In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accura
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
```

In [19]:
```
iris =pd.read_csv("iris.csv")
iris
```

Out[19]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

In [22]:
```
X = iris.data
X
```

Out[22]:
```
array([[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],
```

In [4]:
```
y = iris.target
y
```

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

In [5]:
```
df = pd.DataFrame(data=np.c_[X, y], columns=iris.feature_names + ['target']
df
```

Out[5]:

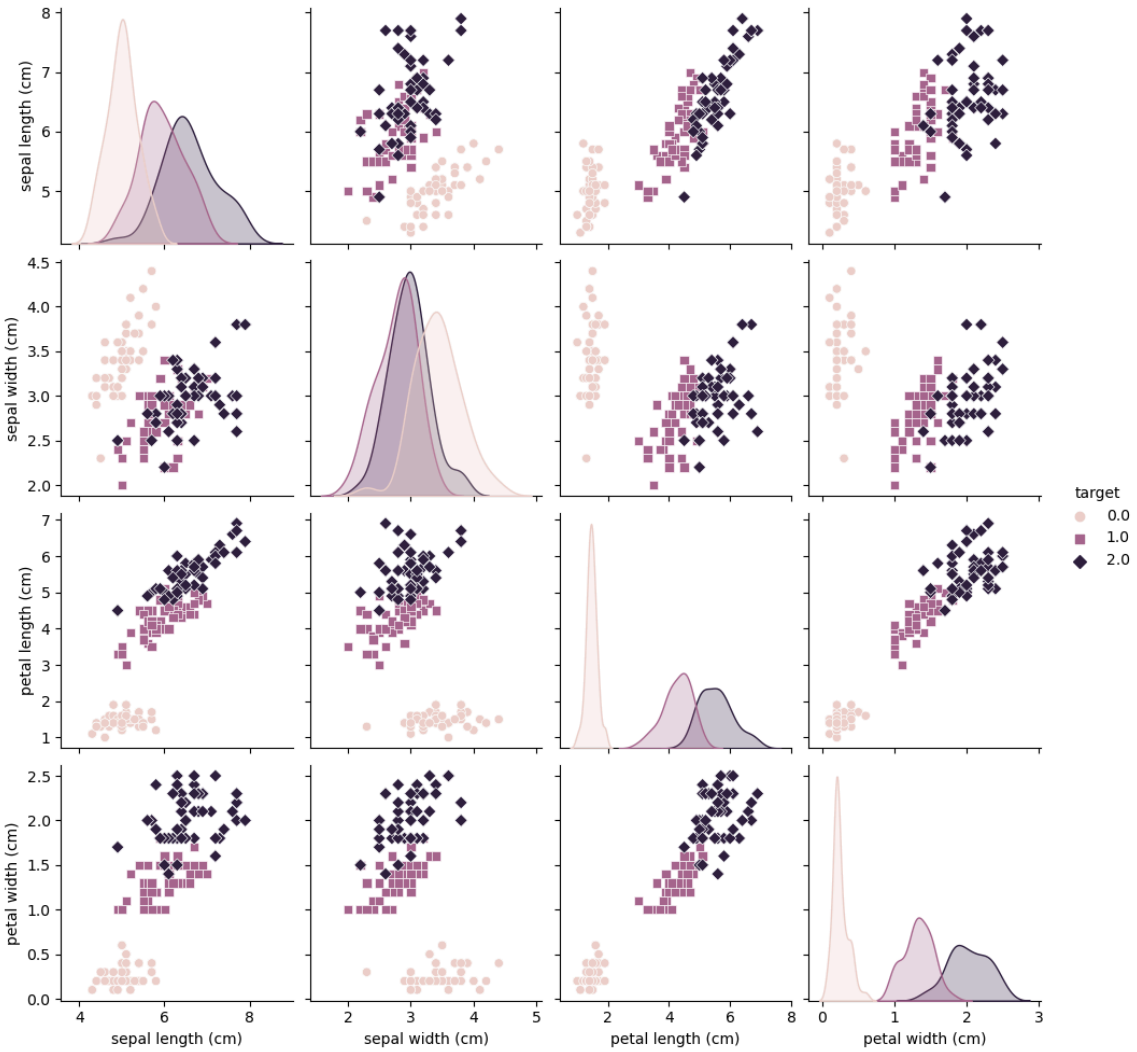
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2.0
146	6.3	2.5	5.0	1.9	2.0
147	6.5	3.0	5.2	2.0	2.0
148	6.2	3.4	5.4	2.3	2.0
149	5.9	3.0	5.1	1.8	2.0

150 rows × 5 columns

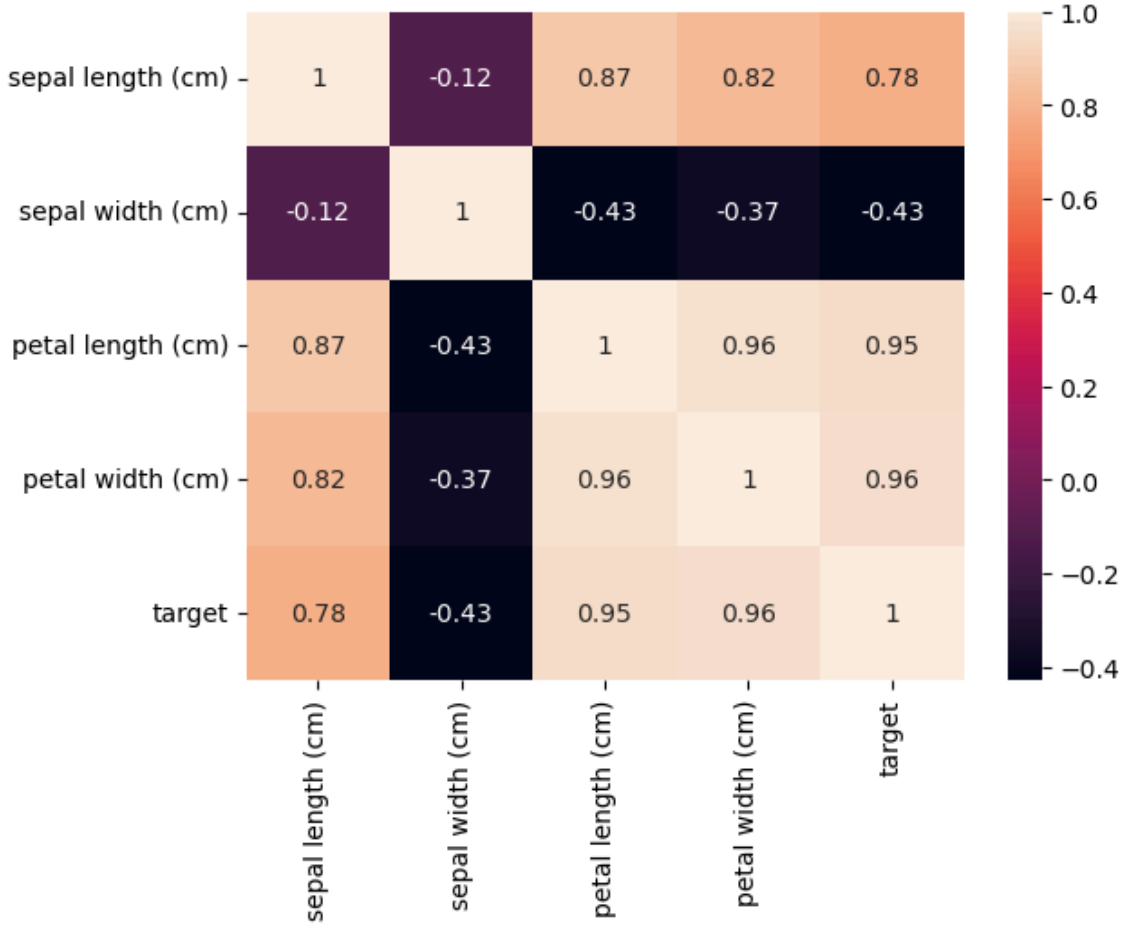
In [6]:
```
print(df.head())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (c
m)  \
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

