Train the Iris data and Test the Model

- Iris is perhaps the best known database to be found in the pattern recognition literature.

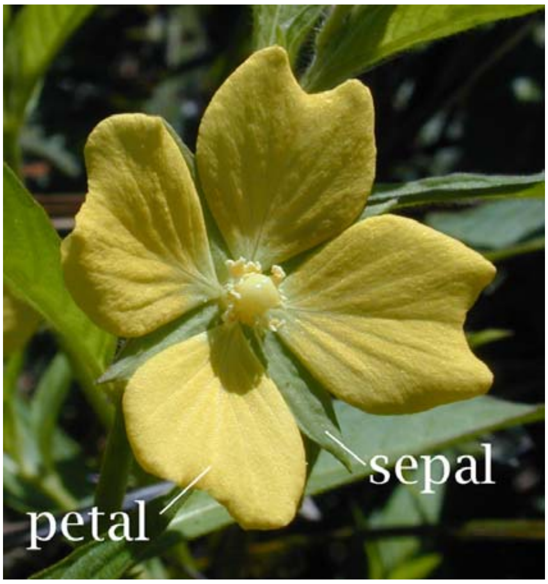


Figure 1: Flower showing petal and sepal [1].

- The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant.
- One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.
  - Number of Instances: 150 (50 in each of three classes)
  - Number of Attributes/features: 4 numeric, predictive attributes and the class
  - Attribute Information:
    1. sepal length in cm
    2. sepal width in cm
    3. petal length in cm
    4. petal width in cm
    5. class:
       - Iris Setosa
       - Iris Versicolour
       - Iris Virginica

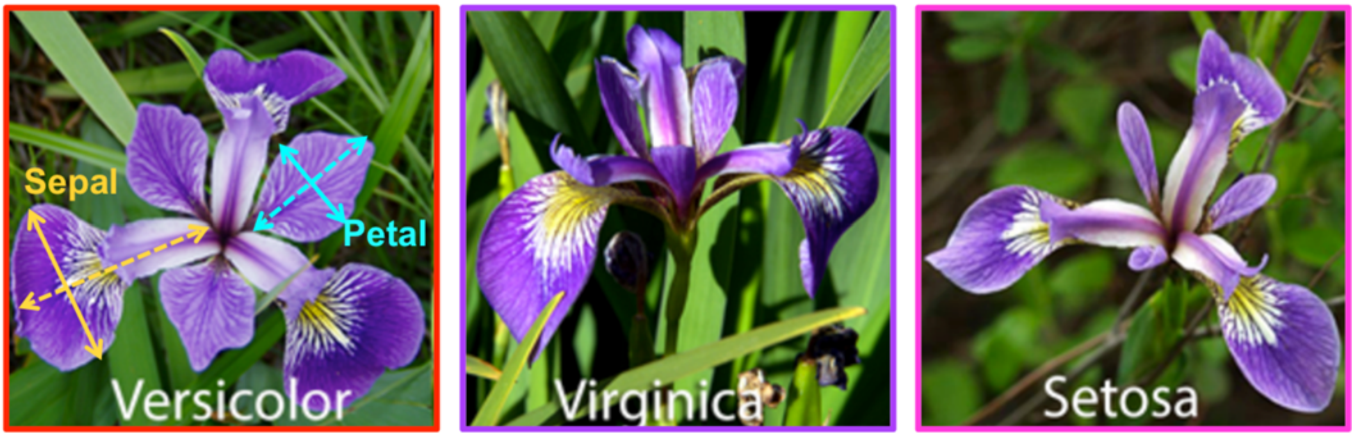


Figure 2: Three categories of Iris flowers [2].

In [1]:
```
# Load pandas library. We want to use its DataFrame which supports tabular form.
import pandas as pd
```

In [2]:
```
# Load the dataset from the current directory into a DataFrame
iris=pd.read_csv("iris.arff")
```

In [3]:

```
# See the content of the iris dataset
iris
```

Out[3]:

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [4]:

```
# Want to see a few rows (actually 4, but 5 including the header)
iris.head()
```

Out[4]:

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

In [5]:

```
# Want to see a few last rows
iris.tail()
```

Out[5]:

	sepal_length	sepal_width	petal_length	petal_width	class
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

In [6]:

```
# Information about the dataset
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal_length    150 non-null float64
sepal_width     150 non-null float64
petal_length    150 non-null float64
petal_width     150 non-null float64
class           150 non-null object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [7]:

