

```
In [ ]: #Data exploration:  
from sklearn import datasets  
data = datasets.load_iris(return_X_y=False, as_frame=True)  
print(data.data.head())  
features_name = data.feature_names
```

	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 [ ]: #statistical analysis  
features = data.data  
classes = data.target_names  
target = data.target  
Iris = features.copy()  
Iris['target'] = target  
Iris.describe()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

```
In [ ]: #explore 3 classes  
classes
```

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

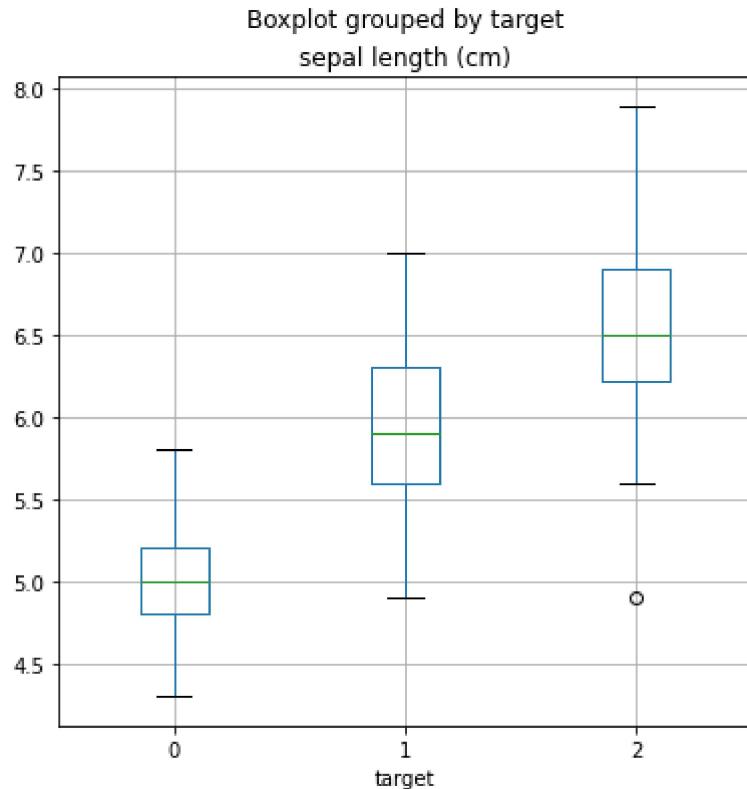
```
In [ ]: features_name
```

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

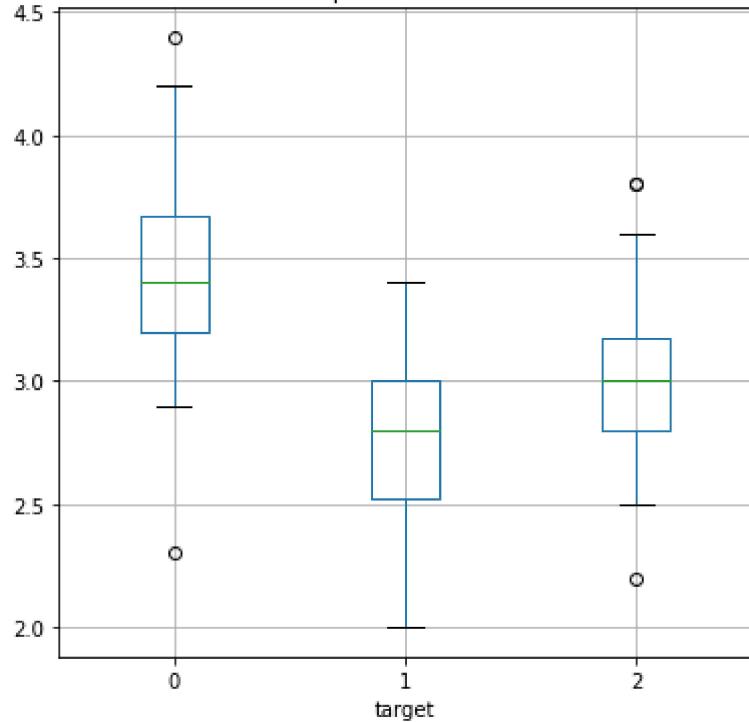
```
In [ ]: Iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   sepal length (cm)    150 non-null   float64 
 1   sepal width (cm)     150 non-null   float64 
 2   petal length (cm)    150 non-null   float64 
 3   petal width (cm)     150 non-null   float64 
 4   target              150 non-null   int32  
dtypes: float64(4), int32(1)
memory usage: 5.4 KB
```