   target
0     0.0
1     0.0
2     0.0
3     0.0
4     0.0
```

In [7]:
```
sns.pairplot(df, hue='target', markers=["o", "s", "D"])
plt.show()
```



In [8]:
```
sns.heatmap(df.corr(), annot=True)
plt.show()
```



In [9]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
X_train, X_test, y_train, y_test
```

Out[9]:
```
(array([[4.6, 3.6, 1. , 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [6.7, 3.1, 4.4, 1.4],
        [4.8, 3.4, 1.6, 0.2],
        [4.4, 3.2, 1.3, 0.2],
        [6.3, 2.5, 5. , 1.9],
        [6.4, 3.2, 4.5, 1.5],
        [5.2, 3.5, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.2, 4.1, 1.5, 0.1],
        [5.8, 2.7, 5.1, 1.9],
        [6. , 3.4, 4.5, 1.6],
        [6.7, 3.1, 4.7, 1.5],
        [5.4, 3.9, 1.3, 0.4],
        [5.4, 3.7, 1.5, 0.2],
        [5.5, 2.4, 3.7, 1. ],
        [6.3, 2.8, 5.1, 1.5],
        [6.4, 3.1, 5.5, 1.8],
        [6.6, 3. , 4.4, 1.4],
```

In [10]:
```
scaler = StandardScaler()
scaler
```

Out[10]:
```
StandardScaler
StandardScaler()
```

In [11]:
```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [12]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

Out[12]:
```
KNeighborsClassifier
KNeighborsClassifier()
```

In [13]:
```
y_pred = knn.predict(X_test)
y_pred
```

Out[13]:
```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0])
```

In [14]:
```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

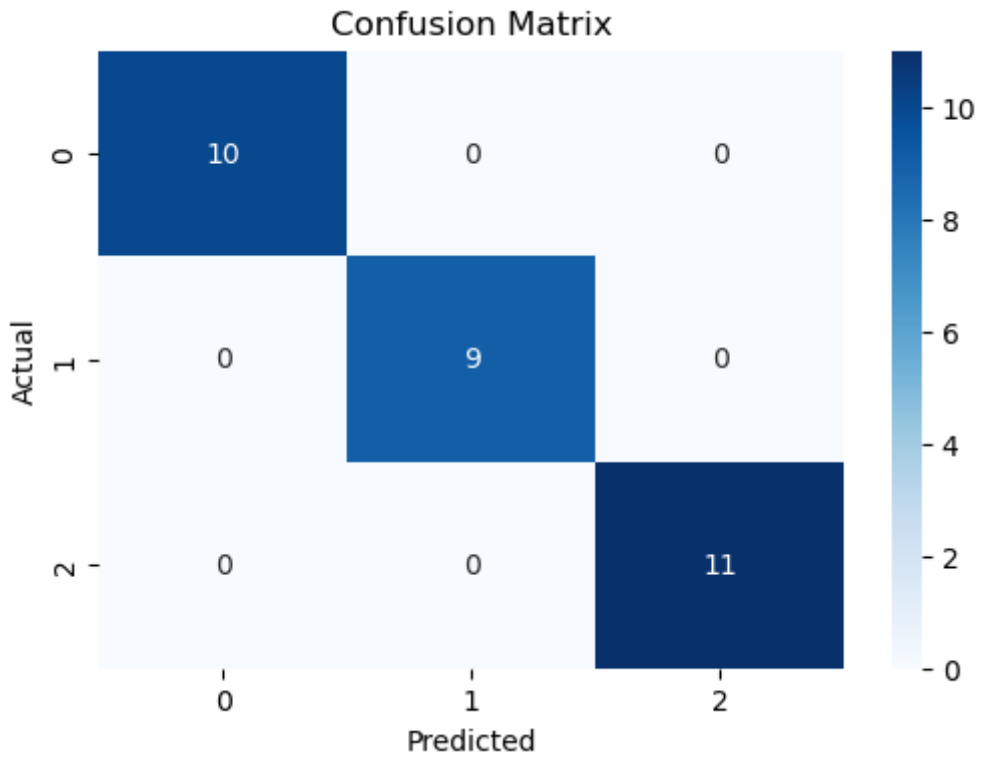
In [15]:
```
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

In [16]:
```
print("\nAccuracy Score:")
print(accuracy_score(y_test, y_pred))
```

```
Accuracy Score:
1.0
```

In [17]:
```
plt.figure(figsize=(6,4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Bl
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



In [29]:
```
from matplotlib.colors import ListedColormap

def plot_decision_boundaries(X, y, classifier, resolution=0.02):
    # Setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # Plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # Plot class samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0],
                    y=X[y == cl, 1],
                    alpha=0.8,
                    c=colors[idx],
                    marker=markers[idx],
                    label=cl,
                    edgecolor='black')
```

