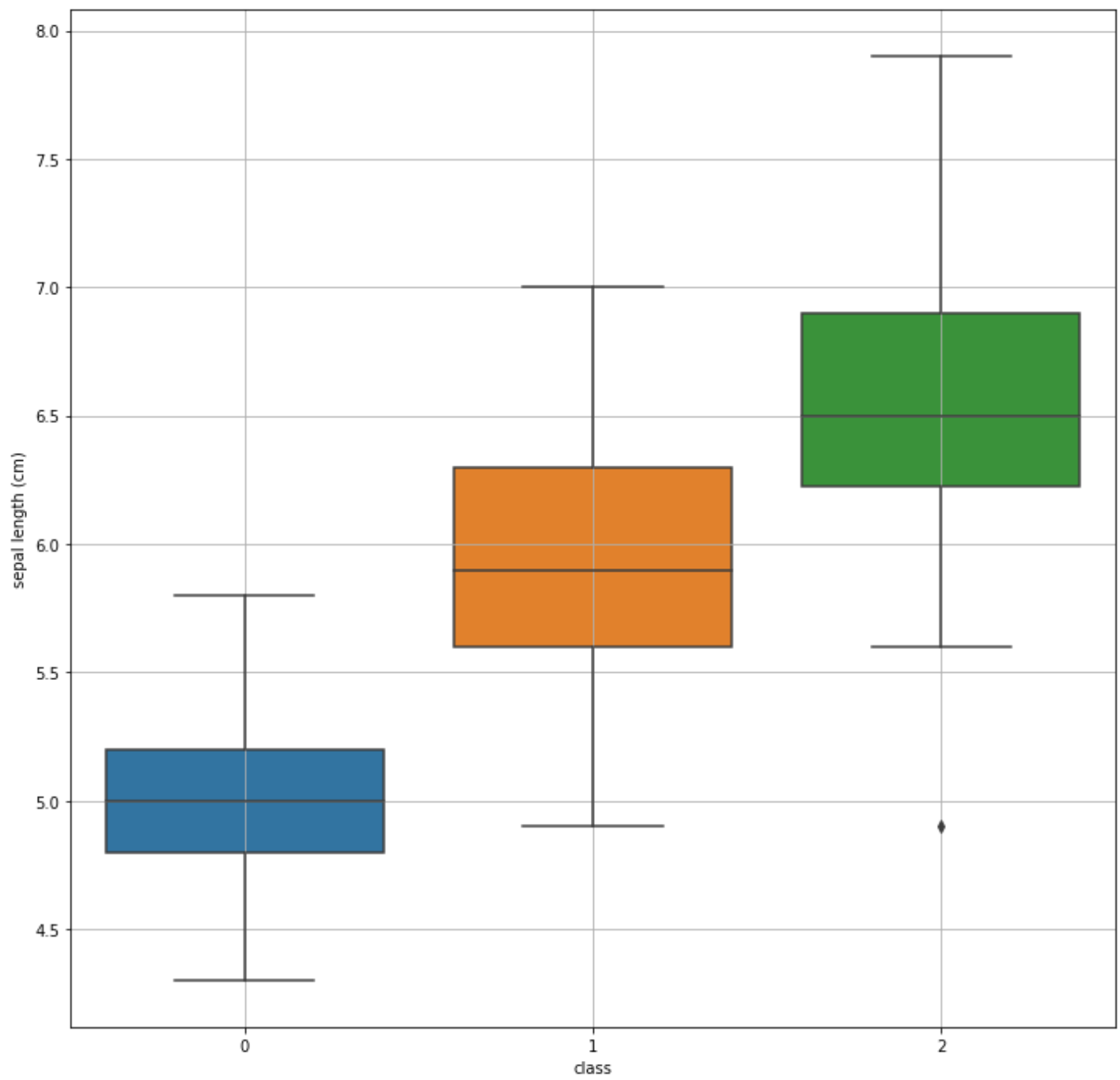
```
petal length (cm)  
-0.2721276664567214
```