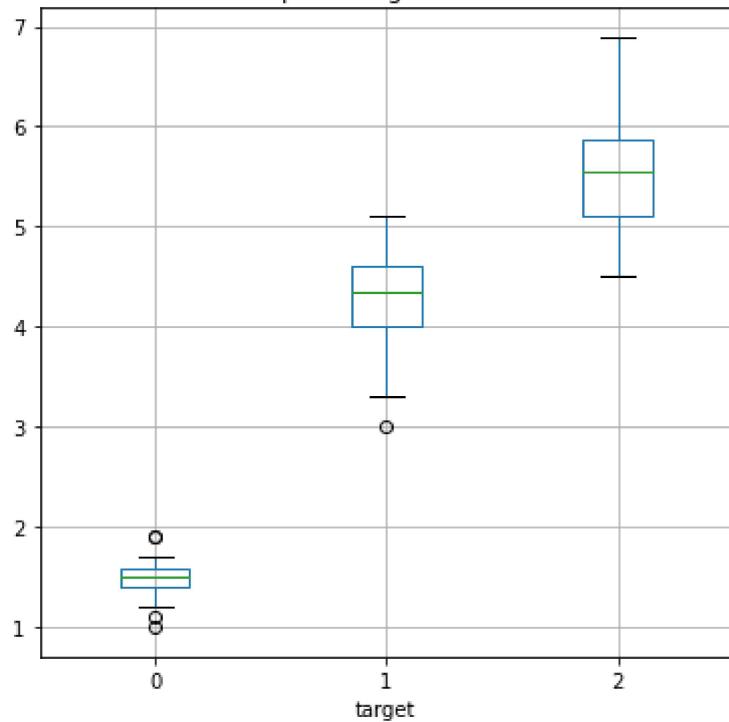
```
In [ ]: #plot each feature among the 3 categorical classes:
from matplotlib import pyplot as plt
#print(Iris.head())
for col in features_name:
    Iris.boxplot(column=col,by='target', figsize=(6,6))
    plt.title(col)
    plt.show()
```