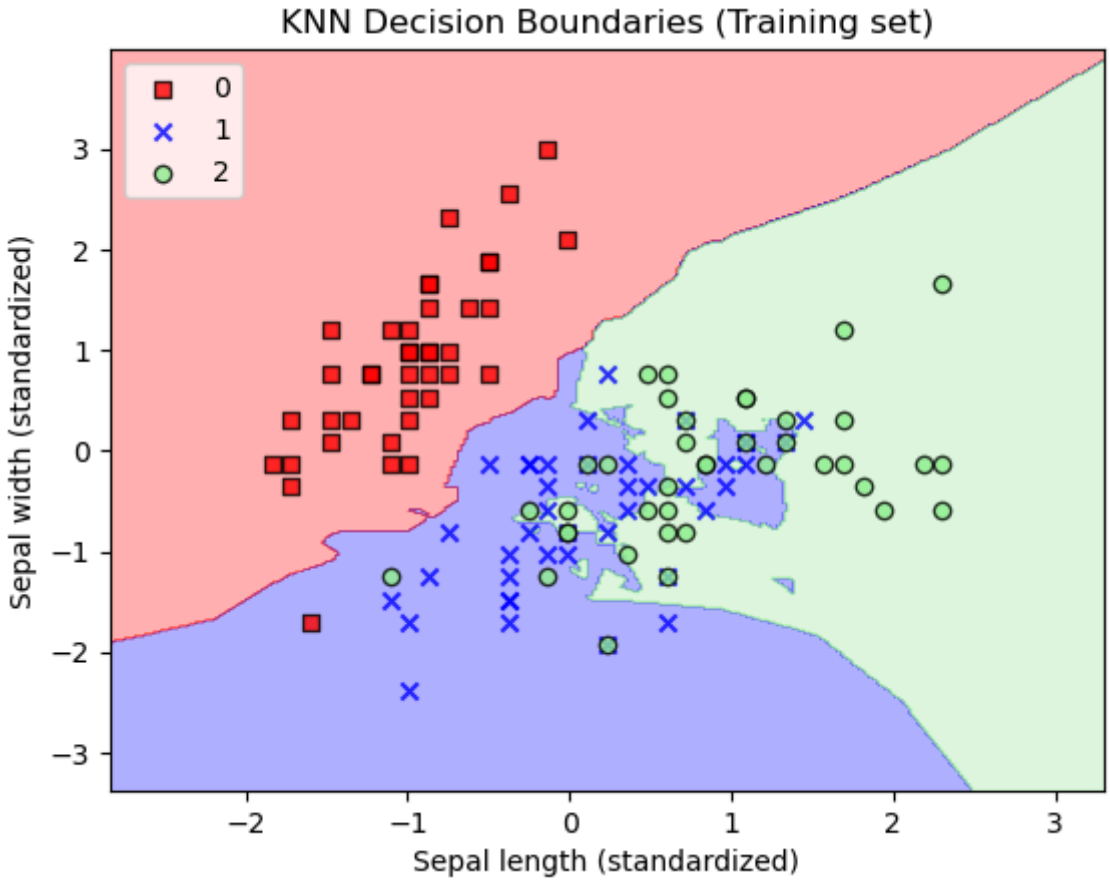
In [30]:
```
X_train_2D = X_train[:, :2]
X_test_2D = X_test[:, :2]
```

In [31]:
```
knn_2D = KNeighborsClassifier(n_neighbors=5)
knn_2D.fit(X_train_2D, y_train)
```

Out[31]:
```
KNeighborsClassifier
KNeighborsClassifier()
```

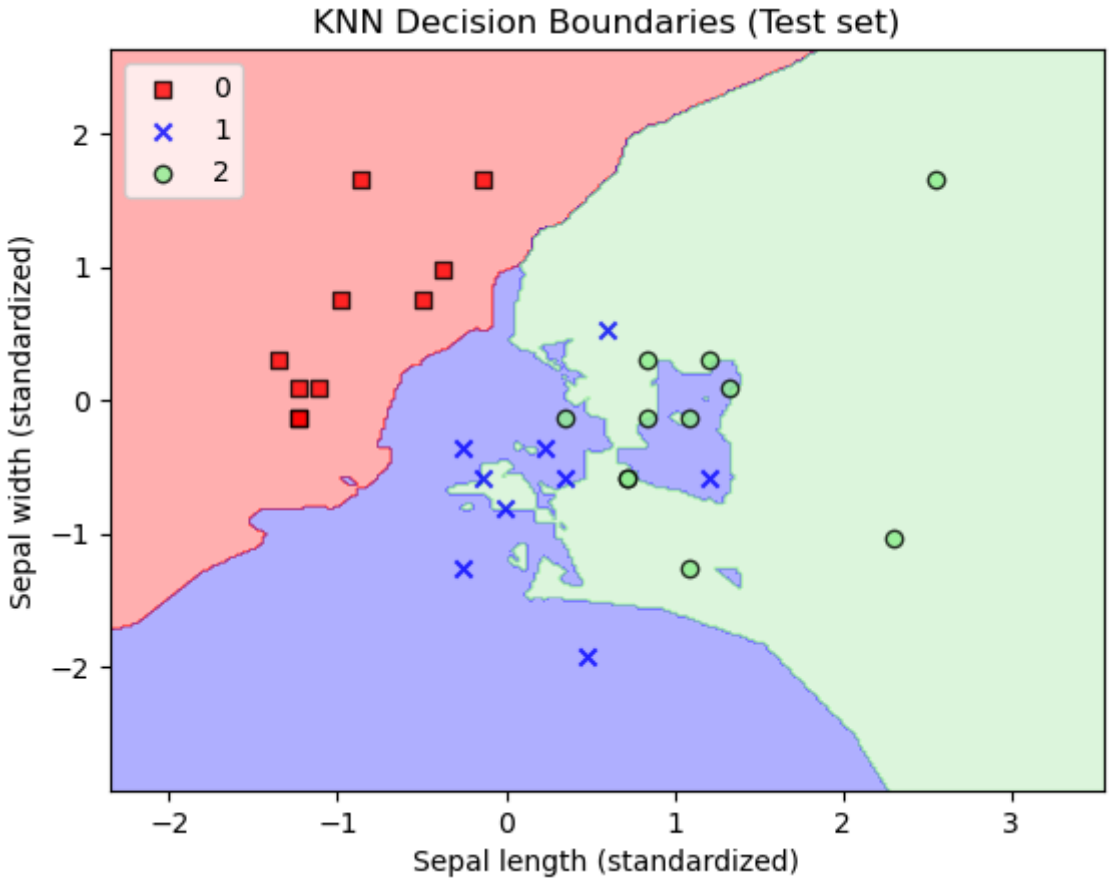
In [32]:
```
plot_decision_boundaries(X_train_2D, y_train, classifier=knn_2D)
plt.xlabel('Sepal length (standardized)')
plt.ylabel('Sepal width (standardized)')
plt.legend(loc='upper left')
plt.title('KNN Decision Boundaries (Training set)')
plt.show()
```

```
C:\Users\Aditya Kudva\AppData\Local\Temp\ipykernel_22968\347347175.py:22:
UserWarning: You passed a edgecolor/edgecolors ('black') for an unfilled m
arker ('x').  Matplotlib is ignoring the edgecolor in favor of the facecol
or.  This behavior may change in the future.
  plt.scatter(x=X[y == cl, 0],
```