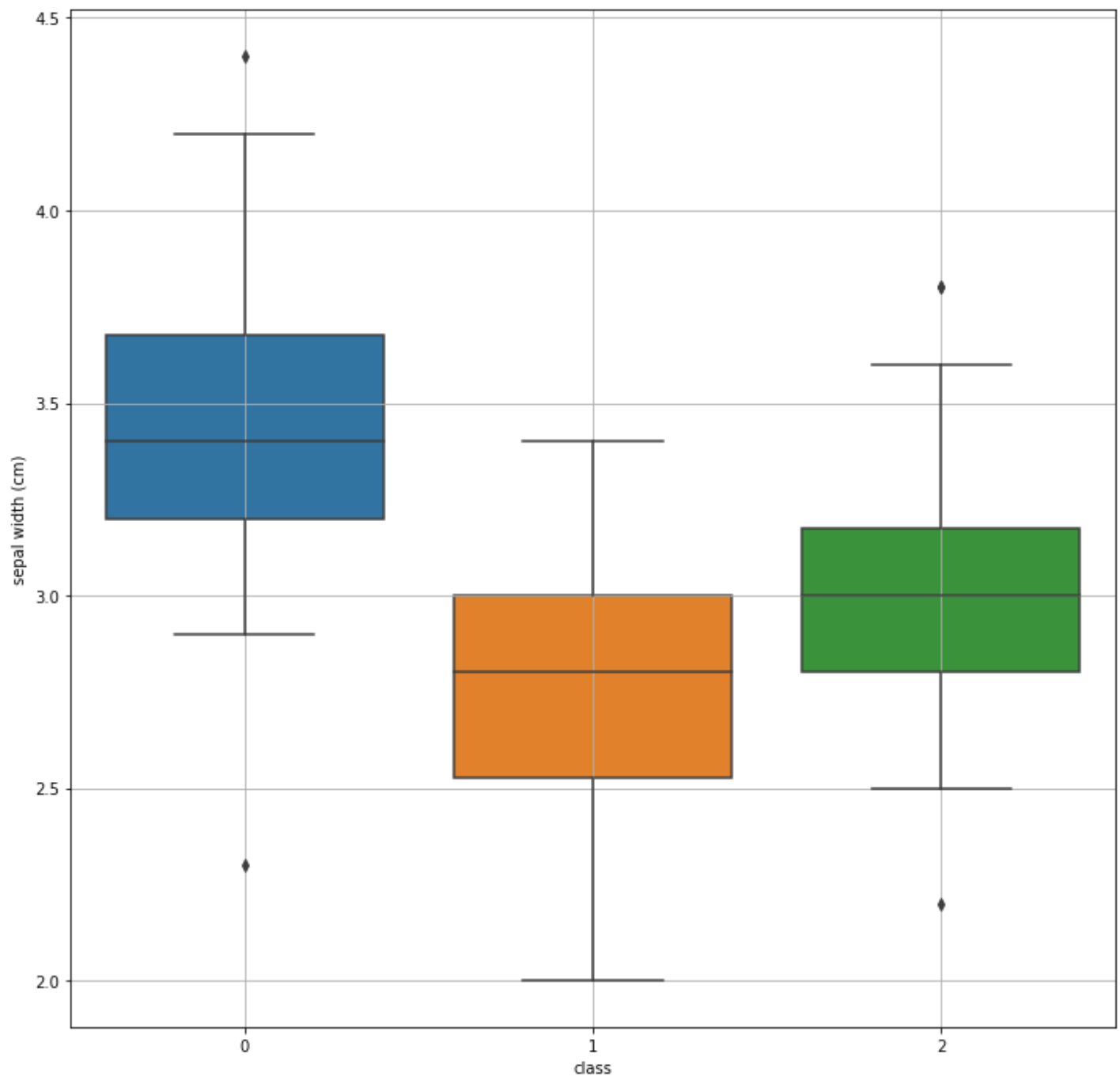
petal width (cm)  
-0.10193420656560036

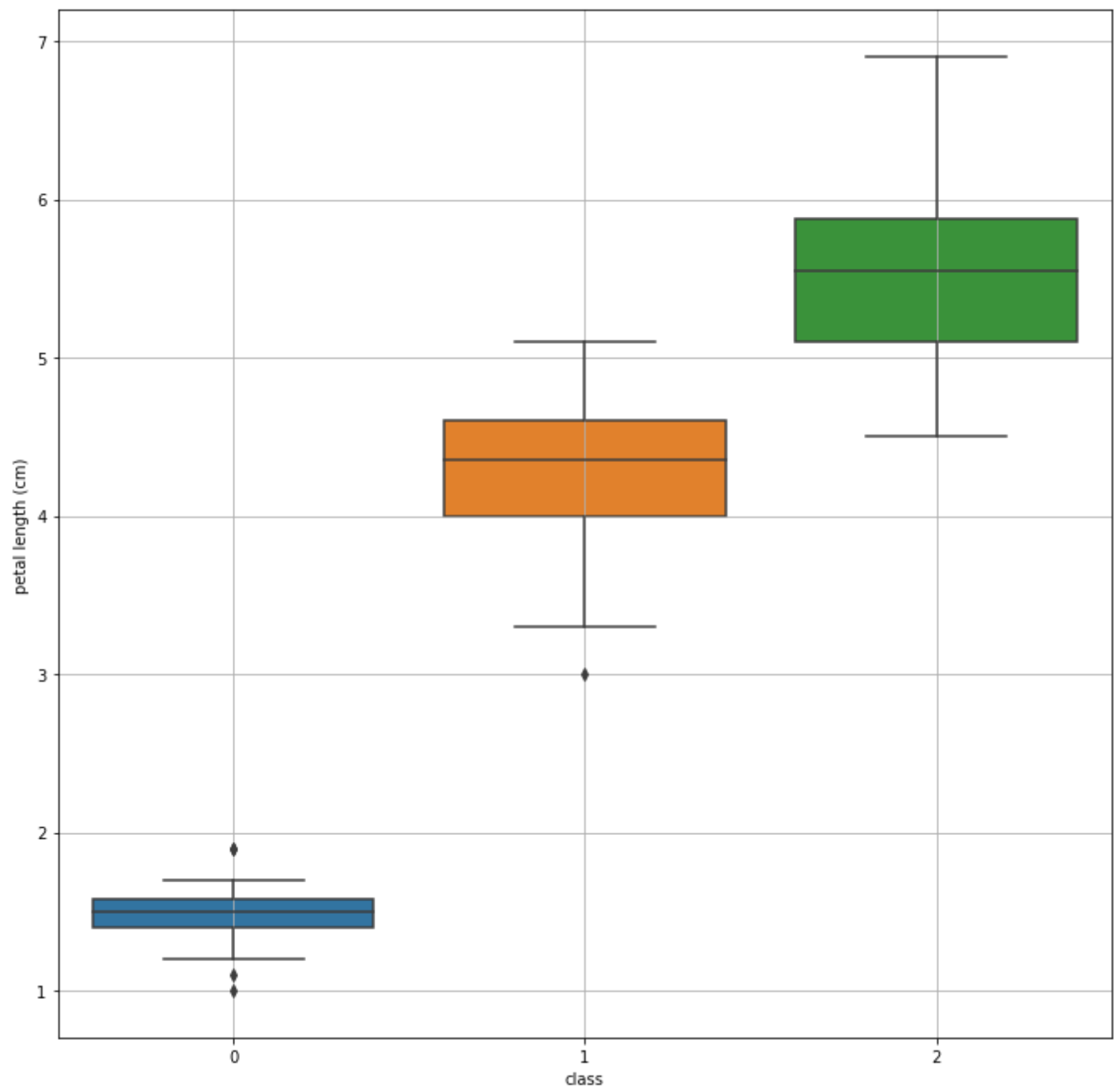


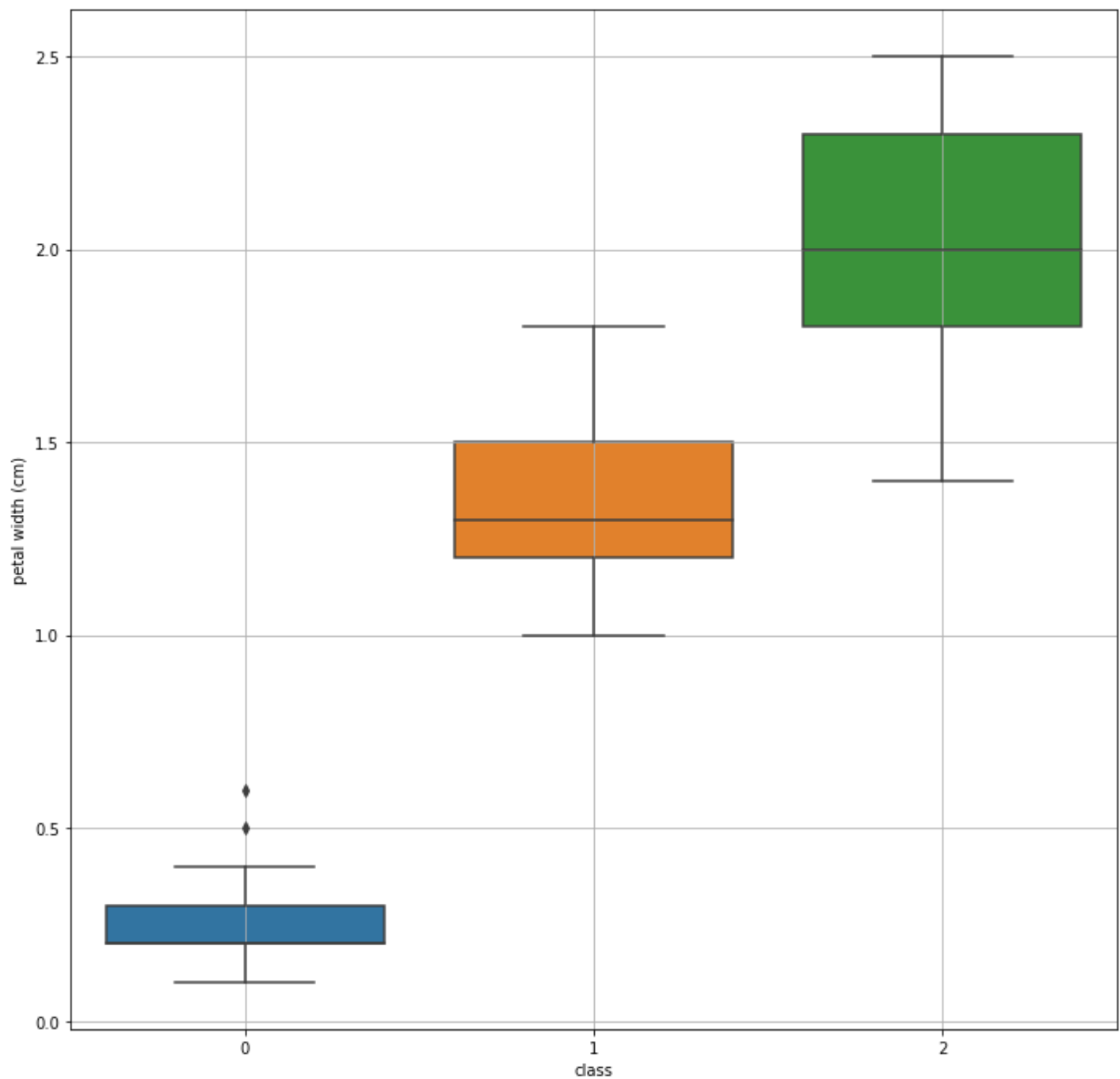
```
In [24]: for i in df[colname]:
plt.figure(figsize=(12,12))
sns.boxplot(x='class',y=df[i],data=df)
plt.grid()
plt.show()
```











```
In [25]: x=df.iloc[:, :-1]
         y=df.iloc[:, -1]
```

```
In [26]: from sklearn.model_selection import train_test_split
```

```
In [27]: xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=1)
```

```
In [28]: def mymodel(model):
         #model creation
         model.fit(xtrain,ytrain)
         ypred=model.predict(xtest)
         #checking bias and variance
         train=model.score(xtrain,ytrain)
         test=model.score(xtest,ytest)
         print(f'training acc :{train}\ntesting acc :{test}')
         #checking accuracy
```

```
print(classification_report(ytest,ypred))  
return model
```

```
In [38]: from sklearn.neighbors import KNeighborsClassifier  
