Support Vector Machines (SVM)

Imports

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Loading Dataset

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

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

In [4]:
```
type(iris)
```

Out[4]:
```
sklearn.utils.Bunch
```

See all the keys

In [5]:
```
iris.keys()
```

Out[5]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filenam
e'])
```

Description

In [6]:
```
print(iris["DESCR"])
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
```

```
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  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.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

Data

In [7]:
```
iris["data"][:5] # Independent columns
```

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]])
```

In [8]:
```
iris["feature_names"] # Column names
```

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

In [9]:
```
np.array(iris["target"]) # Dependent column
```

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, 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])
```

In [10]:
```
iris["target_names"]
# So, 0 is setosa, 1 is versicolor and 2 is virginica
```

Out[10]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

Create Dataframe

In [11]:
```
df = pd.DataFrame(data=iris["data"], columns=iris["feature_names"])
df["target"] = iris["target"]
df.head()
```

Out[11]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
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 [12]:
```
df["target"].unique()
```

Out[12]:
```
array([0, 1, 2])
```

In [13]:
```
df["target"].replace(range(3), iris["target_names"], inplace=True)
```

In [14]:
```
df.head()
```

Out[14]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
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 [15]:
```
df["target"].unique()
```

Out[15]:
```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

EDA

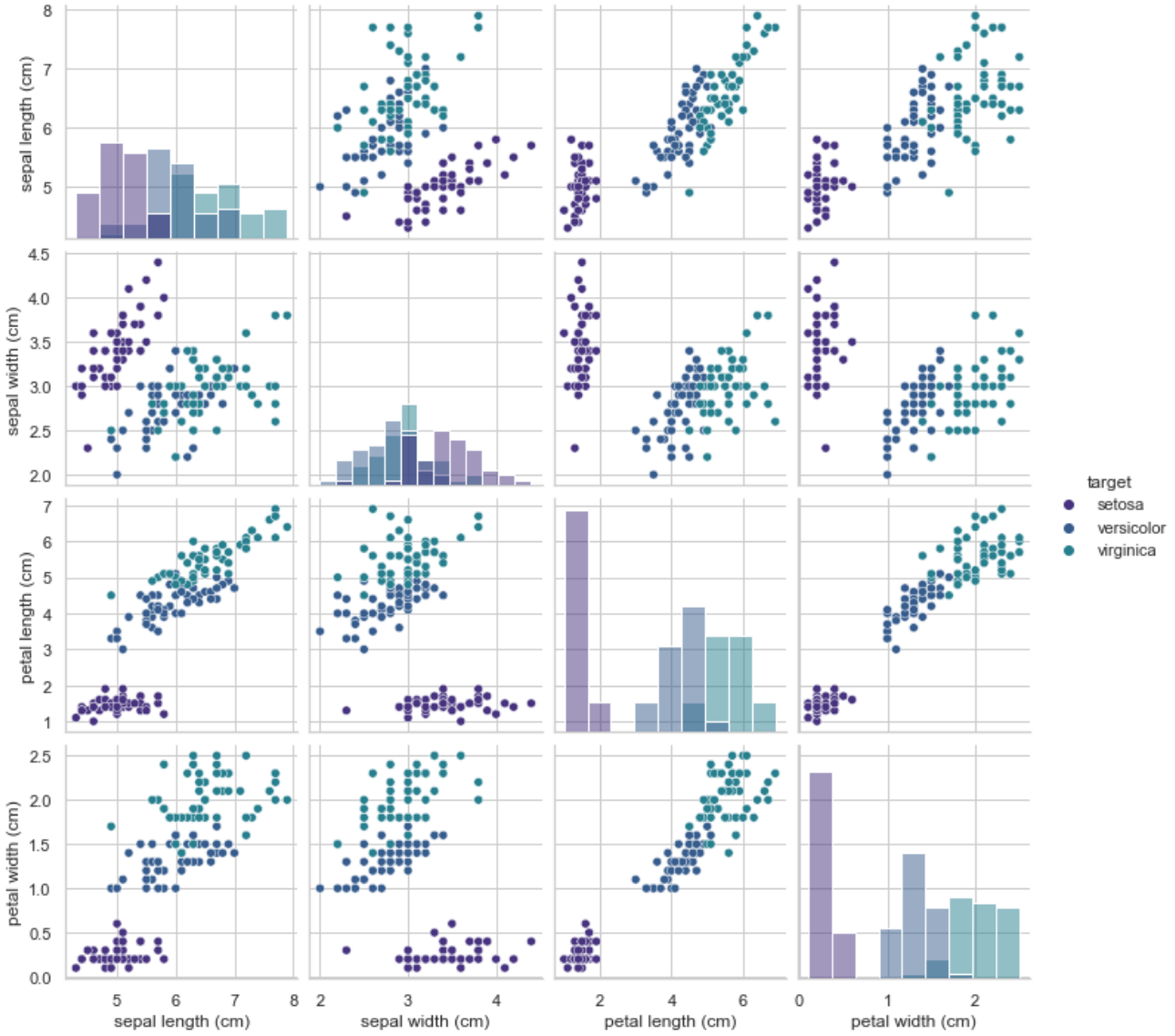
In [16]:
```
sns.set(style="whitegrid", palette="viridis")
```

In [17]:
```

```

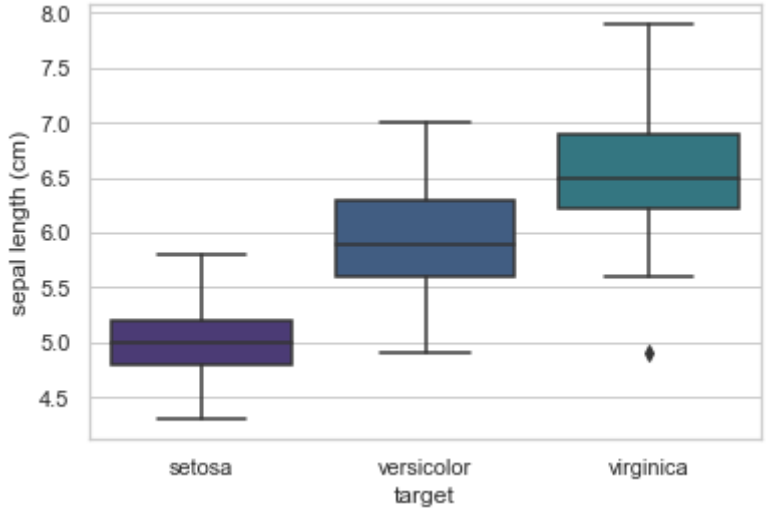
```
sns.pairplot(df, hue="target", diag_kind='hist')
```

Out[17]: <seaborn.axisgrid.PairGrid at 0x13e3c1e6520>



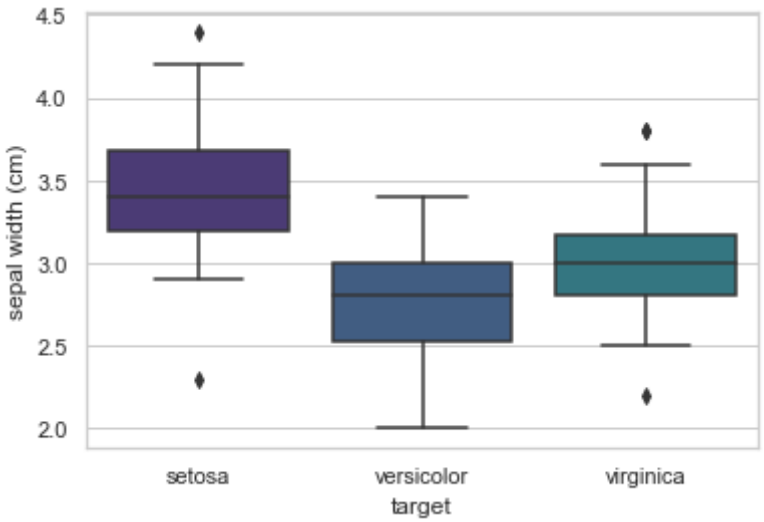
In [18]:
```
sns.boxplot(x="target", y="sepal length (cm)", data=df)
```

Out[18]: <AxesSubplot:xlabel='target', ylabel='sepal length (cm)'>



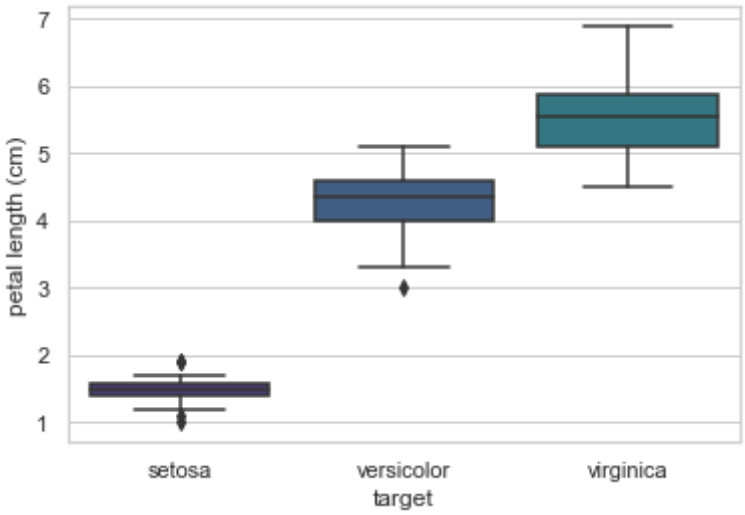
In [19]:
```
sns.boxplot(x="target", y="sepal width (cm)", data=df)
```

Out[19]: <AxesSubplot:xlabel='target', ylabel='sepal width (cm)'>



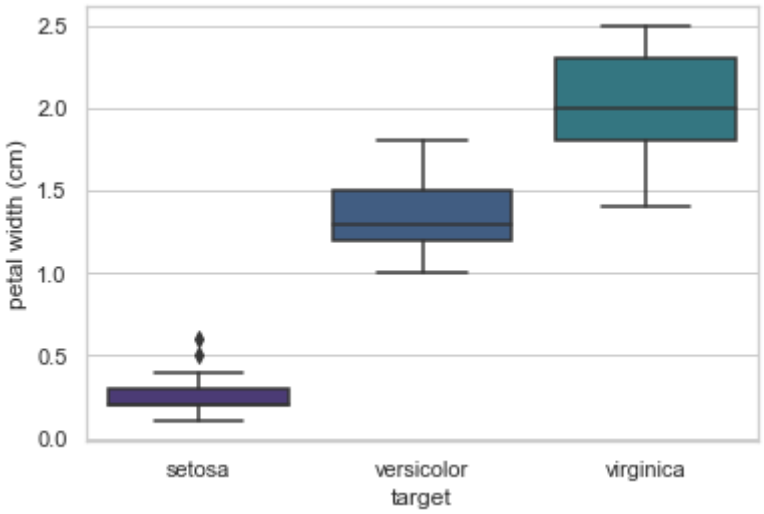
In [20]:
```
sns.boxplot(x="target", y="petal length (cm)", data=df)
```

Out[20]: <AxesSubplot:xlabel='target', ylabel='petal length (cm)'>



In [21]:
```
sns.boxplot(x="target", y="petal width (cm)", data=df)
```

Out[21]: <AxesSubplot:xlabel='target', ylabel='petal width (cm)'>



Feature Selection

In [22]:
```
df.head()
```

Out[22]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
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 [23]:
```
X = df.drop("target", axis=1) # Independent
y = df["target"] # Dependent
```

Train Test Split

In [24]:
```
from sklearn.model_selection import train_test_split
```

In [25]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=202
```

In [26]:
```
X_train[:5]
```

Out[26]:

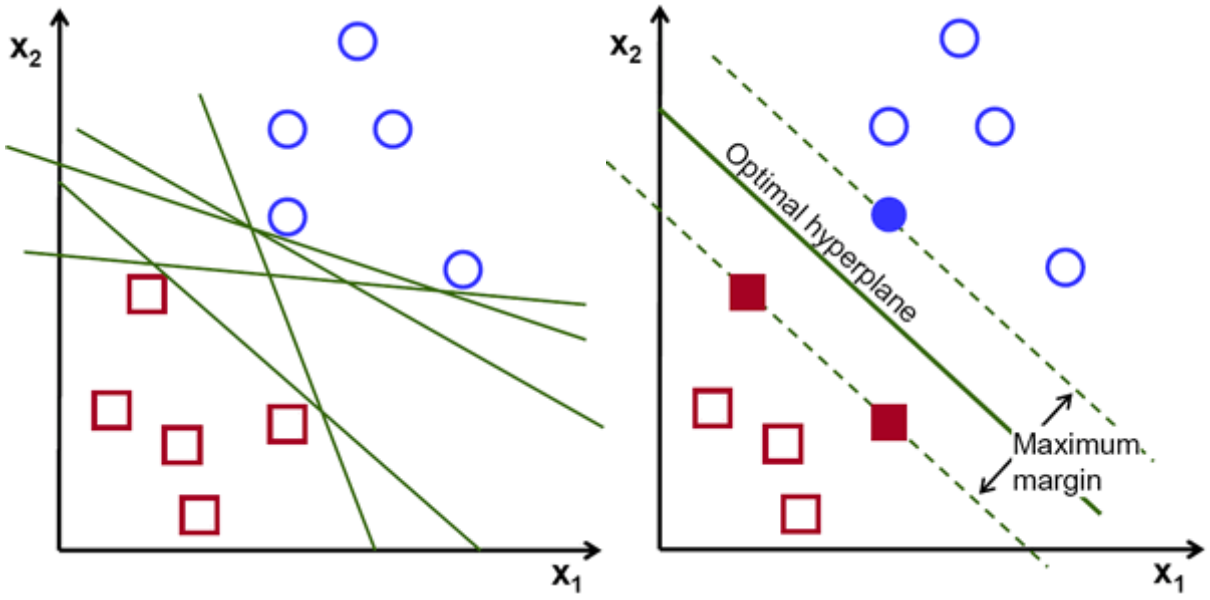
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
120	6.9	3.2	5.7	2.3
117	7.7	3.8	6.7	2.2
76	6.8	2.8	4.8	1.4
86	6.7	3.1	4.7	1.5
57	4.9	2.4	3.3	1.0