In [33]:
```
plot_decision_boundaries(X_test_2D, y_test, classifier=knn_2D)
plt.xlabel('Sepal length (standardized)')
plt.ylabel('Sepal width (standardized)')
plt.legend(loc='upper left')
plt.title('KNN Decision Boundaries (Test set)')
plt.show()
```

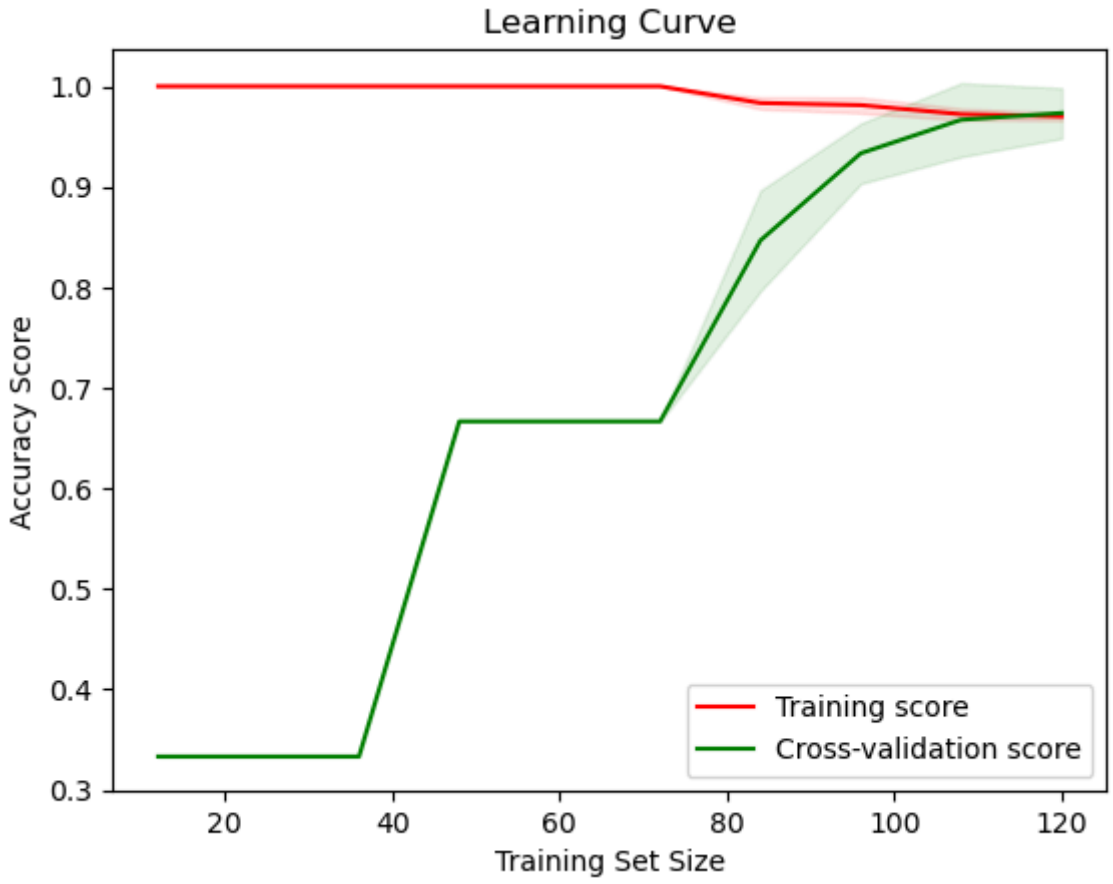
```
C:\Users\Aditya Kudva\AppData\Local\Temp\ipykernel_22968\347347175.py:22:
UserWarning: You passed a edgecolor/edgecolors ('black') for an unfilled m
arker ('x').  Matplotlib is ignoring the edgecolor in favor of the facecol
or.  This behavior may change in the future.
  plt.scatter(x=X[y == cl, 0],
```