from sklearn.linear_model import LogisticRegression  
from sklearn.svm import SVC  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.metrics import classification_report
```

```
In [41]: knn=mymodel(KNeighborsClassifier())
```

```
training acc :0.9523809523809523  
testing acc :0.9777777777777777  
      precision    recall  f1-score   support  
  
 0         1.00        1.00        1.00         14  
 1         0.95        1.00        0.97         18  
 2         1.00        0.92        0.96         13  
  
 accuracy          0.98          0.98          0.98         45  
 macro avg         0.98          0.97          0.98         45  
 weighted avg      0.98          0.98          0.98         45
```

```
In [42]: logreg=mymodel(LogisticRegression())
```

```
training acc :0.9809523809523809  
testing acc :0.9777777777777777  
      precision    recall  f1-score   support  
  
 0         1.00        1.00        1.00         14  
 1         1.00        0.94        0.97         18  
 2         0.93        1.00        0.96         13  
  
 accuracy          0.98          0.98          0.98         45  
 macro avg         0.98          0.98          0.98         45  
 weighted avg      0.98          0.98          0.98         45
```

```
In [43]: svm=mymodel(SVC())
```

```
training acc :0.9619047619047619  
testing acc :0.9777777777777777  
      precision    recall  f1-score   support  
  
 0         1.00        1.00        1.00         14  
 1         1.00        0.94        0.97         18  
 2         0.93        1.00        0.96         13  
  
 accuracy          0.98          0.98          0.98         45  
 macro avg         0.98          0.98          0.98         45  
 weighted avg      0.98          0.98          0.98         45
```

```
In [44]: dt=mymodel(DecisionTreeClassifier())
```

```
training acc :1.0  
testing acc :0.9555555555555556  
      precision    recall  f1-score   support  
  
 0         1.00        1.00        1.00         14
```