In [27]:
```
y_train[:5]
```

Out[27]:
```
120     virginica
117     virginica
76     versicolor
86     versicolor
57     versicolor
Name: target, dtype: object
```

Support Vector Classifier

A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N is the number of features) that distinctly classifies the data points.

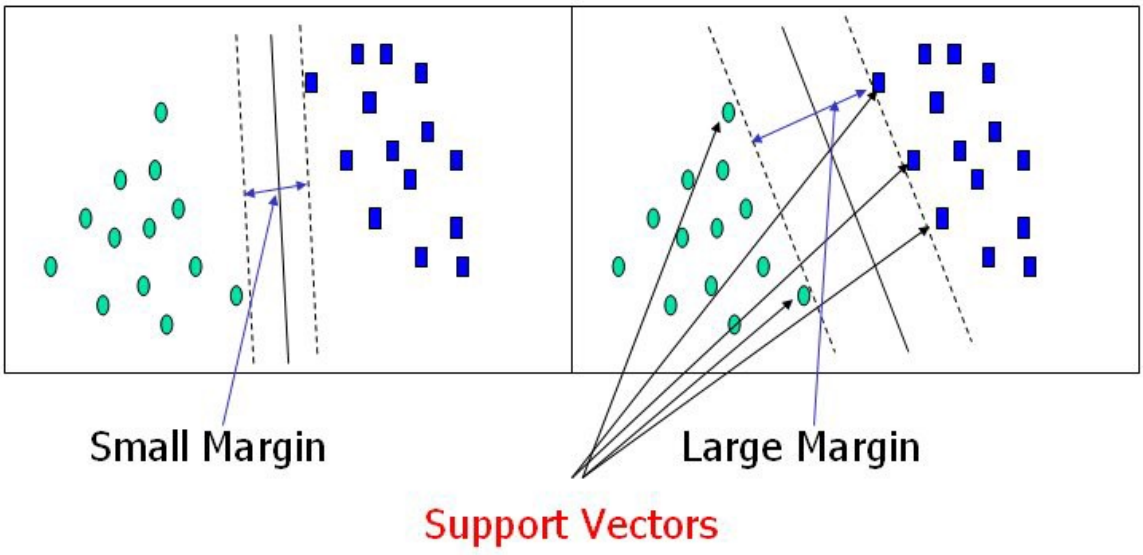


To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

Hyperplanes and Support Vectors

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.



Article Link: Link 1

In [28]:
```
from sklearn.svm import SVC
```

In [29]:
```
svc_model = SVC()
```

In [30]:
```
#Fitting SVM to the Training set
svc_model.fit(X_train, y_train)
```

Out[30]:
```
SVC()
```

In [31]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [32]:
```
# Predicting the Test set results
svc_pred = svc_model.predict(X_test)
```

In [33]:
```
# Making the Confusion Matrix
print(confusion_matrix(y_test, svc_pred))
```

```
[[18  0  0]
 [ 0 12  1]
 [ 0  3 11]]
```

In [34]:
```
print(classification_report(y_test, svc_pred))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        18
  versicolor       0.80      0.92      0.86        13
   virginica       0.92      0.79      0.85        14

    accuracy                           0.91        45
   macro avg       0.91      0.90      0.90        45
weighted avg       0.92      0.91      0.91        45
```

Final Model

Now that we have seen quite a few different types of models, we have determined that among these models k-NN model with k=7 gives us the best results. So, let's create out final model using that.

In [35]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [36]:
```
final_model = KNeighborsClassifier(n_neighbors=7)
final_model.fit(X, y)
```

Out[36]:
```
KNeighborsClassifier(n_neighbors=7)
```

In [37]:
```
# For final model, evaluation is not necessary as it will be biased
final_pred = final_model.predict(X)
print(confusion_matrix(y, final_pred))
print(classification_report(y, final_pred))
```

```
[[50  0  0]
 [ 0 47  3]
 [ 0  1 49]]
```

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	50
versicolor	0.98	0.94	0.96	50
virginica	0.94	0.98	0.96	50
accuracy			0.97	150
macro avg	0.97	0.97	0.97	150
weighted avg	0.97	0.97	0.97	150