In [34]:
```
from sklearn.model_selection import cross_val_score

def evaluate_features(X, y):
    feature_names = iris.feature_names
    for i in range(X.shape[1]):
        X_new = np.delete(X, i, axis=1)
        scores = cross_val_score(KNeighborsClassifier(n_neighbors=5), X_new,
        print(f"Feature removed: {feature_names[i]}, Mean accuracy: {scores

evaluate_features(X, y)
```

```
Feature removed: sepal length (cm), Mean accuracy: 0.9667
Feature removed: sepal width (cm), Mean accuracy: 0.9667
Feature removed: petal length (cm), Mean accuracy: 0.9600
Feature removed: petal width (cm), Mean accuracy: 0.9467
```

In [35]:
```
from sklearn.model_selection import learning_curve

train_sizes, train_scores, test_scores = learning_curve(knn, X, y, cv=5,
                                                        train_sizes=np.linsp

train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

plt.plot(train_sizes, train_mean, label="Training score", color="r")
plt.plot(train_sizes, test_mean, label="Cross-validation score", color="g")

plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_st
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, c

plt.title("Learning Curve")
plt.xlabel("Training Set Size")
plt.ylabel("Accuracy Score")
plt.legend(loc="best")
plt.show()
```



In [36]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {'n_neighbors': np.arange(1, 31)}
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)
grid_search.fit(X_train, y_train)

print(f"Best parameters: {grid_search.best_params_}")
print(f"Best cross-validation score: {grid_search.best_score_:.4f}")
```

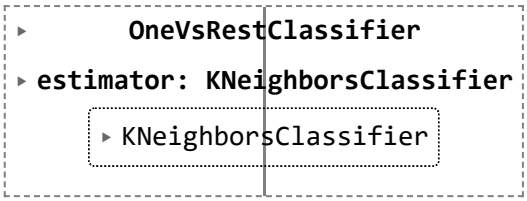
```
Best parameters: {'n_neighbors': 3}
Best cross-validation score: 0.9500
```

In [37]:
```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from scipy import interp
from itertools import cycle
```

In [42]:
```
y_test_bin = label_binarize(y_test, classes=[0, 1, 2])
n_classes = y_test_bin.shape[1]
```

In [43]:
```
classifier = OneVsRestClassifier(KNeighborsClassifier(n_neighbors=5))
classifier
```

Out[43]:



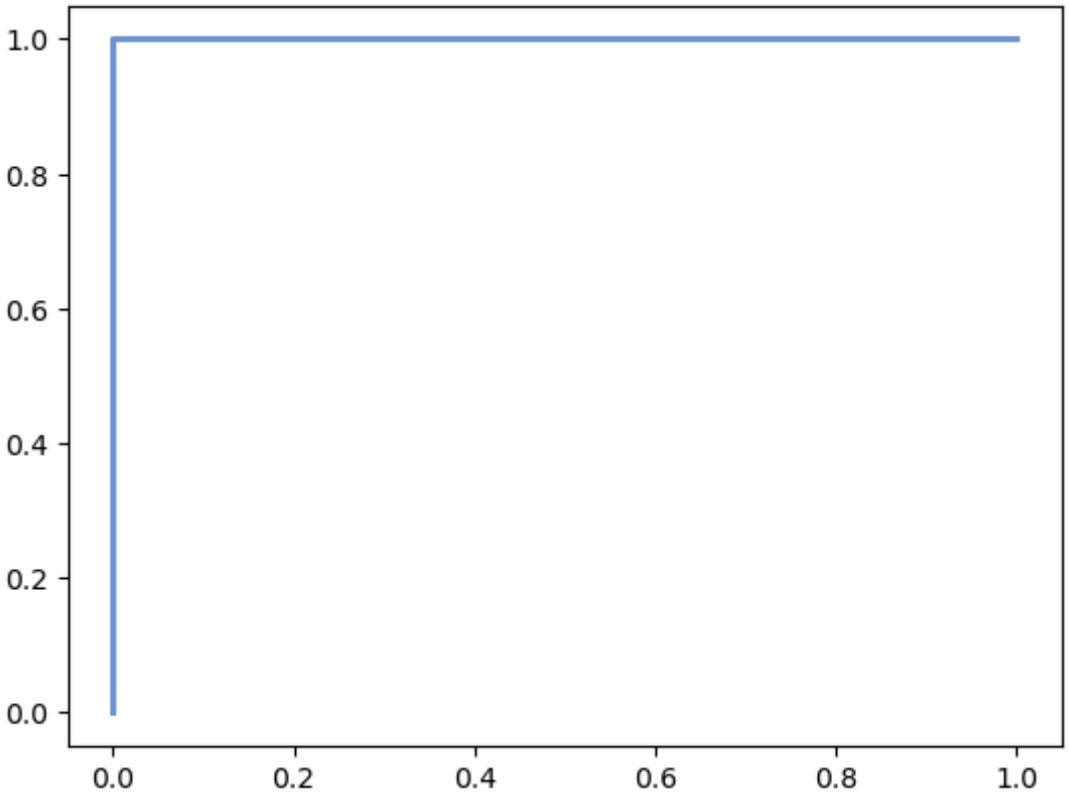
In [44]:
```
y_score = classifier.fit(X_train, y_train).predict_proba(X_test)
y_score
```

Out[44]:
```
array([[0. , 0.8, 0.2],
       [1. , 0. , 0. ],
       [0. , 0. , 1. ],
       [0. , 1. , 0. ],
       [0. , 1. , 0. ],
       [1. , 0. , 0. ],
       [0. , 1. , 0. ],
       [0. , 0. , 1. ],
       [0. , 0.6, 0.4],
       [0. , 1. , 0. ],
       [0. , 0. , 1. ],
       [1. , 0. , 0. ],
       [1. , 0. , 0. ],
       [1. , 0. , 0. ],
       [1. , 0. , 0. ],
       [0. , 0.8, 0.2],
       [0. , 0. , 1. ],
       [0. , 1. , 0. ],
       [0. , 1. , 0. ],
       [0. , 0. , 1. ],
       [1. , 0. , 0. ],
       [0. , 0.2, 0.8],
       [1. , 0. , 0. ],
       [0. , 0. , 1. ],
       [0. , 0. , 1. ],
       [0. , 0. , 1. ],
       [0. , 0.2, 0.8],
       [0. , 0. , 1. ],
       [1. , 0. , 0. ],
       [1. , 0. , 0. ]])
```

In [45]:
```
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```

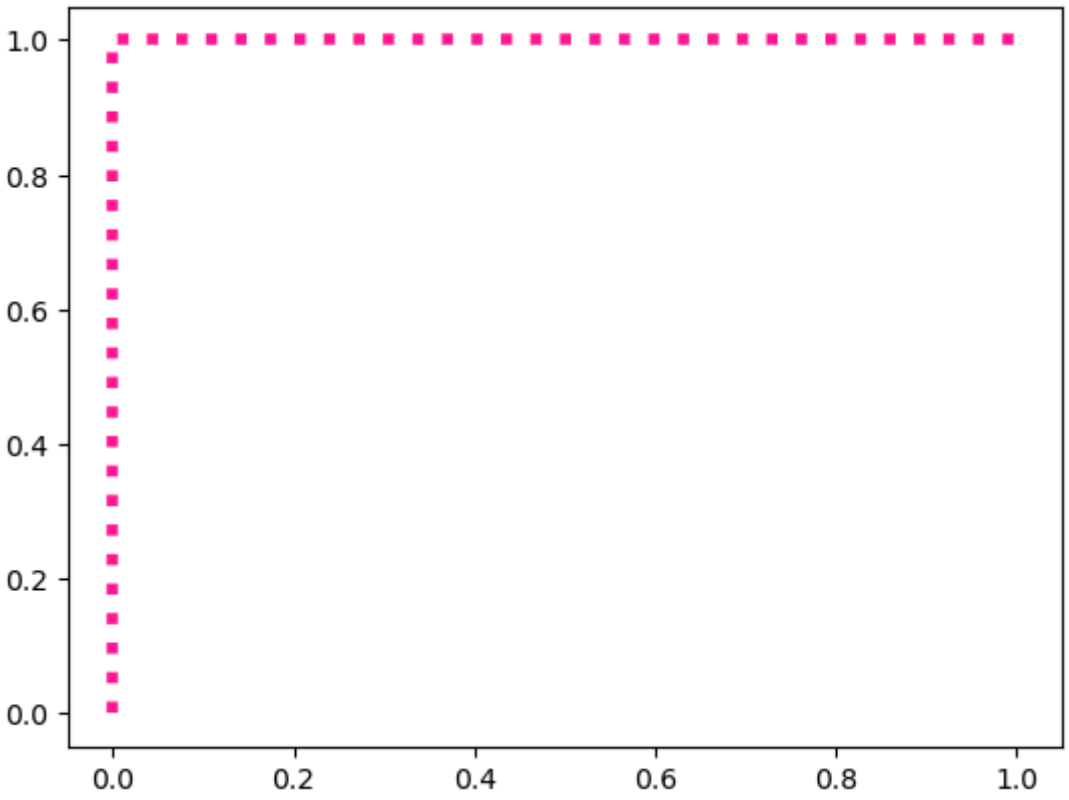
In [46]:
```
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_score.ravel
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
```