	1	0.94	0.94	0.94	18
	2	0.92	0.92	0.92	13
accuracy				0.96	45
macro avg		0.96	0.96	0.96	45
weighted avg		0.96	0.96	0.96	45

```
In [53]: parameters={
    'criterion': ['gini','entropy'],
    'max_depth': list(range(1,20)),
    'min_samples_leaf': list(range(1,20))
}
```

```
In [54]: from sklearn.model_selection import GridSearchCV
grid=GridSearchCV(DecisionTreeClassifier(),parameters,verbose=3)
grid.fit(xtrain,ytrain)
```

```
Fitting 5 folds for each of 722 candidates, totalling 3610 fits
[CV 1/5] END criterion=gini, max_depth=1, min_samples_leaf=1;, score=0.714 total time=
0.0s
[CV 2/5] END criterion=gini, max_depth=1, min_samples_leaf=1;, score=0.714 total time=
0.0s
[CV 3/5] END criterion=gini, max_depth=1, min_samples_leaf=1;, score=0.714 total time=
0.0s
[CV 4/5] END criterion=gini, max_depth=1, min_samples_leaf=1;, score=0.667 total time=
0.0s
[CV 5/5] END criterion=gini, max_depth=1, min_samples_leaf=1;, score=0.667 total time=
0.0s
[CV 1/5] END criterion=gini, max_depth=1, min_samples_leaf=2;, score=0.714 total time=
0.0s
[CV 2/5] END criterion=gini, max_depth=1, min_samples_leaf=2;, score=0.714 total time=
0.0s
[CV 3/5] END criterion=gini, max_depth=1, min_samples_leaf=2;, score=0.714 total time=
0.0s
[CV 4/5] END criterion=gini, max_depth=1, min_samples_leaf=2;, score=0.667 total time=
0.0s
[CV 5/5] END criterion=gini, max_depth=1, min_samples_leaf=2;, score=0.667 total time=
0.0s
[CV 1/5] END criterion=gini, max_depth=1, min_samples_leaf=3;, score=0.714 total time=
0.0s
[CV 2/5] END criterion=gini, max_depth=1, min_samples_leaf=3;, score=0.714 total time=
0.0s
[CV 3/5] END criterion=gini, max_depth=1, min_samples_leaf=3;, score=0.714 total time=
0.0s
[CV 4/5] END criterion=gini, max_depth=1, min_samples_leaf=3;, score=0.667 total time=
0.0s
[CV 5/5] END criterion=gini, max_depth=1, min_samples_leaf=3;, score=0.667 total time=
0.0s
[CV 1/5] END criterion=gini, max_depth=1, min_samples_leaf=4;, score=0.714 total time=
0.0s
[CV 2/5] END criterion=gini, max_depth=1, min_samples_leaf=4;, score=0.714 total time=
0.0s
[CV 3/5] END criterion=gini, max_depth=1, min_samples_leaf=4;, score=0.714 total time=
0.0s
[CV 4/5] END criterion=gini, max_depth=1, min_samples_leaf=4;, score=0.667 total time=
0.0s
[CV 5/5] END criterion=gini, max_depth=1, min_samples_leaf=4;, score=0.667 total time=
0.0s
[CV 1/5] END criterion=gini, max_depth=1, min_samples_leaf=5;, score=0.714 total time=
0.0s
[CV 2/5] END criterion=gini, max_depth=1, min_samples_leaf=5;, score=0.714 total time=
0.0s
```

