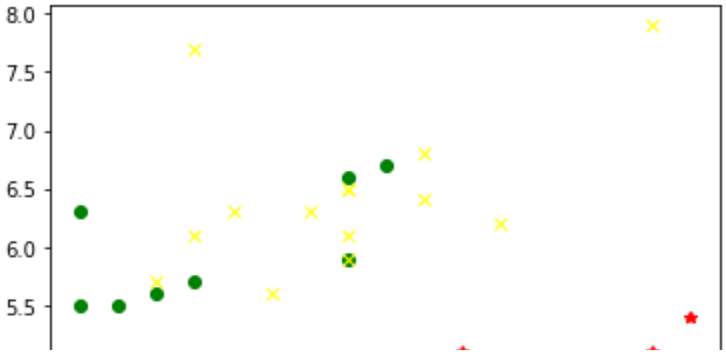
```
#inBuilt Function
import numpy as np
from collections import Counter
#Euclidian Distance
def euclidean_distance(x1, x2):
  return np.sqrt(np.sum((x1 - x2)**2))

class k_nearest_neighbors:
  def __init__(self, k):
    self.k = k

  def knn_fit(self, X_train, y_train):
    self.X_train = X_train
    self.y_train = y_train

  def knn_predict(self, X):
    predicted_lables = [self._predict(x) for x in X]
    #predicted_labels=[]
    #for x in X:
     #   pred=self._predict(x)
      #  predicted_labels.append(pred)
    return np.array(predicted_lables)

  #helper method
  def _predict(self, x):
    #compute distances
    distances = [euclidean_distance(x, x_train) for x_train in self.X_train]
    #get k nearest samples, labels
    k_indices = np.argsort(distances)[:self.k]
    k_nearest_labels = [self.y_train[i] for i in k_indices]
    #majority vote, most common class label
    majority_vote = Counter(k_nearest_labels).most_common(1)
    print("majority vote is",majority_vote)
    return majority_vote[0][0]

#Test KNN on Iris dataset and visualize the results
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
import numpy as np
import matplotlib.pyplot as plt

#User defined module import
#from KNN import k_nearest_neighbors


#Loading dataset
iris_data = datasets.load_iris()
```

```
target = iris_data.target
print(data)
print(target)
# Train/Test splits
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2)
print("traning instances: ",len(X_train))
print("Test instances: ",len(X_test))

#Train KNN model
my_model = k_nearest_neighbors(k = 3)
model=my_model.knn_fit(X_train, y_train)

#predictions = my_model.knn_predict(X_test)

predictions = my_model.knn_predict(X_test)
#Evaluation report
#print("confusion Matrix:")
#print(confusion_matrix(y_test,predictions))
#print("Classification report:", classification_report(y_test, predictions))

#Visulize the predictions
for class_value in range(3):
    row_ix = np.where(predictions== class_value)
    row_px = np.where(y_test== class_value)
  # create scatter of these samples
    if(class_value==0):
        m='*'
        c='red'
    elif(class_value==1):
        m="o"
        c='green'
    elif(class_value==2):
        m='x'
        c='yellow'

    plot1 = plt.figure(1)
    plt.plot(X_test[row_ix, 1], X_test[row_ix, 0],marker=m,color=c)

   # create scatter of these samples
    if(class_value==0):
        m='*'
        c='violet'
    elif(class_value==1):
        m="o"
        c='black'
    elif(class_value==2):
        m='x'
        c='cyan'

    #plt.subplot(1, 2, 2)
    #plt.subplot(1, 2, 2)
    plot2= plt.figure(2)
    plt.plot(X_test[row_px, 1], X_test[row_px, 0],marker=m,color=c)

plt.show()
```

```
[[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]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
 [5.2 3.5 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [4.7 3.2 1.6 0.2]
 [4.8 3.1 1.6 0.2]
 [5.4 3.4 1.5 0.4]
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.2]
 [5.  3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
 [4.9 3.6 1.4 0.1]
 [4.4 3.  1.3 0.2]
 [5.1 3.4 1.5 0.2]
 [5.  3.5 1.3 0.3]
 [4.5 2.3 1.3 0.3]
 [4.4 3.2 1.3 0.2]
 [5.  3.5 1.6 0.6]
 [5.1 3.8 1.9 0.4]
 [4.8 3.  1.4 0.3]
 [5.1 3.8 1.6 0.2]
 [4.6 3.2 1.4 0.2]
 [5.3 3.7 1.5 0.2]
```

```
[5.3 3.7 1.5 0.2]
[5.  3.3 1.4 0.2]
[7.  3.2 4.7 1.4]
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4.  1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
[4.9 2.4 3.3 1. ]
[6.6 2.9 4.6 1.3]
[5.2 2.7 3.9 1.4]
[5.  2.  3.5 1. ]
[5.9 3.  4.2 1.5]
[6.  2.2 4.  1. ]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3.  4.5 1.5]
[5.8 2.7 4.1 1. ]
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
[5.9 3.2 4.8 1.8]
[6.1 2.8 4.  1.3]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3.  4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3.  5.  1.7]
[6.  2.9 4.5 1.5]
[5.7 2.6 3.5 1. ]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1. ]
[5.8 2.7 3.9 1.2]
[6.  2.7 5.1 1.6]
[5.4 3.  4.5 1.5]
[6.  3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3.  4.1 1.3]
[5.5 2.5 4.  1.3]
[5.5 2.6 4.4 1.2]
[6.1 3.  4.6 1.4]
[5.8 2.6 4.  1.2]
[5.  2.3 3.3 1. ]
[5.6 2.7 4.2 1.3]
[5.7 3.  4.2 1.2]
[5.7 2.9 4.2 1.3]
[6.2 2.9 4.3 1.3]
[5.1 2.5 3.  1.1]
[5.7 2.8 4.1 1.3]
[6.3 3.3 6.  2.5]
[5.8 2.7 5.1 1.9]
[7.1 3.  5.9 2.1]
[6.3 2.9 5.6 1.8]
```

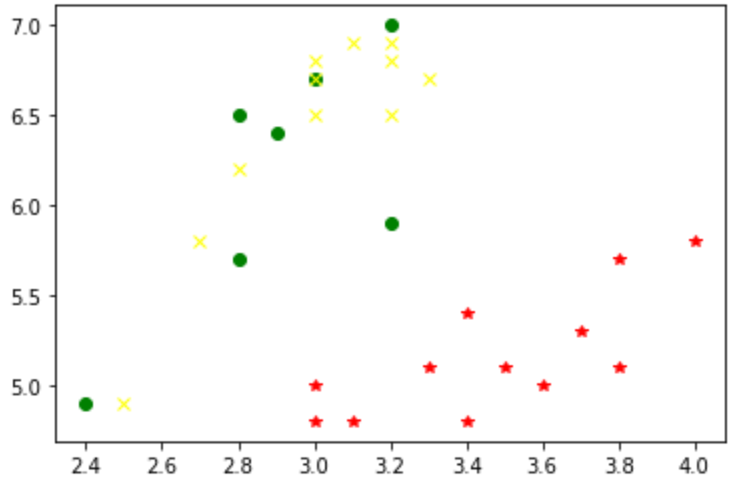
```
 [6.3 2.9 5.6 1.8]
 [6.5 3.  5.8 2.2]
 [7.6 3.  6.6 2.1]
 [4.9 2.5 4.5 1.7]
 [7.3 2.9 6.3 1.8]
 [6.7 2.5 5.8 1.8]
 [7.2 3.6 6.1 2.5]
 [6.5 3.2 5.1 2. ]
 [6.4 2.7 5.3 1.9]
 [6.8 3.  5.5 2.1]
 [5.7 2.5 5.  2. ]
 [5.8 2.8 5.1 2.4]
 [6.4 3.2 5.3 2.3]
 [6.5 3.  5.5 1.8]
 [7.7 3.8 6.7 2.2]
 [7.7 2.6 6.9 2.3]
 [6.  2.2 5.  1.5]
 [6.9 3.2 5.7 2.3]
 [5.6 2.8 4.9 2. ]
 [7.7 2.8 6.7 2. ]
 [6.3 2.7 4.9 1.8]
 [6.7 3.3 5.7 2.1]
 [7.2 3.2 6.  1.8]
 [6.2 2.8 4.8 1.8]
 [6.1 3.  4.9 1.8]
 [6.4 2.8 5.6 2.1]
 [7.2 3.  5.8 1.6]
 [7.4 2.8 6.1 1.9]
 [7.9 3.8 6.4 2. ]
 [6.4 2.8 5.6 2.2]
 [6.3 2.8 5.1 1.5]
 [6.1 2.6 5.6 1.4]
 [7.7 3.  6.1 2.3]
 [6.3 3.4 5.6 2.4]
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
[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 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 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
 2 2]
traning instances:  120
Test instances:  30
majority vote is [(2, 3)]
majority vote is [(0, 3)]
majority vote is [(2, 3)]
majority vote is [(0, 3)]
majority vote is [(2, 2)]
majority vote is [(1, 3)]
majority vote is [(0, 3)]
majority vote is [(0, 3)]
majority vote is [(2, 3)]
majority vote is [(2, 3)]
majority vote is [(2, 3)]
majority vote is [(1, 3)]
majority vote is [(0, 3)]
majority vote is [(2, 3)]
majority vote is [(2, 3)]
majority vote is [(2, 3)]
majority vote is [(2, 3)]
majority vote is [(2, 3)]
majority vote is [(2, 2)]
majority vote is [(1, 3)]
majority vote is [(1, 3)]
majority vote is [(1, 3)]
majority vote is [(2, 2)]
majority vote is [(0, 3)]
majority vote is [(1, 3)]
majority vote is [(0, 3)]
majority vote is [(1, 3)]
majority vote is [(2, 3)]
majority vote is [(0, 3)]
majority vote is [(1, 3)]
```

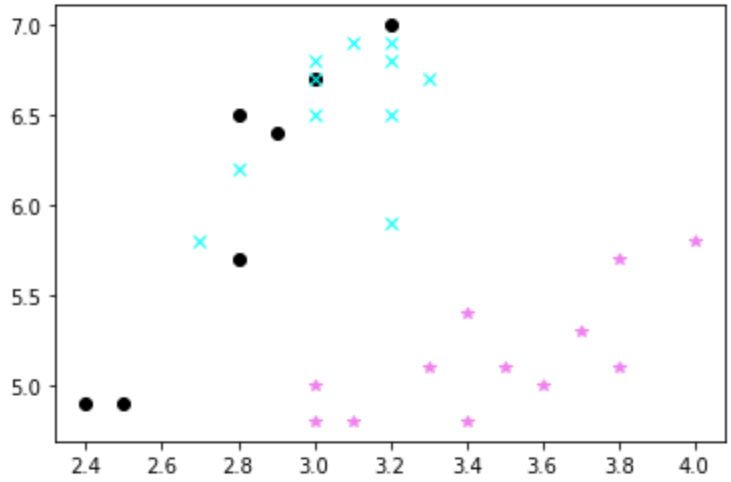


```
#Using Library Functions
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
import numpy as np
import matplotlib.pyplot as plt

#Loading dataset
iris_data = datasets.load_iris()
data = iris_data.data
target = iris_data.target

# Train/Test splits
```

```
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2)
print("traning instances: ",len(X_train))
print("Test instances: ",len(X_test))

#Train KNN model
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
predictions = knn.predict(X_test)
#Evaluation report
print("confusion Matrix:")
print(confusion_matrix(y_test,predictions))
print("Classification report:", classification_report(y_test, predictions))

#Visulize the predictions
for class_value in range(3):
    row_px = np.where(predictions== class_value)
    row_ix = np.where(y_test== class_value)
    # create scatter of these samples
    if(class_value==0):
        m='*'
        c='red'
    elif(class_value==1):
        m="o"
        c='green'
    elif(class_value==2):
        m='x'
        c='yellow'

    plot1 = plt.figure(1)
    plt.plot(X_test[row_ix, 1], X_test[row_ix, 0],marker=m,color=c)

   # create scatter of these samples
    if(class_value==0):
        m='*'
        c='violet'
    elif(class_value==1):
        m="o"
        c='black'
    elif(class_value==2):
        m='x'
        c='cyan'

    #plt.subplot(1, 2, 2)
    plot2= plt.figure(2)
    plt.plot(X_test[row_px, 1], X_test[row_px, 0],marker=m,color=c)

plt.show()
print(knn.score(X_test, y_test))
print("xaxis predicted",X_test[row_px, 1])
print("yaxis predicted",X_test[row_px, 0])
print("xaxis inputted ",X_test[row_ix, 1])
print("yaxis inputted",X_test[row_ix, 0])
```

```
traning instances:  120
Test instances:  30
confusion Matrix:
[[12  0  0]
 [ 0  6  1]
 [ 0  1 10]]
Classification report:               precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.86      0.86      0.86         7
           2       0.91      0.91      0.91        11

    accuracy                           0.93        30
   macro avg       0.92      0.92      0.92        30
weighted avg       0.93      0.93      0.93        30
```





```
0.9333333333333333
xaxis predicted [[3.1 3.  3.2 2.8 3.  3.2 3.2 2.7 3.2 3.3 3. ]]
yaxis predicted [[6.9 6.5 5.9 6.2 6.8 6.5 6.8 5.8 6.9 6.7 6.7]]
xaxis inputted  [[3.1 3.  2.8 2.5 3.  3.2 3.2 2.7 3.2 3.3 3. ]]
```

```
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import classification_report,confusion_matrix
import numpy as np
import matplotlib.pyplot as plt

#User defined module import
#from KNN import k_nearest_neighbors



#Loading dataset
iris_data = datasets.load_iris()
data = iris_data.data
target = iris_data.target
print(data)
print(target)
```

```
[[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]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
 [5.2 3.5 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [4.7 3.2 1.6 0.2]
 [4.8 3.1 1.6 0.2]
 [5.4 3.4 1.5 0.4]
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.2]
 [5.  3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
 [4.9 3.6 1.4 0.1]
 [4.4 3.  1.3 0.2]
 [5.1 3.4 1.5 0.2]
 [5.1 3.4 1.5 0.2]
 [5.  3.5 1.3 0.3]
 [4.5 2.3 1.3 0.3]
 [4.4 3.2 1.3 0.2]
 [5.  3.5 1.6 0.6]
 [5.1 3.8 1.9 0.4]
 [4.8 3.  1.4 0.3]
 [5.1 3.8 1.6 0.2]
 [4.6 3.2 1.4 0.2]
 [5.3 3.7 1.5 0.2]
 [5.  3.3 1.4 0.2]
 [7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
 [6.9 3.1 4.9 1.5]
 [5.5 2.3 4.  1.3]
 [6.5 2.8 4.6 1.5]
 [5.7 2.8 4.5 1.3]
 [6.3 3.3 4.7 1.6]
 [4.9 2.4 3.3 1. ]
```