```
# Some Statistical info. of the dataset
iris.describe()
```

Out[7]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [8]:

```
# Want to see the column
iris.columns
```

Out[8]:

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class '], dtype='o
bject')
```

In [9]:

```
# I am intertested to see the unique values in the class column because I want to replace
the text with the numeric values
iris['class '].unique()
```

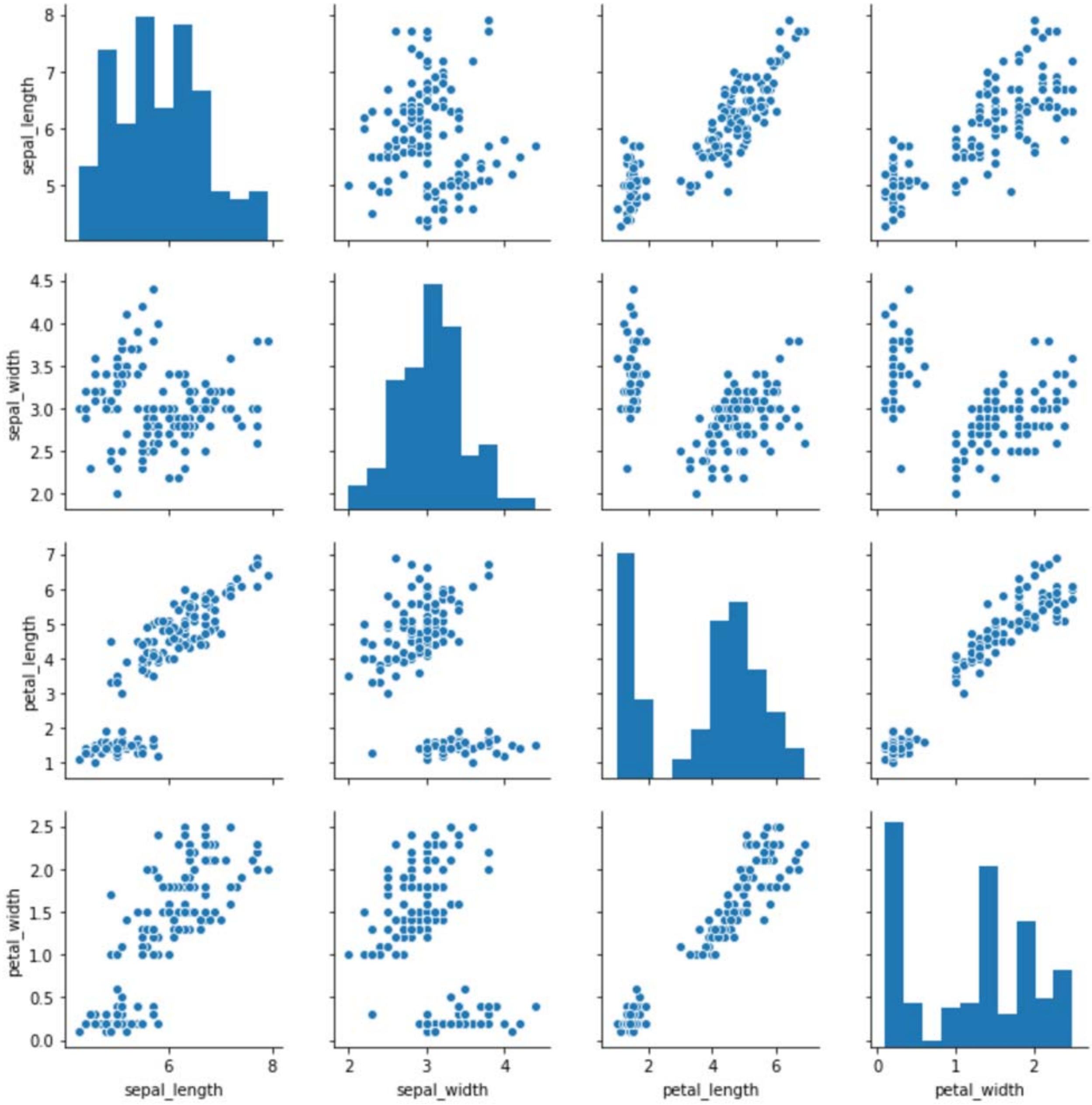
Out[9]:

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

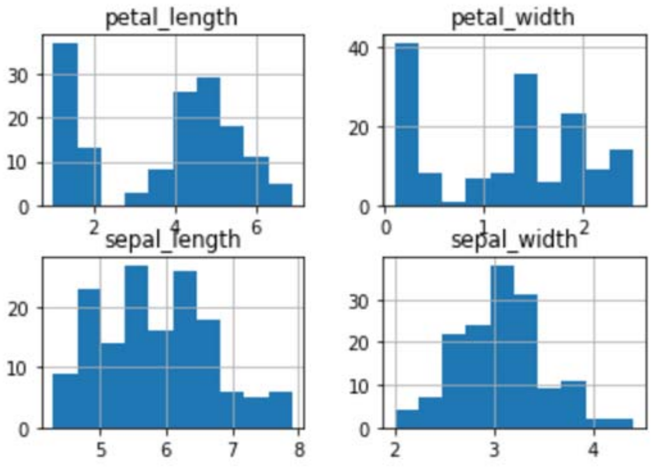
In [10]:

```
# Visualizing pairwise relationships
import seaborn as sns # for more on seaborn, see https://seaborn.pydata.org/
```

```
In [11]: sns.pairplot(iris); #';' avoid outputing the internal location info here
```



```
In [12]: # Want to see the histogram of the numerical columns using malplotlib
         import matplotlib.pyplot as plt
         %matplotlib inline
         iris.hist()
         plt.show()
```



In [13]:

```
# I want to replace 'Iris-setosa' with 0, 'Iris-versicolor' with 1, 'Iris-virginica' with 2
iris.replace("Iris-setosa",0)
```

Out[13]:

	sepal_length	sepal_width	petal_length	petal_width	class
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	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [14]:

```
# But the above table is a view - and the replacement will not be a permanent change [we need to use option: inplace=True]
iris
```

Out[14]:

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [15]:
```
# We also do not want to change the original dataset, so we make a copy

iriscp=iris.copy()
iriscp
```

Out[15]:

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [16]:
```
iriscp.replace(to_replace="Iris-setosa",value=0,inplace=True)
#iriscp.replace("Iris-setosa",0,inplace=True) # This will work as well
```

In [17]:
```
iriscp
```

Out[17]:

	sepal_length	sepal_width	petal_length	petal_width	class
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	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [18]:
```
# Instead of replace them one-by-one I want to replace them all at once
# So I make a dictionary (dict) first
myreplacementlist= {"Iris-setosa":0, "Iris-versicolor":1,"Iris-virginica":2}
```

In [19]:
```
myreplacementlist
# Note: I want the replacement to work only for column 'class '
```