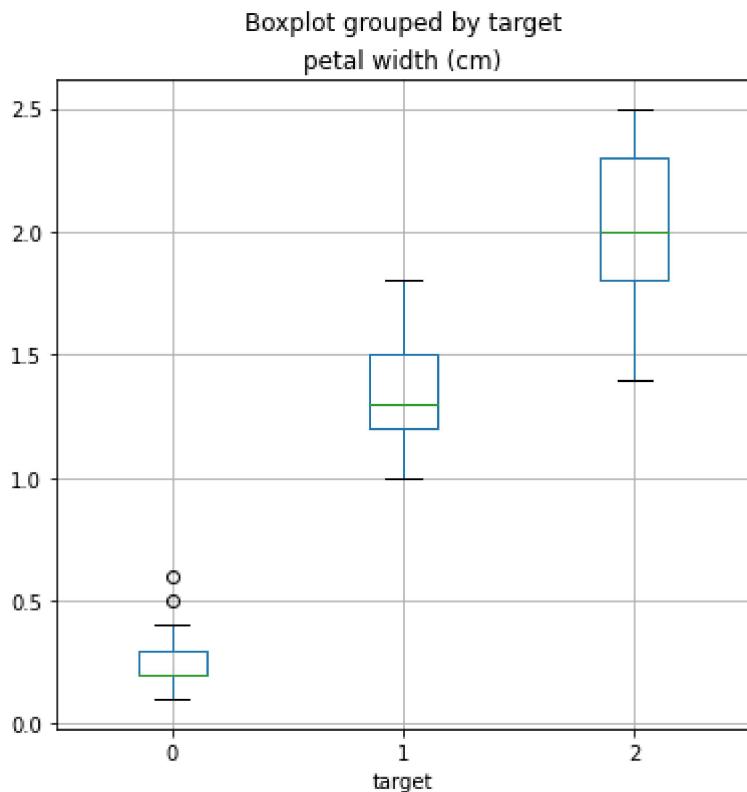


Boxplot grouped by target
sepal width (cm)



Boxplot grouped by target
petal length (cm)





```
In [ ]: #check data imbalanced:  
Iris.groupby('target').count()
```

```
Out[ ]:      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  
  
target  
-----  
 0          50              50              50              50  
 1          50              50              50              50  
 2          50              50              50              50
```

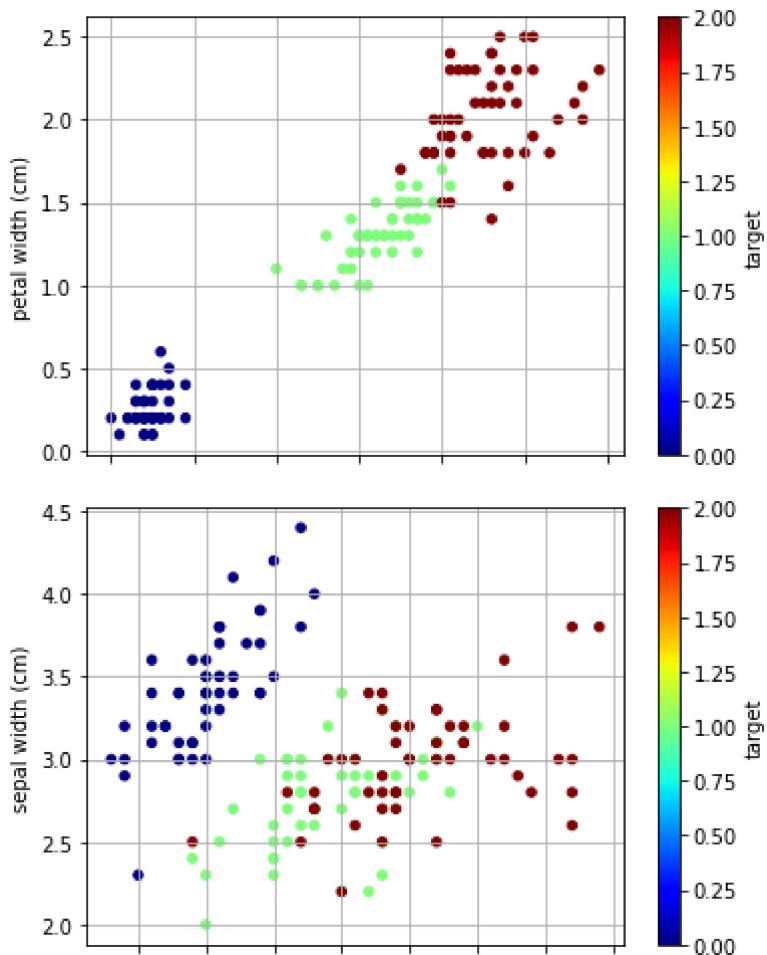
```
In [ ]: #check data Imbalanced  
Iris['target'].value_counts()
```

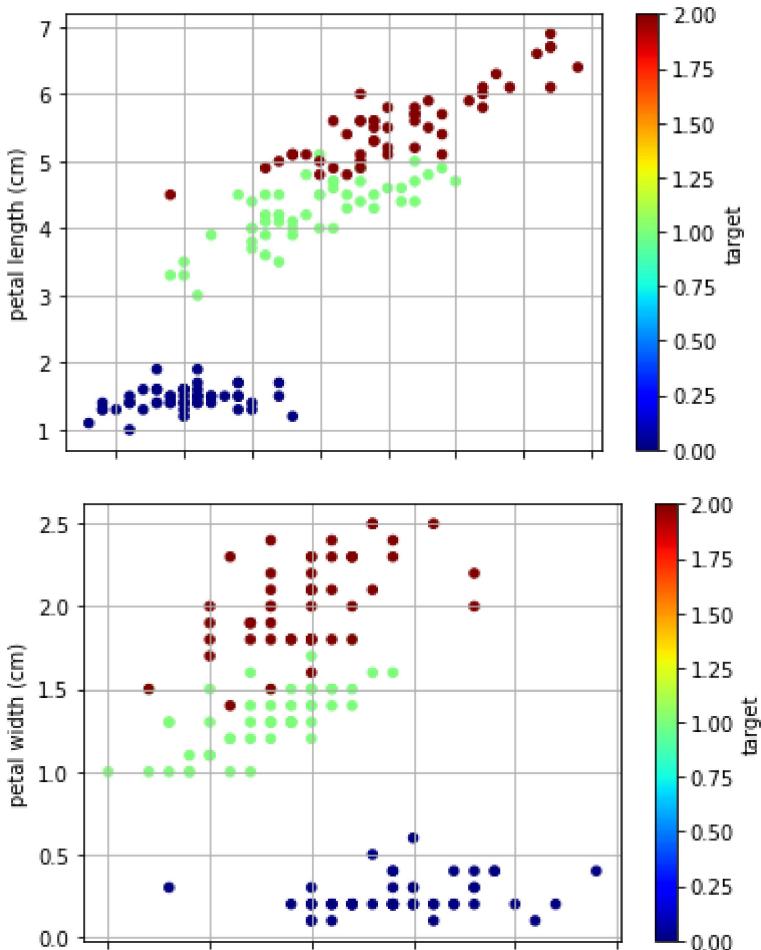
```
Out[ ]: 0    50  
1    50  
2    50  
Name: target, dtype: int64
```

```
In [ ]: #check number of samples, and chek null values  
Iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   sepal length (cm)    150 non-null   float64 
 1   sepal width (cm)     150 non-null   float64 
 2   petal length (cm)    150 non-null   float64 
 3   petal width (cm)     150 non-null   float64 
 4   target              150 non-null   int32  
dtypes: float64(4), int32(1)
memory usage: 5.4 KB
```

```
In [ ]: #represent the dataset for each class clustering :
Iris.plot(kind='scatter',x='petal length (cm)',y='petal width (cm)',grid=True,c='target')
plt.show()
Iris.plot(kind='scatter',x='sepal length (cm)',y='sepal width (cm)',grid=True,c='target')
plt.show()
Iris.plot(kind='scatter',x='sepal length (cm)',y='petal length (cm)',grid=True,c='target')
plt.show()
Iris.plot(kind='scatter',x='sepal width (cm)',y='petal width (cm)',grid=True,c='target')
plt.show()
```





```
In [ ]: #split the data for training and testing
from sklearn.model_selection import train_test_split
X=features
y=target
Xtrain,Xtest,ytrain,ytest=train_test_split( X,y, test_size=0.2,random_state=42,shuffle=True)
Xtrain.shape
```

```
Out[ ]: (120, 4)
```

```
In [ ]: Xtest.shape
```

```
Out[ ]: (30, 4)
```

```
In [ ]: from sklearn.pipeline import make_pipeline,FunctionTransformer
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
def column_ratio(X):
    return X[:, [0]] / X[:, [1]]
def ratio_name(function_transformer, feature_names_in):
    return ["ratio"] # feature names out

def ratio_pipeline():
    return make_pipeline(
        SimpleImputer(strategy="median"),
        FunctionTransformer(column_ratio, feature_names_out=ratio_name))

preprocessing = ColumnTransformer([
    ("widtratio", ratio_pipeline(), ["petal width (cm)", "sepal width (cm)"]),
    ("ratio", ratio_pipeline(), ["petal length (cm)", "sepal length (cm)"])]))
```

```
("lengthratio", ratio_pipeline(), ["petal length (cm)", "sepal length (cm)"]),
("features","passthrough",[ "petal width (cm)", "sepal width (cm)", "petal length (cm)"]]))
```

```
In [ ]: from sklearn.linear_model import SGDClassifier, LogisticRegressionCV, LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
clf = {
    'SGDClassifier': SGDClassifier(random_state=42, alpha=.1, max_iter=30000, tol=.0001, loss='squared'),
    'KNeighborsClassifier': KNeighborsClassifier(n_neighbors=3),
    'DecisionTreeClassifier': DecisionTreeClassifier(random_state=42, max_features=3),
    'RandomForestClassifier': RandomForestClassifier(random_state=42, max_features=3),
    'LinearSVC': LinearSVC(random_state=42, max_iter=30000, class_weight='balanced',
                           multi_class="crammer_singer", C=1, tol=.001),
    'SVC': SVC(C=1, max_iter=1000, tol=.001),
    'logistic_reg': LogisticRegression(l1_ratio=1, C=1, penalty="elasticnet", max_iter=10000,
                                       random_state=42, tol=.0004),
    #'VotingClassifier': VotingClassifier()
}
```

```
In [ ]: #calculate the score for multible models and find the best estimator:
from sklearn.model_selection import cross_val_score
import numpy as np