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

```

[CV 2/5] END criterion=entropy, max_depth=19, min_samples_leaf=15;; score=0.810 total time
= 0.0s
[CV 3/5] END criterion=entropy, max_depth=19, min_samples_leaf=15;; score=1.000 total time
= 0.0s
[CV 4/5] END criterion=entropy, max_depth=19, min_samples_leaf=15;; score=0.952 total time
= 0.0s
[CV 5/5] END criterion=entropy, max_depth=19, min_samples_leaf=15;; score=0.857 total time
= 0.0s
[CV 1/5] END criterion=entropy, max_depth=19, min_samples_leaf=16;; score=1.000 total time
= 0.0s
[CV 2/5] END criterion=entropy, max_depth=19, min_samples_leaf=16;; score=0.810 total time
= 0.0s
[CV 3/5] END criterion=entropy, max_depth=19, min_samples_leaf=16;; score=1.000 total time
= 0.0s
[CV 4/5] END criterion=entropy, max_depth=19, min_samples_leaf=16;; score=0.952 total time
= 0.0s
[CV 5/5] END criterion=entropy, max_depth=19, min_samples_leaf=16;; score=0.857 total time
= 0.0s
[CV 1/5] END criterion=entropy, max_depth=19, min_samples_leaf=17;; score=1.000 total time
= 0.0s
[CV 2/5] END criterion=entropy, max_depth=19, min_samples_leaf=17;; score=0.810 total time
= 0.0s
[CV 3/5] END criterion=entropy, max_depth=19, min_samples_leaf=17;; score=1.000 total time
= 0.0s
[CV 4/5] END criterion=entropy, max_depth=19, min_samples_leaf=17;; score=0.952 total time
= 0.0s
[CV 5/5] END criterion=entropy, max_depth=19, min_samples_leaf=17;; score=0.857 total time
= 0.0s
[CV 1/5] END criterion=entropy, max_depth=19, min_samples_leaf=18;; score=1.000 total time
= 0.0s
[CV 2/5] END criterion=entropy, max_depth=19, min_samples_leaf=18;; score=0.810 total time
= 0.0s
[CV 3/5] END criterion=entropy, max_depth=19, min_samples_leaf=18;; score=1.000 total time
= 0.0s
[CV 4/5] END criterion=entropy, max_depth=19, min_samples_leaf=18;; score=0.952 total time
= 0.0s
[CV 5/5] END criterion=entropy, max_depth=19, min_samples_leaf=18;; score=0.857 total time
= 0.0s
[CV 1/5] END criterion=entropy, max_depth=19, min_samples_leaf=19;; score=1.000 total time
= 0.0s
[CV 2/5] END criterion=entropy, max_depth=19, min_samples_leaf=19;; score=0.810 total time
= 0.0s
[CV 3/5] END criterion=entropy, max_depth=19, min_samples_leaf=19;; score=1.000 total time
= 0.0s
[CV 4/5] END criterion=entropy, max_depth=19, min_samples_leaf=19;; score=0.952 total time
= 0.0s
[CV 5/5] END criterion=entropy, max_depth=19, min_samples_leaf=19;; score=0.857 total time
= 0.0s

```

```

Out[54]: GridSearchCV(estimator=DecisionTreeClassifier(),
                      param_grid={'criterion': ['gini', 'entropy'],
                                   'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                                  13, 14, 15, 16, 17, 18, 19],
                                   'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
                                                         12, 13, 14, 15, 16, 17, 18, 19]},
                      verbose=3)

```

```

In [55]: grid.best_score_

```

```

Out[55]: 0.9523809523809523

```

```

In [56]: grid.best_estimator_

```

```

Out[56]: DecisionTreeClassifier(max_depth=3)

```

```
In [57]: dt=mymodel(grid.best_estimator_)
```

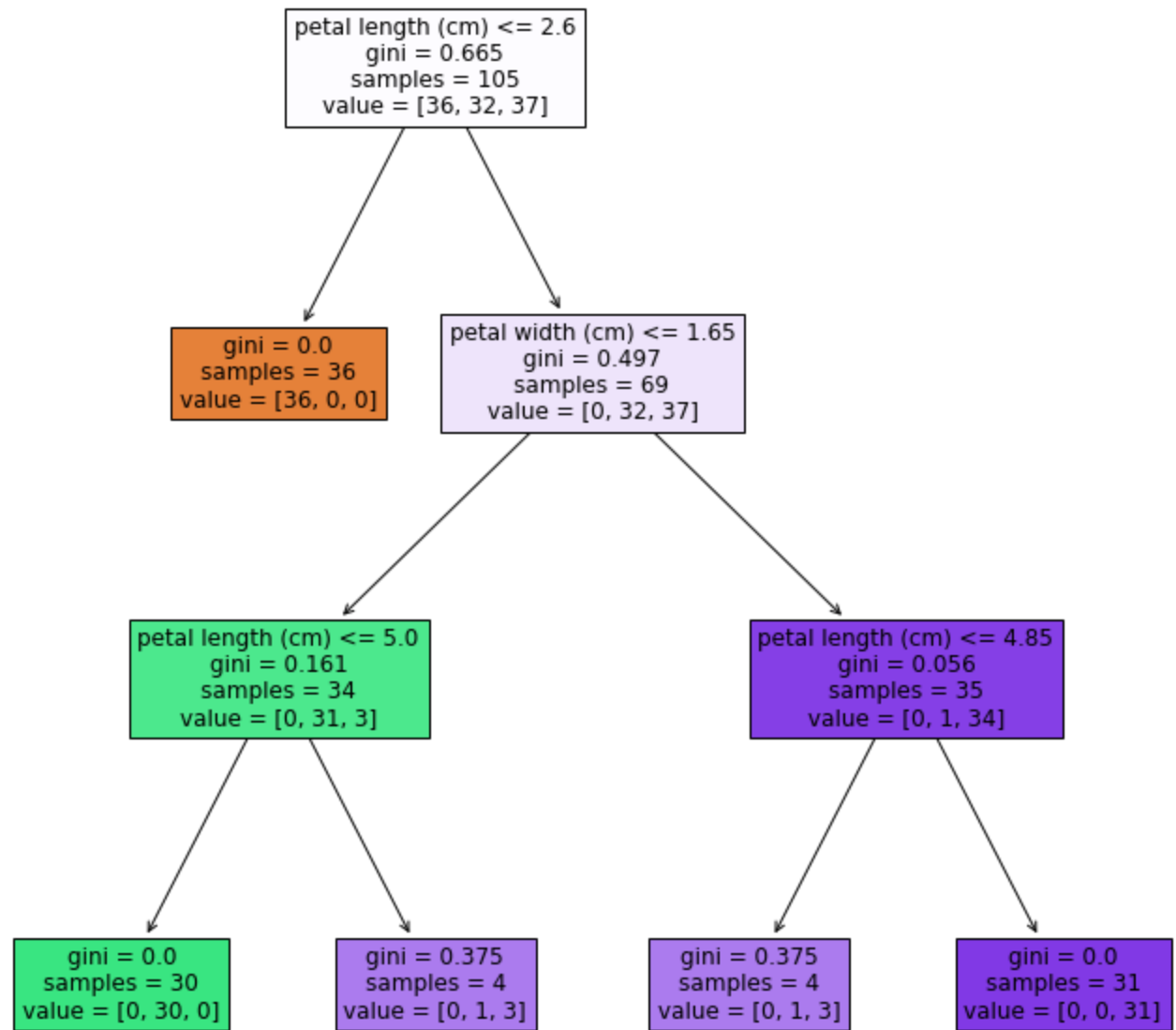
```
training acc :0.9809523809523809
```

```
testing acc :0.9555555555555556
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	14
1	0.94	0.94	0.94	18
2	0.92	0.92	0.92	13
accuracy			0.96	45
macro avg	0.96	0.96	0.96	45
weighted avg	0.96	0.96	0.96	45

```
In [58]: from sklearn import tree
```

```
In [59]: plt.figure(figsize=(12,12))
tree.plot_tree(dt,feature_names=iris.feature_names,filled=True)
plt.show()
```



In [ ]: