In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
iris = load_iris()

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

📌 dir(iris) — Inspecting Available Attributes and Methods

In [2]:
```
dir(iris)
```

Out[2]:
```
['DESCR',
 'data',
 'data_module',
 'feature_names',
 'filename',
 'frame',
 'target',
 'target_names']
```

In [3]:
```
iris.feature_names
```

Out[3]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [4]:
```
df = pd.DataFrame(iris.data, columns = iris.feature_names)
df
```

Out[4]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [5]:
```
df['target'] = iris.target
df
```

Out[5]:

	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
...	...	...	...	...	...
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 [6]:
```
iris.target_names
```

Out[6]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [7]:
```
df['flower_name'] = df.target.apply(lambda x: iris.target_names[x])
```

In [8]:
```
df
```

Out[8]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2	virginica
146	6.3	2.5	5.0	1.9	2	virginica
147	6.5	3.0	5.2	2.0	2	virginica
148	6.2	3.4	5.4	2.3	2	virginica
149	5.9	3.0	5.1	1.8	2	virginica

150 rows × 6 columns

📌 Splitting the Dataset by Iris Species

In [9]:

```
df0 = df[df.target==0]
df1 = df[df.target==1]
df2 = df[df.target==2]
```

In [10]:

```
df0.head()
```

Out[10]:

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

In [11]:

```
df1.head()
```

Out[11]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
50	7.0	3.2	4.7	1.4	1	versicolor
51	6.4	3.2	4.5	1.5	1	versicolor
52	6.9	3.1	4.9	1.5	1	versicolor
53	5.5	2.3	4.0	1.3	1	versicolor
54	6.5	2.8	4.6	1.5	1	versicolor

In [12]:
```
df2.head()
```

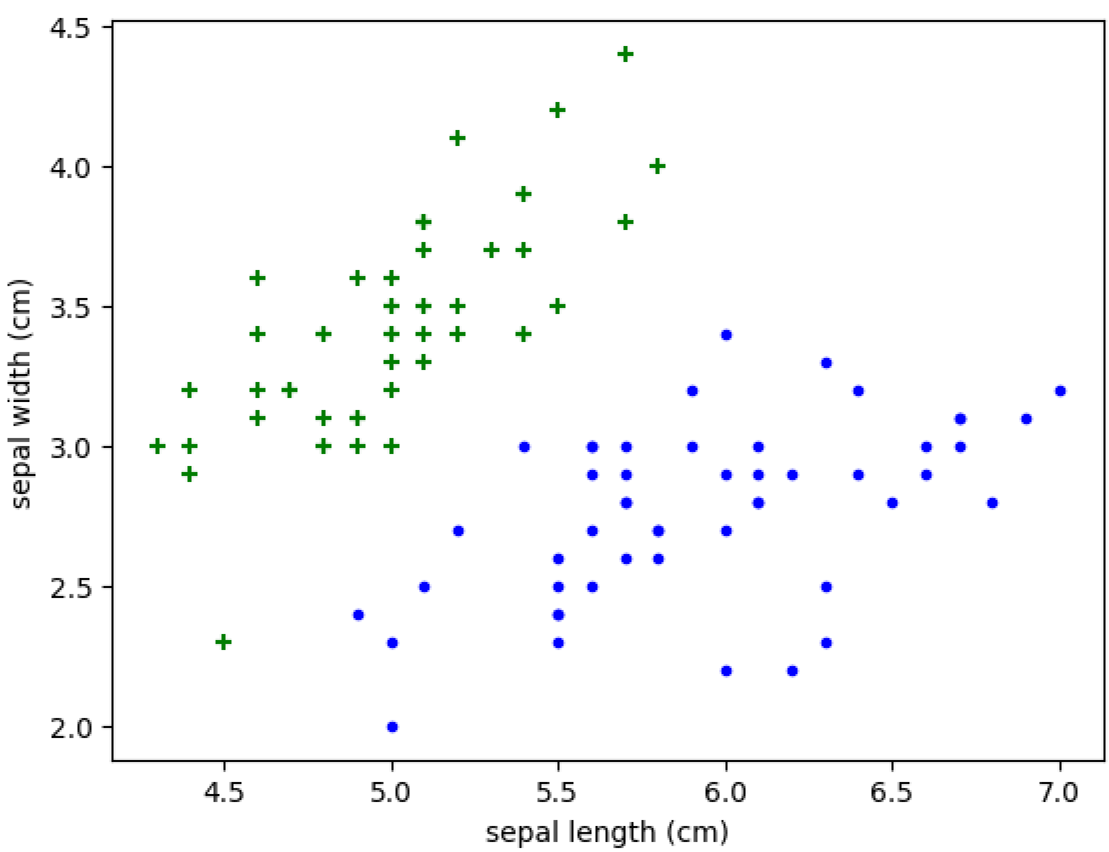
Out[12]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
100	6.3	3.3	6.0	2.5	2	virginica
101	5.8	2.7	5.1	1.9	2	virginica
102	7.1	3.0	5.9	2.1	2	virginica
103	6.3	2.9	5.6	1.8	2	virginica
104	6.5	3.0	5.8	2.2	2	virginica

📊 Scatter Plot of Sepal Length vs Sepal Width by Species

In [13]:
```
plt.scatter(df0['sepal length (cm)'],df0['sepal width (cm)'], color='green',ma
plt.scatter(df1['sepal length (cm)'],df1['sepal width (cm)'], color='blue', ma
plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')
```

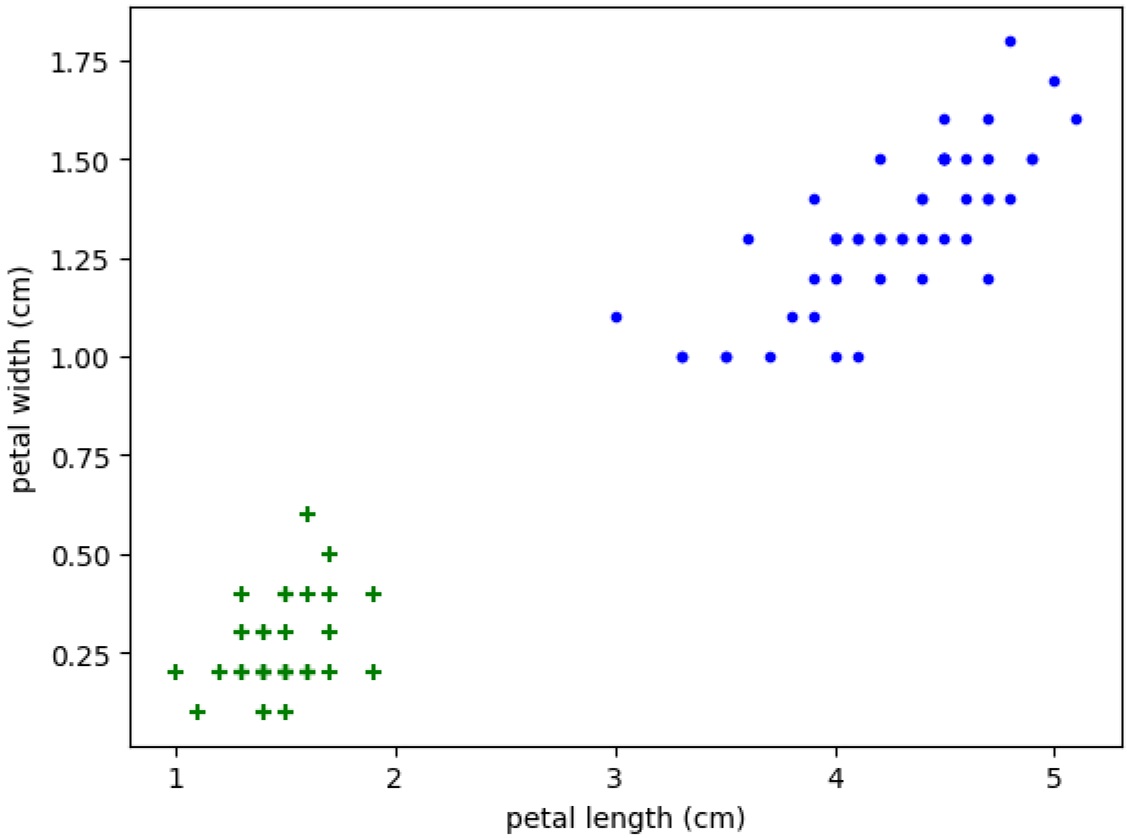
Out[13]: Text(0, 0.5, 'sepal width (cm)')



📊 Scatter Plot of petal Length vs petal Width by Species

In [14]:
```
plt.scatter(df0['petal length (cm)'],df0['petal width (cm)'],color='green',marl
plt.scatter(df1['petal length (cm)'],df1['petal width (cm)'],color='blue',marke
plt.xlabel('petal length (cm)')
plt.ylabel('petal width (cm)')
```

Out[14]: Text(0, 0.5, 'petal width (cm)')



🎯 **Preparing Features and Labels for Machine Learning**

In [15]:
```
X = df.drop(['flower_name','target'],axis=1)
y = df['target']
```

✂️ **Splitting the Dataset into Training and Testing Sets**

In [16]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.8)
```

In [17]:
```
len(X_train)
```

Out[17]: 120

In [18]:
```
len(X_test)
```

Out[18]: 30

Training a Support Vector Machine (SVM) Classifier What SVM does:

It tries to find the best hyperplane that separates the classes in the feature space.

By default, it uses the RBF (Radial Basis Function) kernel, which can handle non-linear boundaries.

In [19]:
```
model = SVC()
```

```
model.fit(X_train, y_train)
```

Out[19]:
```
▾ SVC
SVC()
```

Evaluating the Model Accuracy on Test Data

In [20]:
```
model.score(X_test, y_test)
```

Out[20]:
```
0.9666666666666667
```

Training SVM with Regularization Parameter C=10 SVC(C=10) creates a Support Vector Classifier with a regularization parameter C set to 10. .

📌 **What does C do?** C controls the trade-off between maximizing the margin and minimizing classification error:

Higher C (like 10) → Less tolerance for misclassification → tries to fit the training data more strictly.

Lower C (like 0.1) → More tolerant of misclassifications → allows a wider margin.

In [21]:
```
model_1 = SVC(C=10)

model_1.fit(X_train, y_train)
```

Out[21]:
```
▾ SVC
SVC(C=10)
```

In [22]:
```
model_1.score(X_test, y_test)
```

Out[22]:
```
0.9333333333333333
```

Training SVM with gamma=10 (Kernel Parameter) .

📌 What does gamma do? gamma controls the influence of a single training point in the decision boundary:

Higher gamma → More influence from each individual training point → creates a more complex decision boundary (can lead to overfitting).

Lower gamma → Each point has less influence → creates a simpler, smoother decision boundary (can lead to underfitting).

In [23]:
```
model_2 = SVC(gamma=10)

model_2.fit(X_train, y_train)
```

Out[23]:
```
▾ SVC
SVC(gamma=10)
```

In [24]:
```
model_2.score(X_test, y_test)
```

Out[24]:
```
0.9
```

Training SVM with a Linear Kernel

📌 What does kernel='linear' mean? The linear kernel assumes that the data is linearly separable, meaning that a straight line (or hyperplane) can divide the classes in the feature space.

This is effective when the data is already well-separated or nearly linear.

In [25]:
```
model_3 = SVC(kernel='linear')

model_3.fit(X_train, y_train)
```

Out[25]:
```
▾ SVC
SVC(kernel='linear')
```

In [26]:
```
model_3.score(X_test, y_test)
```

Out[26]:
```
0.9333333333333333
```

📌 **What is Logistic Regression?Logistic Regression is a probabilistic model used to predict class probabilities and is often used for binary classification.The model outputs a probability score between 0 and 1, representing the likelihood of belonging to a class.**

In [27]:
```
model = LogisticRegression()
```

In [28]:
```
model.fit(X_train, y_train)
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
```

Out[28]:
```
▾ LogisticRegression
LogisticRegression()
```

In [29]:
```
model.score(X_test, y_test)
```
Out[29]:
```
0.9333333333333333
```

📌 **What is a Confusion Matrix?** A confusion matrix is a table used to evaluate the performance of a classification model. It provides the following:

True Positives (TP): Correctly predicted positive samples.

True Negatives (TN): Correctly predicted negative samples.

False Positives (FP): Incorrectly predicted as positive.

False Negatives (FN): Incorrectly predicted as negative.

For multi-class classification (like the iris dataset), it will be a square matrix where:

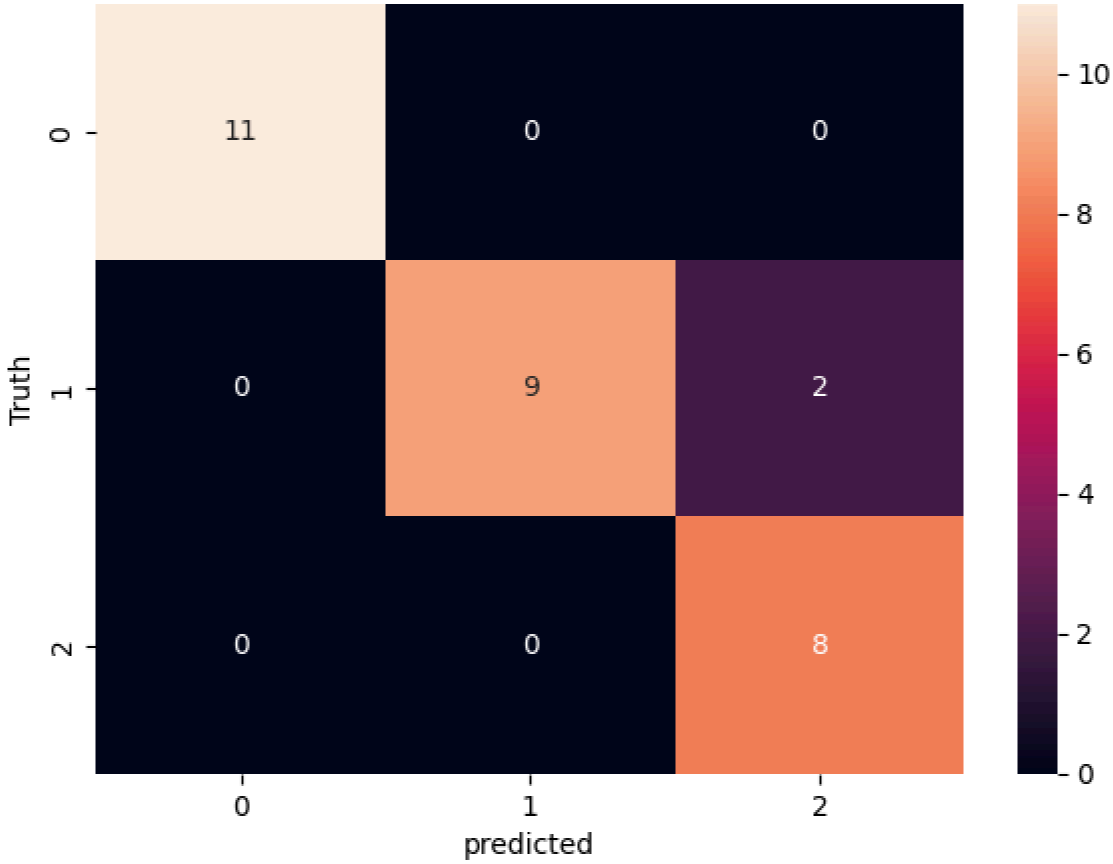
Rows represent the true class.

Columns represent the predicted class.

In [30]:
```
y_pred = model.predict(X_test)
confusion_m = confusion_matrix(y_test, y_pred)
confusion_m
```
Out[30]:
```
array([[11,  0,  0],
       [ 0,  9,  2],
       [ 0,  0,  8]])
```

📌 **Why use a Heatmap?A heatmap is a great way to visualize the confusion matrix because it makes it easy to see:Which classes are most frequently misclassified.The distribution of errors across classes.**

In [31]:
```
plt.figure(figsize=(7,5))
sns.heatmap(confusion_m, annot=True)
plt.xlabel('predicted')
plt.ylabel('Truth')
```
Out[31]:
```
Text(58.222222222222214, 0.5, 'Truth')
```



📌 **Why use a Classification Report?The classification report provides a more comprehensive view of your model's performance across each class:Precision is important when false positives are costly.Recall is important when false negatives are costly.F1-score balances precision and recall, which is useful when you need a single metric for model performance.**

In [32]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       1.00      0.82      0.90        11
           2       0.80      1.00      0.89         8

    accuracy                           0.93        30
   macro avg       0.93      0.94      0.93        30
weighted avg       0.95      0.93      0.93        30
```

📊 **Classification ReportThe classification report shows key metrics for each class (0, 1, and 2), as well as overall accuracy and averages.**

Class-wise Metrics: Class 0 (Setosa):

Precision: 1.00 – Every prediction of class 0 is correct.

Recall: 1.00 – All actual class 0 instances were correctly predicted.

F1-score: 1.00 – Perfect balance between precision and recall.

Support: 12 – There are 12 instances of class 0 in the test set.

Class 1 (Versicolor):

Precision: 1.00 – All predictions for class 1 are correct.

Recall: 0.91 – 91% of the actual class 1 instances were correctly predicted.

F1-score: 0.95 – High balance between precision and recall.

Support: 11 – There are 11 instances of class 1.

Class 2 (Virginica):

Precision: 0.88 – 88% of predicted class 2 instances are correct.

Recall: 1.00 – All actual class 2 instances were correctly predicted.

F1-score: 0.93 – Good balance between precision and recall.

Support: 7 – There are 7 instances of class 2.

In [33]:
```
y_test.shape
```
Out[33]:
```
(30,)
```

Accuracy

In [34]:
```
round((12+10+7)/(12+10+1+7),2)
```
Out[34]:
```
0.97
```

Precision for 1 class

In [35]:
```
round(10/(10+0),2)
```
Out[35]:
```
1.0
```

In [36]:
```
round(7/(7+1),2)
```
Out[36]:
```
0.88
```

Recall mean total Truth

In [37]:
```
round(10/(10+1),2)
```
Out[37]:
```
0.91
```

In [38]:
```
round(12/(12+0),2)
```

Out[38]: 1.0

F1-Score

2((precision.recall)/(precision+recall))

In [39]:
```
round(2*(1*0.91)/(1+0.91),2)
```

Out[39]: 0.95

In [40]:
```
round(2*(0.88*1)/(0.88+1),2)
```

Out[40]: 0.94

.

📌 **Interpretation:The model has performed excellently with an overall accuracy of 97%.Class 1 has a slightly lower recall (0.91), meaning some instances of Versicolor were misclassified.The F1-scores are close to perfect for all classes, indicating a good balance between precision and recall.**

In [ ]:
```

```