results = []
for key in clf.keys():
    full_pipeline = make_pipeline(preprocessing, clf[key])
    score = cross_val_score(full_pipeline, Xtrain, ytrain, scoring="accuracy", cv=3)
    results.append((key, score.mean()*100))
print('models scores:', results)
best_model_idx = np.array(results)[:, 1].argmax()
print('best model:', results[best_model_idx][0], results[best_model_idx][1].round(1))

models scores: [('SGDClassifier', 94.16666666666667), ('KNeighborsClassifier', 95.0), ('DecisionTreeClassifier', 91.66666666666666), ('RandomForestClassifier', 95.83333333333334), ('LinearSVC', 97.5), ('SVC', 95.83333333333334), ('logistic_reg', 97.5)]
best model: LinearSVC 97.5
```

```
In [ ]: #test the Linear svc on test dataset
score = cross_val_score(clf['LinearSVC'], Xtrain, ytrain, scoring="accuracy", cv=3)
score.mean()
```

```
Out[ ]: 0.975
```

```
In [ ]: score = cross_val_score(clf['logistic_reg'], Xtrain, ytrain, scoring="accuracy", cv=3)
score.mean()
```

```
Out[ ]: 0.975
```

```
In [ ]: #try the voting classifier model for the highest 3 models :
eclf = VotingClassifier([("lsvc", clf['LinearSVC']),
                        ("log_reg", clf['logistic_reg'])], voting='hard')# , weights=[1,2,1])
eclf.fit(Xtrain, ytrain)
```

```
Out[ ]: 
    ▶ VotingClassifier
        lsvc          log_reg
        ▶ LinearSVC  ▶ LogisticRegression
```

```
In [ ]: eclf.get_params()
```

```
Out[ ]: {'estimators': [('lsvc',
    LinearSVC(C=1, class_weight='balanced', max_iter=30000,
                multi_class='crammer_singer', random_state=42, tol=0.001)),
   ('log_reg',
    LogisticRegression(C=1, l1_ratio=1, max_iter=1000, penalty='elasticnet',
                        random_state=42, solver='saga', tol=0.0004))],
  'flatten_transform': True,
  'n_jobs': None,
  'verbose': False,
  'voting': 'hard',
  'weights': None,
  'lsvc': LinearSVC(C=1, class_weight='balanced', max_iter=30000,
                      multi_class='crammer_singer', random_state=42, tol=0.001),
  'log_reg': LogisticRegression(C=1, l1_ratio=1, max_iter=1000, penalty='elasticnet',
                                random_state=42, solver='saga', tol=0.0004),
  'lsvc__C': 1,
  'lsvc__class_weight': 'balanced',
  'lsvc__dual': True,
  'lsvc__fit_intercept': True,
  'lsvc__intercept_scaling': 1,
  'lsvc__loss': 'squared_hinge',
  'lsvc__max_iter': 30000,
  'lsvc__multi_class': 'crammer_singer',
  'lsvc__penalty': 'l2',
  'lsvc__random_state': 42,
  'lsvc__tol': 0.001,
  'lsvc__verbose': 0,
  'log_reg__C': 1,
  'log_reg__class_weight': None,
  'log_reg__dual': False,
  'log_reg__fit_intercept': True,
  'log_reg__intercept_scaling': 1,
  'log_reg__l1_ratio': 1,
  'log_reg__max_iter': 1000,
  'log_reg__multi_class': 'auto',
  'log_reg__n_jobs': None,
  'log_reg__penalty': 'elasticnet',
  'log_reg__random_state': 42,
  'log_reg__solver': 'saga',
  'log_reg__tol': 0.0004,
  'log_reg__verbose': 0,
  'log_reg__warm_start': False}
```

```
In [ ]: #find the score(accuracy)
score=cross_val_score(eclf,Xtrain,ytrain,scoring='accuracy',cv=3)
accuracy=score.mean()
print('accuracy percentage:',accuracy)
```

```
accuracy percentage: 0.975
```

```
In [ ]: #find confusuin matrix for the three classes
```

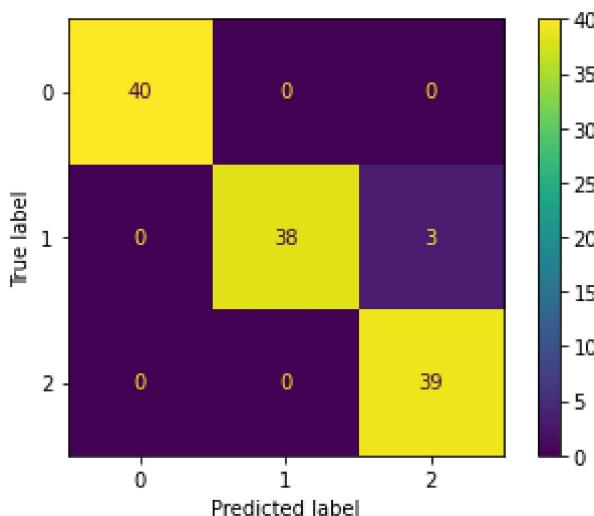
```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix

ytrain_pred = cross_val_predict(eclf, Xtrain, ytrain, cv=3)
cm = confusion_matrix(ytrain, ytrain_pred)
cm
```

Out[]: array([[40, 0, 0],
 [0, 38, 3],
 [0, 0, 39]], dtype=int64)

```
In [ ]: #plot confusuin matrix for the three classes
from sklearn.metrics import ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_predictions(ytrain, ytrain_pred)
plt.show()
```



```
In [ ]: #precision
from sklearn.metrics import precision_score, recall_score

precision_score(ytrain, ytrain_pred, average='macro')
```

Out[]: 0.9761904761904763

```
In [ ]: #recall
recall_score(ytrain, ytrain_pred, average='macro')
```

Out[]: 0.975609756097561

```
In [ ]: #find f1score:
from sklearn.metrics import f1_score

f1_score(ytrain, ytrain_pred, average='macro')
```

Out[]: 0.9749960931395533

```
In [ ]: #find the Classification report and scoring details for each class:
from sklearn.metrics import classification_report, confusion_matrix, multilabel_confusion_matrix
import pandas as pd
```

```
report=classification_report(ytrain, ytrain_pred,target_names=data.target_names,output_df = pd.DataFrame.from_dict(report)
df
```

Out[]:

	setosa	versicolor	virginica	accuracy	macro avg	weighted avg
precision	1.0	1.000000	0.928571	0.975	0.976190	0.976786
recall	1.0	0.926829	1.000000	0.975	0.975610	0.975000
f1-score	1.0	0.962025	0.962963	0.975	0.974996	0.974988
support	40.0	41.000000	39.000000	0.975	120.000000	120.000000

In []:

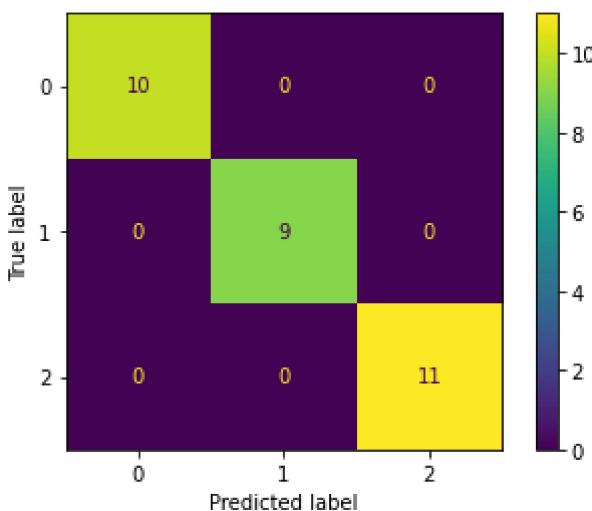
```
#test the model on dataset:
from sklearn.metrics import accuracy_score
#score=cross_val_score(eclf,Xtest,ytest,scoring='accuracy',cv=3)
ytest_pred = eclf.predict(Xtest)

score=accuracy_score(ytest,ytest_pred)
accuracy=score.mean()*100
print('accuracy percentage:',accuracy)
```

accuracy percentage: 100.0

In []:

```
#plot confusuin matrix for the three classes
from sklearn.metrics import ConfusionMatrixDisplay
#ytest_pred = cross_val_predict(eclf, Xtest, ytest, cv=3)
cm = confusion_matrix(ytest, ytest_pred)
cm
ConfusionMatrixDisplay.from_predictions(ytest, ytest_pred)
plt.show()
```



In []:

```
report=classification_report(ytest, ytest_pred,target_names=data.target_names,output_df = pd.DataFrame.from_dict(report)
df
```

	setosa	versicolor	virginica	accuracy	macro avg	weighted avg
precision	1.0	1.0	1.0	1.0	1.0	1.0
recall	1.0	1.0	1.0	1.0	1.0	1.0
f1-score	1.0	1.0	1.0	1.0	1.0	1.0
support	10.0	9.0	11.0	1.0	30.0	30.0

```
In [ ]: #train the best model in the whole data and Save the final Model:
import joblib
```

```
final_model=eclf.fit(features,target)

joblib.dump(final_model,'clf_final_model.pkl')
```

```
Out[ ]: ['clf_final_model.pkl']
```

```
In [ ]: #Load the model for prediction
final_model=joblib.load('clf_final_model.pkl')
new_data=features.iloc[:5]
predictions=final_model.predict(new_data)
predictions
```

```
Out[ ]: array([0, 0, 0, 0, 0])
```

```
In [ ]: target.iloc[:5]
```

```
Out[ ]: 0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int32
```

```
In [ ]: #perfect!! :)
```

```
In [ ]:
```