Out[19]:
```
{'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}
```

In [20]:
```
# Testing the dict
myreplacementlist['Iris-versicolor']
```

Out[20]:
```
1
```

In [21]:
```
iriscp.replace({'class ': myreplacementlist}, inplace=True)
```

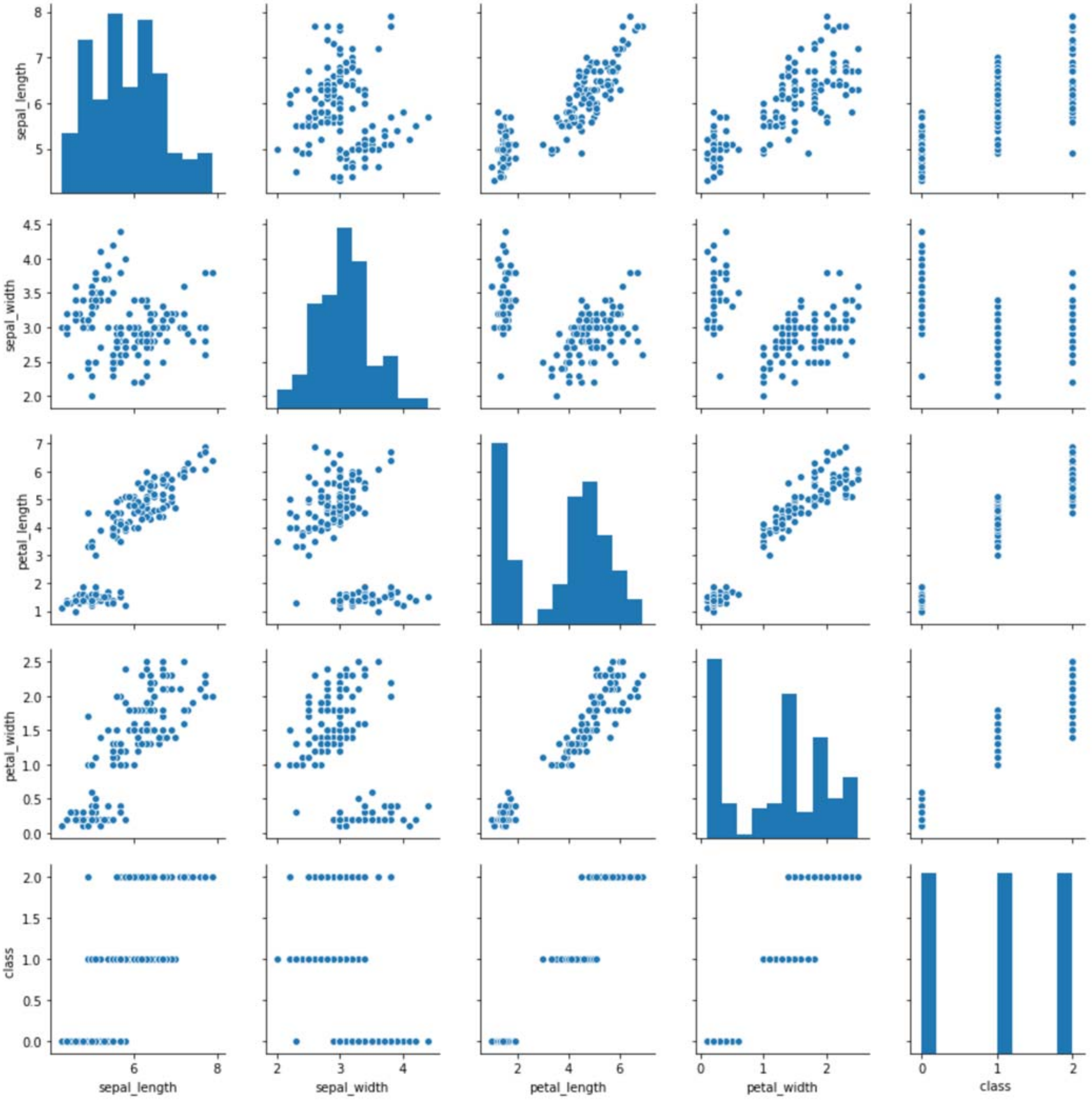
In [22]:
```
iriscp
```

Out[22]:

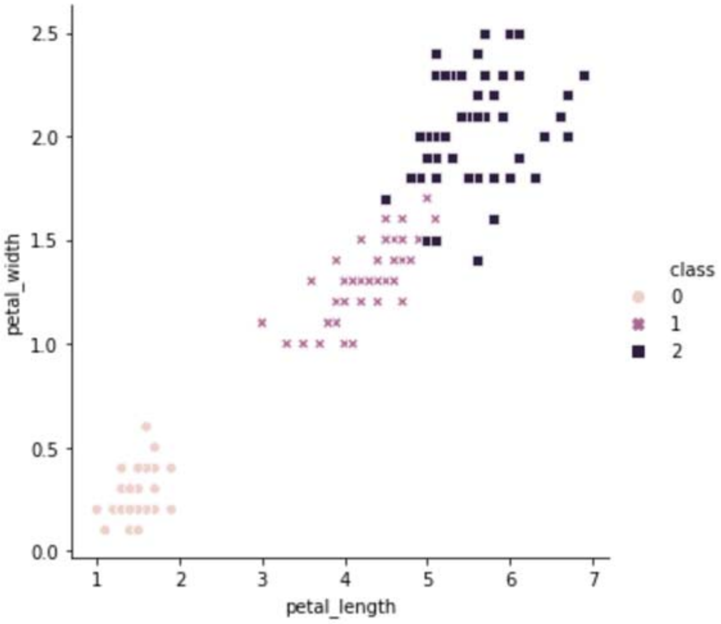
	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

In [23]:
```
# Now, all columns are numerical column - I want to run the pairplot again
sns.pairplot(iriscp);
```



In [24]:
```
# Want to examine how features help separate classes
sns.relplot(x='petal_length',y='petal_width',data=iriscp, hue='class ', style='class ');
```



In [25]:
```
# I want to save this table into a file
iriscp.to_csv('myiriscp.csv')
```

In [26]:

```
# read the file to check whether it is saved or not
!cat 'myiriscp.csv'
```

```
,sepal_length,sepal_width,petal_length,petal_width,class
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
5,5.4,3.9,1.7,0.4,0
6,4.6,3.4,1.4,0.3,0
7,5.0,3.4,1.5,0.2,0
8,4.4,2.9,1.4,0.2,0
9,4.9,3.1,1.5,0.1,0
10,5.4,3.7,1.5,0.2,0
11,4.8,3.4,1.6,0.2,0
12,4.8,3.0,1.4,0.1,0
13,4.3,3.0,1.1,0.1,0
14,5.8,4.0,1.2,0.2,0
15,5.7,4.4,1.5,0.4,0
16,5.4,3.9,1.3,0.4,0
17,5.1,3.5,1.4,0.3,0
18,5.7,3.8,1.7,0.3,0
19,5.1,3.8,1.5,0.3,0
20,5.4,3.4,1.7,0.2,0
21,5.1,3.7,1.5,0.4,0
22,4.6,3.6,1.0,0.2,0
23,5.1,3.3,1.7,0.5,0
24,4.8,3.4,1.9,0.2,0
25,5.0,3.0,1.6,0.2,0
26,5.0,3.4,1.6,0.4,0
27,5.2,3.5,1.5,0.2,0
28,5.2,3.4,1.4,0.2,0
29,4.7,3.2,1.6,0.2,0
30,4.8,3.1,1.6,0.2,0
31,5.4,3.4,1.5,0.4,0
32,5.2,4.1,1.5,0.1,0
33,5.5,4.2,1.4,0.2,0
34,4.9,3.1,1.5,0.1,0
35,5.0,3.2,1.2,0.2,0
36,5.5,3.5,1.3,0.2,0
37,4.9,3.1,1.5,0.1,0
38,4.4,3.0,1.3,0.2,0
39,5.1,3.4,1.5,0.2,0
40,5.0,3.5,1.3,0.3,0
41,4.5,2.3,1.3,0.3,0
42,4.4,3.2,1.3,0.2,0
43,5.0,3.5,1.6,0.6,0
44,5.1,3.8,1.9,0.4,0
45,4.8,3.0,1.4,0.3,0
46,5.1,3.8,1.6,0.2,0
47,4.6,3.2,1.4,0.2,0
48,5.3,3.7,1.5,0.2,0
49,5.0,3.3,1.4,0.2,0
50,7.0,3.2,4.7,1.4,1
51,6.4,3.2,4.5,1.5,1
52,6.9,3.1,4.9,1.5,1
53,5.5,2.3,4.0,1.3,1
54,6.5,2.8,4.6,1.5,1
55,5.7,2.8,4.5,1.3,1
56,6.3,3.3,4.7,1.6,1
57,4.9,2.4,3.3,1.0,1
58,6.6,2.9,4.6,1.3,1
59,5.2,2.7,3.9,1.4,1
60,5.0,2.0,3.5,1.0,1
61,5.9,3.0,4.2,1.5,1
62,6.0,2.2,4.0,1.0,1
63,6.1,2.9,4.7,1.4,1
64,5.6,2.9,3.6,1.3,1
65,6.7,3.1,4.4,1.4,1
66,5.6,3.0,4.5,1.5,1
67,5.8,2.7,4.1,1.0,1
68,6.2,2.2,4.5,1.5,1
69,5.6,2.5,3.9,1.1,1
70,5.9,3.2,4.8,1.8,1
71,6.1,2.8,4.0,1.3,1
```

In [27]:

```
# I can also use window's type command
!type myiriscp.csv
```

```
,sepal_length,sepal_width,petal_length,petal_width,class
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
5,5.4,3.9,1.7,0.4,0
6,4.6,3.4,1.4,0.3,0
7,5.0,3.4,1.5,0.2,0
8,4.4,2.9,1.4,0.2,0
9,4.9,3.1,1.5,0.1,0
10,5.4,3.7,1.5,0.2,0
11,4.8,3.4,1.6,0.2,0
12,4.8,3.0,1.4,0.1,0
13,4.3,3.0,1.1,0.1,0
14,5.8,4.0,1.2,0.2,0
15,5.7,4.4,1.5,0.4,0
16,5.4,3.9,1.3,0.4,0
17,5.1,3.5,1.4,0.3,0
18,5.7,3.8,1.7,0.3,0
19,5.1,3.8,1.5,0.3,0
20,5.4,3.4,1.7,0.2,0
21,5.1,3.7,1.5,0.4,0
22,4.6,3.6,1.0,0.2,0
23,5.1,3.3,1.7,0.5,0
24,4.8,3.4,1.9,0.2,0
25,5.0,3.0,1.6,0.2,0
26,5.0,3.4,1.6,0.4,0
27,5.2,3.5,1.5,0.2,0
28,5.2,3.4,1.4,0.2,0
29,4.7,3.2,1.6,0.2,0
30,4.8,3.1,1.6,0.2,0
31,5.4,3.4,1.5,0.4,0
32,5.2,4.1,1.5,0.1,0
33,5.5,4.2,1.4,0.2,0
34,4.9,3.1,1.5,0.1,0
35,5.0,3.2,1.2,0.2,0
36,5.5,3.5,1.3,0.2,0
37,4.9,3.1,1.5,0.1,0
38,4.4,3.0,1.3,0.2,0
39,5.1,3.4,1.5,0.2,0
40,5.0,3.5,1.3,0.3,0
41,4.5,2.3,1.3,0.3,0
42,4.4,3.2,1.3,0.2,0
43,5.0,3.5,1.6,0.6,0
44,5.1,3.8,1.9,0.4,0
45,4.8,3.0,1.4,0.3,0
46,5.1,3.8,1.6,0.2,0
47,4.6,3.2,1.4,0.2,0
48,5.3,3.7,1.5,0.2,0
49,5.0,3.3,1.4,0.2,0
50,7.0,3.2,4.7,1.4,1
51,6.4,3.2,4.5,1.5,1
52,6.9,3.1,4.9,1.5,1
53,5.5,2.3,4.0,1.3,1
54,6.5,2.8,4.6,1.5,1
55,5.7,2.8,4.5,1.3,1
56,6.3,3.3,4.7,1.6,1
57,4.9,2.4,3.3,1.0,1
58,6.6,2.9,4.6,1.3,1
59,5.2,2.7,3.9,1.4,1
60,5.0,2.0,3.5,1.0,1
61,5.9,3.0,4.2,1.5,1
62,6.0,2.2,4.0,1.0,1
63,6.1,2.9,4.7,1.4,1
64,5.6,2.9,3.6,1.3,1
65,6.7,3.1,4.4,1.4,1
66,5.6,3.0,4.5,1.5,1
67,5.8,2.7,4.1,1.0,1
68,6.2,2.2,4.5,1.5,1
69,5.6,2.5,3.9,1.1,1
70,5.9,3.2,4.8,1.8,1
71,6.1,2.8,4.0,1.3,1
```

In [28]:
```
# Now, I see index of each row is saved as well, which is a new item and it is now the 1st
column of the table.
# I do not want the index column to be saved - so, I use a modified command below:
iriscp.to_csv('myiriscp_nonewcolumn.csv', index=False) # So, I made the index=False and it
worked (see below)
```

In [29]:
```
# I want to read from this file - which I might need to do in future.
# I am reading in, say, 'irisnewcp' DataFrame
irisnewcp = pd.read_csv('myiriscp_nonewcolumn.csv')
irisnewcp
```

Out[29]:

	sepal_length	sepal_width	petal_length	petal_width	class
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 [30]:
```
# I see the index column above but not in the file - so it is created on the fly. Check th
e columns:
irisnewcp.columns
```

Out[30]:
```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class '], dtype='o
bject')
```

In [31]:
```
# I see the dataset is originally sorted (class-wise), which is not a good idea for machin
e learning - let us unsort it
from sklearn.utils import shuffle # NOTE: sklearn (Scikit-learn) will be our main Machine
Learning python library
```

In [32]:
```
irisnewcp_sh=shuffle(irisnewcp, random_state=345) # 'random_state' is used for initializin
g the internal random number generator
```

```
In [33]: irisnewcp_sh
```

Out[33]:

	sepal_length	sepal_width	petal_length	petal_width	class
34	4.9	3.1	1.5	0.1	0
134	6.1	2.6	5.6	1.4	2
78	6.0	2.9	4.5	1.5	1
27	5.2	3.5	1.5	0.2	0
10	5.4	3.7	1.5	0.2	0
...	...	...	...	...	...
75	6.6	3.0	4.4	1.4	1
42	4.4	3.2	1.3	0.2	0
137	6.4	3.1	5.5	1.8	2
83	6.0	2.7	5.1	1.6	1
24	4.8	3.4	1.9	0.2	0

150 rows × 5 columns

```
In [34]: X=irisnewcp_sh.iloc[:,0:4] # 'iloc' is integer index based, so you have to specify rows and columns by their integer value of the index
         X
```

Out[34]:

	sepal_length	sepal_width	petal_length	petal_width
34	4.9	3.1	1.5	0.1
134	6.1	2.6	5.6	1.4
78	6.0	2.9	4.5	1.5
27	5.2	3.5	1.5	0.2
10	5.4	3.7	1.5	0.2
...	...	...	...	...
75	6.6	3.0	4.4	1.4
42	4.4	3.2	1.3	0.2
137	6.4	3.1	5.5	1.8
83	6.0	2.7	5.1	1.6
24	4.8	3.4	1.9	0.2

150 rows × 4 columns

```
In [35]: y=irisnewcp_sh.iloc[:,4:5]
```

In [36]:

```
y
```

Out[36]:

	class
34	0
134	2
78	1
27	0
10	0
...	...
75	1
42	0
137	2
83	1
24	0

150 rows × 1 columns

In [37]:

```
# Let us use kNN, with k=5, from sklearn
from sklearn.neighbors import KNeighborsClassifier
```

In [38]:

```
# create an instance of KNeighborsClassifier along with necessary parameters
knn = KNeighborsClassifier(n_neighbors=5)
```

In [39]:

```
# print the instance variable to see the parameters of knn
print(knn)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [40]:

```
# Train the classifier with the dataset
knn.fit(X,y)
```

```
E:\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y to (n_sam
ples, ), for example using ravel().
```

Out[40]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [41]:

```
# Since index column (& header) is a problem now, I need to drop the index column (& header) from both X and y
# Also, sklearn expects X, y in array
X=X.values.tolist() # 'values' are the content without the header and index of the DataFrame. toList converts into array
X
```

```
Out[41]: [[4.9, 3.1, 1.5, 0.1],
          [6.1, 2.6, 5.6, 1.4],
          [6.0, 2.9, 4.5, 1.5],
          [5.2, 3.5, 1.5, 0.2],
          [5.4, 3.7, 1.5, 0.2],
          [5.5, 2.3, 4.0, 1.3],
          [6.1, 3.0, 4.9, 1.8],
          [5.1, 3.8, 1.9, 0.4],
          [5.7, 2.5, 5.0, 2.0],
          [6.1, 2.8, 4.7, 1.2],
          [5.7, 3.0, 4.2, 1.2],
          [5.0, 3.3, 1.4, 0.2],
          [6.4, 3.2, 5.3, 2.3],
          [4.8, 3.1, 1.6, 0.2],
          [6.1, 2.9, 4.7, 1.4],
          [5.8, 2.7, 5.1, 1.9],
          [5.2, 4.1, 1.5, 0.1],
          [5.4, 3.4, 1.7, 0.2],
          [7.4, 2.8, 6.1, 1.9],
          [5.7, 3.8, 1.7, 0.3],
          [5.6, 2.7, 4.2, 1.3],
          [5.0, 3.0, 1.6, 0.2],
          [6.3, 3.4, 5.6, 2.4],
          [5.1, 3.5, 1.4, 0.2],
          [5.0, 2.3, 3.3, 1.0],
          [4.3, 3.0, 1.1, 0.1],
          [7.7, 2.8, 6.7, 2.0],
          [6.9, 3.2, 5.7, 2.3],
          [5.8, 2.7, 5.1, 1.9],
          [5.7, 2.6, 3.5, 1.0],
          [5.4, 3.4, 1.5, 0.4],
          [5.8, 2.6, 4.0, 1.2],
          [6.7, 3.1, 4.4, 1.4],
          [5.1, 3.8, 1.5, 0.3],
          [5.0, 3.4, 1.5, 0.2],
          [4.4, 3.0, 1.3, 0.2],
          [5.8, 2.7, 3.9, 1.2],
          [6.2, 2.8, 4.8, 1.8],
          [4.9, 3.1, 1.5, 0.1],
          [5.9, 3.2, 4.8, 1.8],
          [6.8, 3.2, 5.9, 2.3],
          [4.8, 3.0, 1.4, 0.3],
          [4.9, 2.4, 3.3, 1.0],
          [5.0, 3.2, 1.2, 0.2],
          [5.4, 3.9, 1.3, 0.4],
          [5.7, 2.8, 4.1, 1.3],
          [4.4, 2.9, 1.4, 0.2],
          [4.9, 3.0, 1.4, 0.2],
          [5.1, 3.4, 1.5, 0.2],
          [5.5, 2.4, 3.7, 1.0],
          [6.3, 2.9, 5.6, 1.8],
          [6.9, 3.1, 5.4, 2.1],
          [6.7, 3.0, 5.0, 1.7],
          [5.8, 2.8, 5.1, 2.4],
          [5.5, 3.5, 1.3, 0.2],
          [6.3, 3.3, 4.7, 1.6],
          [4.6, 3.4, 1.4, 0.3],
          [6.7, 3.3, 5.7, 2.5],
          [7.1, 3.0, 5.9, 2.1],
          [7.6, 3.0, 6.6, 2.1],
          [7.7, 3.0, 6.1, 2.3],
          [6.9, 3.1, 4.9, 1.5],
          [7.9, 3.8, 6.4, 2.0],
          [6.4, 2.9, 4.3, 1.3],
          [4.9, 2.5, 4.5, 1.7],
          [5.3, 3.7, 1.5, 0.2],
          [6.7, 2.5, 5.8, 1.8],
          [6.0, 3.4, 4.5, 1.6],
          [6.4, 2.8, 5.6, 2.2],
          [6.5, 3.2, 5.1, 2.0],
          [5.7, 4.4, 1.5, 0.4],
          [5.5, 4.2, 1.4, 0.2],
          [5.5, 2.5, 4.0, 1.3],
```

In [42]:

```
# flatten() will remove the header and will convert y in a 1d array.
#You can also use .ravel().  .ravel() returns a view and .flatten() return a copy
y=y.values.flatten()
y
```

Out[42]:

```
array([0, 2, 1, 0, 0, 1, 2, 0, 2, 1, 1, 0, 2, 0, 1, 2, 0, 0, 2, 0, 1, 0,
       2, 0, 1, 0, 2, 2, 2, 1, 0, 1, 1, 0, 0, 0, 1, 2, 0, 1, 2, 0, 1, 0,
       0, 1, 0, 0, 0, 1, 2, 2, 1, 2, 0, 1, 0, 2, 2, 2, 2, 1, 2, 1, 2, 0,
       2, 1, 2, 2, 0, 0, 1, 0, 2, 2, 1, 2, 2, 0, 1, 1, 1, 2, 1, 0, 2, 1,
       2, 1, 0, 1, 0, 2, 1, 0, 1, 0, 0, 0, 2, 2, 1, 2, 1, 2, 0, 2, 2, 1,
       0, 2, 1, 2, 1, 1, 1, 1, 1, 1, 0, 2, 2, 0, 2, 0, 2, 2, 1, 1, 0, 2,
       2, 2, 1, 0, 1, 1, 0, 1, 2, 0, 0, 0, 1, 1, 0, 2, 1, 0], dtype=int64)
```

In [43]:

```
# Now, try to train again
knn.fit(X,y)
```

Out[43]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [44]:

```
# create sample test dataset => expected answers are: 0, 2, 1
X_test = [4.8, 2.9, 1.54, 0.15], [5.9, 2.5, 5.5, 1.2], [5.9, 3.0, 4.6, 1.4]
```

In [45]:

```
# predict the class to which the sample falls into
knn.predict(X_test)
```

Out[45]:

```
array([0, 2, 1], dtype=int64)
```

Save and Load the Model

In [46]:

```
# Python pickle module is used for serializing and de-serializing a Python object structur
e
import pickle
# Note: you can also use joblib
# joblib is optimized to be fast and robust on large data in particular
# to write use 'joblib.dump'  & to read use 'joblib.load'
```

In [47]:

```
# Save the model
f1=open('iris_saved_knn_model','wb') # wb => write binary
pickle.dump(knn, f1)
```

In [48]:

```
# better close (or flush) a file when done.
f1.close()
```

In [49]:

```
# Load the model & Test
f2=open('iris_saved_knn_model', 'rb')
loaded_model = pickle.load(f2)
```

In [50]:

```
X_test = [4.8, 2.9, 1.54, 0.15], [5.9, 2.5, 5.5, 1.2], [5.9, 3.0, 4.6, 1.4]
```

In [51]:

```
loaded_model.predict(X_test)
```

Out[51]:

```
array([0, 2, 1], dtype=int64)
```

In [52]:

```
# If you know the test answers and want to compute the accuracy then do the following
Y_test=[0,2,1]
accuracy = loaded_model.score(X_test, Y_test)
```

In [53]:

```
print(accuracy)
```

```
1.0
```

In [54]:

```
f2.close()
```

References:

[1] https://en.wikipedia.org/wiki/Sepal (https://en.wikipedia.org/wiki/Sepal)

[2] http://suruchifialoke.com/2016-10-13-machine-learning-tutorial-iris-classification/ (http://suruchifialoke.com/2016-10-13-machine-learning-tutorial-iris-classification/)