In [47]:
```
plt.figure()
colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))
```



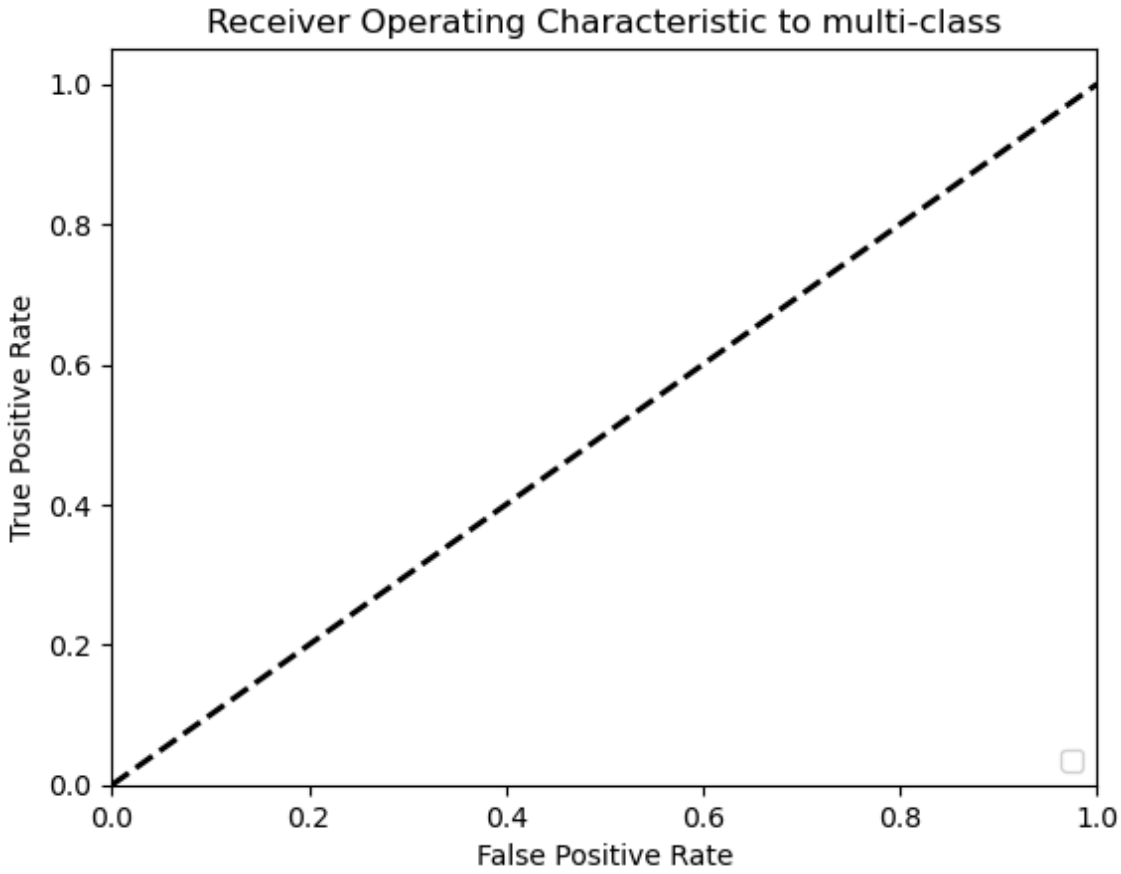
In [48]:
```
plt.plot(fpr["micro"], tpr["micro"], color='deeppink', linestyle=':', linew
         label='micro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["micro"]))
```

Out[48]:
```
[<matplotlib.lines.Line2D at 0x1c9ff455c10>]
```



In [49]:
```
plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic to multi-class')
plt.legend(loc="lower right")
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose la
bel start with an underscore are ignored when legend() is called with no a
rgument.
```



In [ ]: