In [27]:
```
from sklearn.svm import SVC
from sklearn.datasets import load_iris
import pandas as pd
```

In [28]:
```
iris=load_iris()
```

In [ ]:
```

```

In [29]:
```
dir(iris)
```

Out[29]:
```
['DESCR',
 'data',
 'data_module',
 'feature_names',
 'filename',
 'frame',
 'target',
 'target_names']
```

In [30]:
```
df=pd.DataFrame(iris.data,columns=iris.feature_names)
```

In [31]:
```
iris.target_names
```

Out[31]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [32]:
```
df['flower']=iris.target
```

In [33]:
```
df
```

Out[33]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	flower
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 [34]:
```
from sklearn.model_selection import cross_val_score
```

In [36]:
```
cross_val_score(SVC(kernel="linear",C=1,gamma="auto"),iris.data,iris.target)
```

Out[36]:
```
array([0.96666667, 1.        , 0.96666667, 0.96666667, 1.        ])
```

In [37]:
```
cross_val_score(SVC(kernel="rbf",C=10,gamma="auto"),iris.data,iris.target)
```

Out[37]:
```
array([0.96666667, 1.        , 0.96666667, 0.96666667, 1.        ])
```

In [38]:
```
# we did the cross validation to find out which is the best parameter for model
```

In [39]:
```
# we cannot do this if we have many model so we we gridsearchcv
```

In [41]:
```
from sklearn.model_selection import GridSearchCV
```

In [47]:
```
reg=GridSearchCV(SVC(gamma="auto"),{
    "C":[1,5,10],
    "kernel":["linear","rbf"]

},cv=5,return_train_score=False)# this model follow the k fold technique for training the dataset and then make the score for each parameter

#this is mainly used to find the best parameter
```

In [48]:
```
reg.fit(iris.data,iris.target)
```

Out[48]:

▸ **GridSearchCV**

▸ **estimator: SVC**

▸ SVC

In [50]:
```
reg.cv_results_
```

Out[50]:
```
{'mean_fit_time': array([0.00072331, 0.00059862, 0.00079784, 0.00059867, 0.00079765,
        0.00059838]),
 'std_fit_time': array([0.00039867, 0.00048877, 0.00039892, 0.00048881, 0.00039883,
        0.00048858]),
 'mean_score_time': array([0.00039911, 0.00079765, 0.00019946, 0.00079837, 0.00019941,
        0.00039897]),
 'std_score_time': array([0.00048881, 0.00039883, 0.00039892, 0.00039918, 0.00039883,
        0.00048864]),
 'param_C': masked_array(data=[1, 1, 5, 5, 10, 10],
              mask=[False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_kernel': masked_array(data=['linear', 'rbf', 'linear', 'rbf', 'linear', 'rbf'],
              mask=[False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'C': 1, 'kernel': 'linear'},
  {'C': 1, 'kernel': 'rbf'},
  {'C': 5, 'kernel': 'linear'},
  {'C': 5, 'kernel': 'rbf'},
  {'C': 10, 'kernel': 'linear'},
  {'C': 10, 'kernel': 'rbf'}],
 'split0_test_score': array([0.96666667, 0.96666667, 1.        , 0.96666667, 1.        ,
        0.96666667]),
 'split1_test_score': array([1., 1., 1., 1., 1., 1.]),
 'split2_test_score': array([0.96666667, 0.96666667, 0.93333333, 0.96666667, 0.9       ,
        0.96666667]),
 'split3_test_score': array([0.96666667, 0.96666667, 0.96666667, 0.96666667, 0.96666667,
        0.96666667]),
 'split4_test_score': array([1., 1., 1., 1., 1., 1.]),
 'mean_test_score': array([0.98      , 0.98      , 0.98      , 0.98      , 0.97333333,
        0.98      ]),
 'std_test_score': array([0.01632993, 0.01632993, 0.02666667, 0.01632993, 0.03887301,
        0.01632993]),
 'rank_test_score': array([1, 1, 1, 1, 6, 1])}
```

In [51]:
```
ram=reg.cv_results_
```

In [56]:
```
df=pd.DataFrame(ram)
```

In [57]:
```
df
```

Out[57]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_kernel	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	mean_test_score	std_test_score	rank_test_score
0	0.000723	0.000399	0.000399	0.000489	1	linear	{'C': 1, 'kernel': 'linear'}	0.966667	1.0	0.966667	0.966667	1.0	0.980000	0.016330	1
1	0.000599	0.000489	0.000798	0.000399	1	rbf	{'C': 1, 'kernel': 'rbf'}	0.966667	1.0	0.966667	0.966667	1.0	0.980000	0.016330	1
2	0.000798	0.000399	0.000199	0.000399	5	linear	{'C': 5, 'kernel': 'linear'}	1.000000	1.0	0.933333	0.966667	1.0	0.980000	0.026667	1
3	0.000599	0.000489	0.000798	0.000399	5	rbf	{'C': 5, 'kernel': 'rbf'}	0.966667	1.0	0.966667	0.966667	1.0	0.980000	0.016330	1
4	0.000798	0.000399	0.000199	0.000399	10	linear	{'C': 10, 'kernel': 'linear'}	1.000000	1.0	0.900000	0.966667	1.0	0.973333	0.038873	6
5	0.000598	0.000489	0.000399	0.000489	10	rbf	{'C': 10, 'kernel': 'rbf'}	0.966667	1.0	0.966667	0.966667	1.0	0.980000	0.016330	1

In [59]:
```
df[['param_C',"param_kernel","mean_test_score"]]
```

Out[59]:

	param_C	param_kernel	mean_test_score
0	1	linear	0.980000
1	1	rbf	0.980000
2	5	linear	0.980000
3	5	rbf	0.980000
4	10	linear	0.973333
5	10	rbf	0.980000

In [60]:
```
# here we can see ki C=1 huda ani kernel linear huda ccuracy high so this way we do hyper tuning
```

In [ ]:
```

```