

```
In [2]: from sklearn.svm import SVC
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import datasets
```

```
In [4]: df=datasets.load_iris(as_frame=True).frame
```

```
In [6]: df.isnull().sum()
```

```
Out[6]: sepal length (cm)    0
sepal width (cm)           0
petal length (cm)          0
petal width (cm)           0
target                    0
dtype: int64
```

```
In [7]: df.shape
```

```
Out[7]: (150, 5)
```

```
In [5]: df.head()
```

```
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

```
In [8]: X=df.drop("target",axis=1)
y=df["target"]
```

```
In [9]: X.head()
```

```
Out[9]:
```

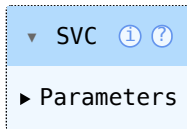
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
In [10]: X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.3, random_state=42, stratify=y
)
#stratify=y Maintains same class distribution in train & test sets
# If dataset has:
# 70% Class A
# 30% Class B
# Then both train & test will preserve this ratio
```

```
In [11]: from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X_train_scaled=scaler.fit_transform(X_train)
X_test_scaled=scaler.transform(X_test)
```

```
In [15]: # model
svc=SVC()
svc.fit(X_train_scaled,y_train)
```

Out[15]:



```
In [16]: y_pred=svc.predict(X_test_scaled)
```

```
In [17]: # Evaluate
from sklearn.metrics import accuracy_score, classification_report
print("accuracy:", accuracy_score(y_test, y_pred))
print("classification report:\n", classification_report(y_test, y_pred))
```

```
accuracy: 0.9333333333333333
classification report:
              precision    recall  f1-score   support

     0           1.00        1.00        1.00        15
     1           0.88        0.93        0.90        15
     2           0.93        0.87        0.90        15

 accuracy
macro avg       0.93        0.93        0.93        45
weighted avg    0.93        0.93        0.93        45
```

```
In [21]: # linear kernel
svc=SVC(kernel="linear")
svc.fit(X_train_scaled, y_train)
y_pred=svc.predict(X_test_scaled)
print("accuracy:", accuracy_score(y_test, y_pred))
```

```
accuracy: 0.9111111111111111
```

```
In [22]: # Polynomial kernel

svc = SVC(kernel="poly")
svc.fit(X_train_scaled, y_train)

y_pred = svc.predict(X_test_scaled)

print("accuracy: ", accuracy_score(y_test, y_pred))
```

```
accuracy: 0.8666666666666667
```

```
In [23]: # Sigmoid kernel

svc = SVC(kernel="sigmoid")
svc.fit(X_train_scaled, y_train)

y_pred = svc.predict(X_test_scaled)

print("accuracy: ", accuracy_score(y_test, y_pred))
```

```
accuracy: 0.9111111111111111
```

C is the Regularization parameter. It controls the trade-off between: ♦ Maximizing margin ♦ Minimizing classification error Small C → Large margin, more misclassification allowed Large C → Small margin, strict classifications, less error allowed

```
In [28]: C_vals=[0.5,1,2,3,4,5]
for c_val in C_vals:
    svc=SVC(C=c_val, kernel="rbf")
    svc.fit(X_train_scaled, y_train)

    y_pred = svc.predict(X_test_scaled)
    print("C = ", c_val, "& accuracy: ", accuracy_score(y_test, y_pred))
cs
```

```
C = 0.5 & accuracy: 0.9111111111111111
C = 1 & accuracy: 0.9333333333333333
C = 2 & accuracy: 0.9111111111111111
C = 3 & accuracy: 0.9111111111111111
C = 4 & accuracy: 0.9333333333333333
C = 5 & accuracy: 0.9333333333333333
```

C Value	Behavior	Result
Low C (0.01, 0.1)	High regularization	More bias, less variance → underfitting
Medium C (1)	Balanced	Good generalization
High C (10, 100)	Low regularization	Low bias, high variance → overfitting