In [1]:
```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import statsmodels.api as sm
from sklearn.cluster import KMeans
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import classification_report, roc_curve, auc, confusion_matrix, cohen_kappa_score, matthews_corr
from sklearn.preprocessing import StandardScaler, label_binarize, LabelEncoder
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, learning_curve
from sklearn import model_selection
from sklearn import metrics
from yellowbrick.classifier import ClassificationReport
from mlxtend.plotting import plot_decision_regions
from sklearn.decomposition import PCA
from matplotlib.colors import ListedColormap
```

In [2]:
```
df = pd.read_csv(r"C:\Users\abhim\Downloads\Iris.csv")
```

In [3]:
```
df
```

Out[3]:

	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 [4]:
```
encoder = LabelEncoder()
df['Species'] = encoder.fit_transform(df['Species'])
```

In [5]:
```
X = df.drop(columns=['Id', 'Species'])
y = df['Species']
```

In [6]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [7]:
```
param_grid = {'solver': ['svd', 'lsqr', 'eigen'],
              'tol': [0.0001, 0.0002, 0.0003]}

lda = LinearDiscriminantAnalysis()

grid_search = GridSearchCV(estimator=lda, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)
print("Best Parameters: ", grid_search.best_params_)
```

Best Parameters: {'solver': 'svd', 'tol': 0.0001}

In [8]:
```
lda = LinearDiscriminantAnalysis(solver=grid_search.best_params_['solver'], tol=grid_search.best_params_['tol'])
lda.fit(X_train, y_train)
```

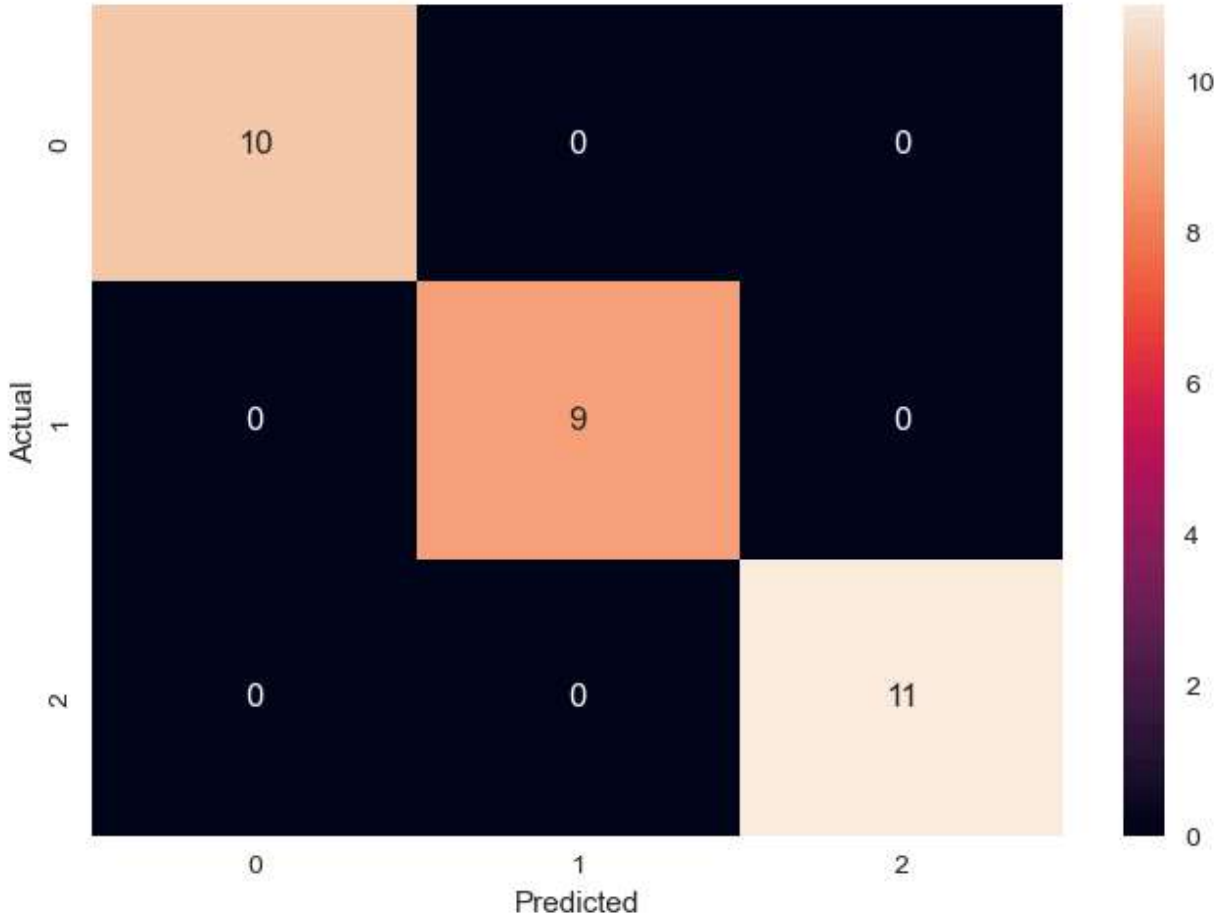
Out[8]:

▾ LinearDiscriminantAnalysis

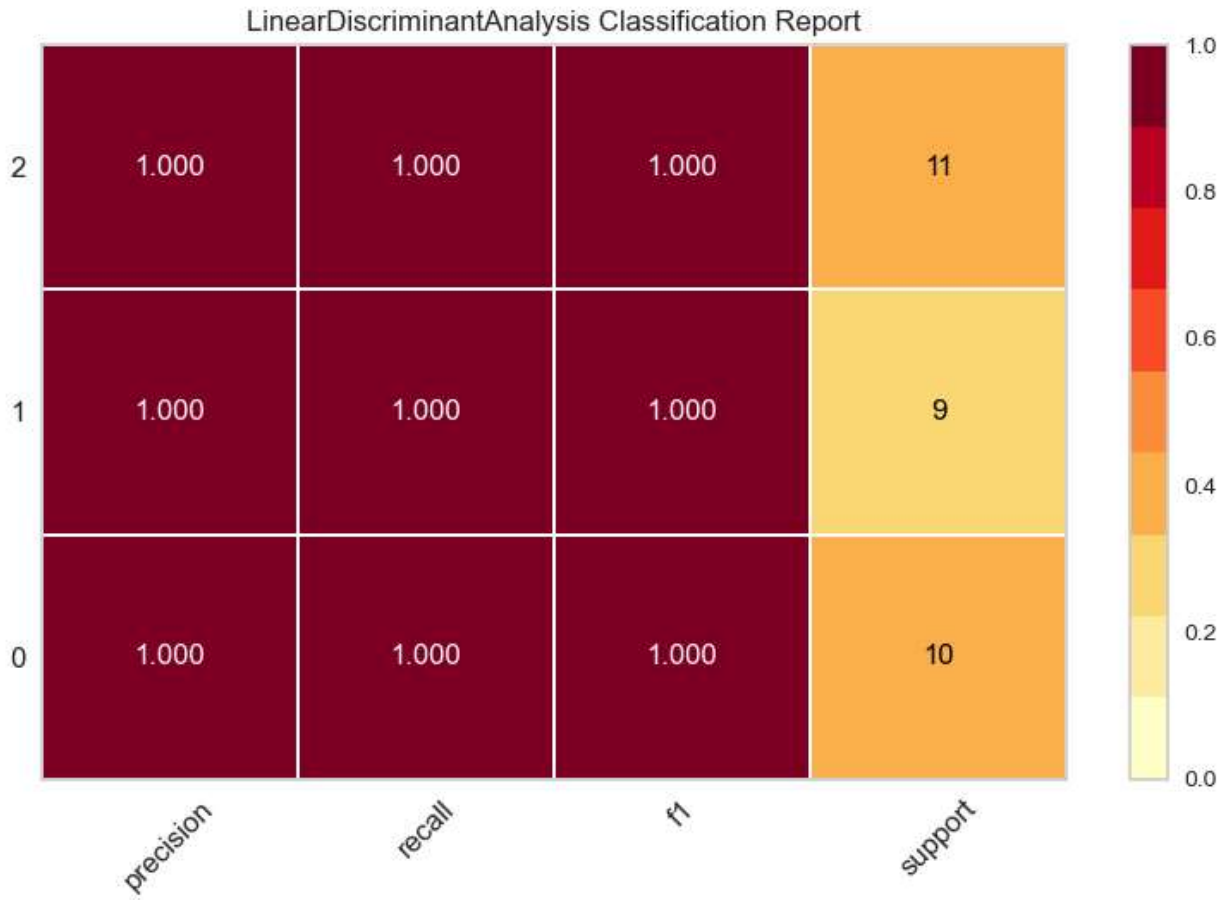
```
LinearDiscriminantAnalysis()
```

In [9]:
```
y_pred = lda.predict(X_test)

# Model evaluation
print(classification_report(y_test, y_pred))
conf_mat = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_mat, annot=True, fmt='d',
            xticklabels=lda.classes_, yticklabels=lda.classes_)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

	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 [10]:
```
visualizer = ClassificationReport(lda, support=True)
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()
```



Out[10]: <Axes: title={'center': 'LinearDiscriminantAnalysis Classification Report'}>

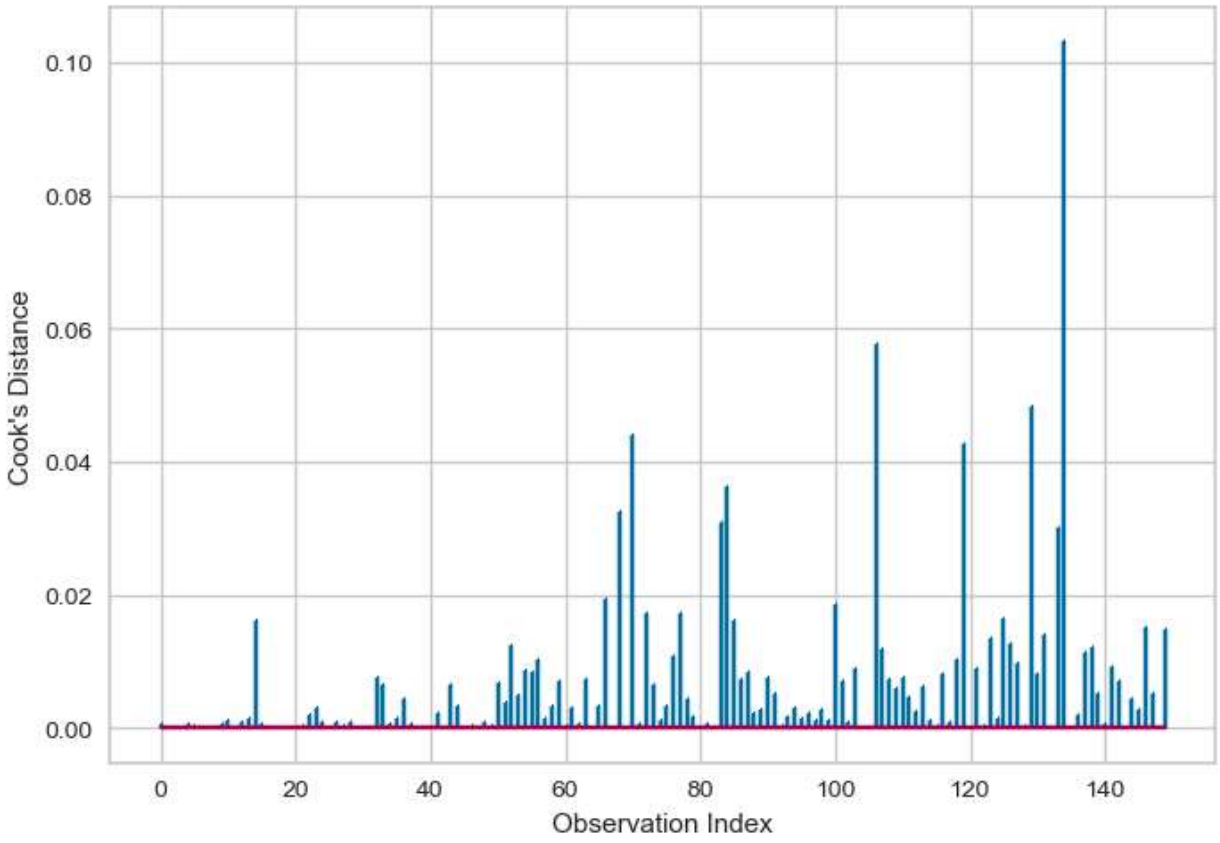
In [11]:
```
#COHEN's KAPPA
kappa = cohen_kappa_score(y_test, y_pred)
print("Cohen's Kappa: ", kappa)

# Matthews Correlation Coefficient (MCC)
mcc = matthews_corrcoef(y_test, y_pred)
print("Matthews Correlation Coefficient: ", mcc)
```

```
Cohen's Kappa:  1.0
Matthews Correlation Coefficient:  1.0
```

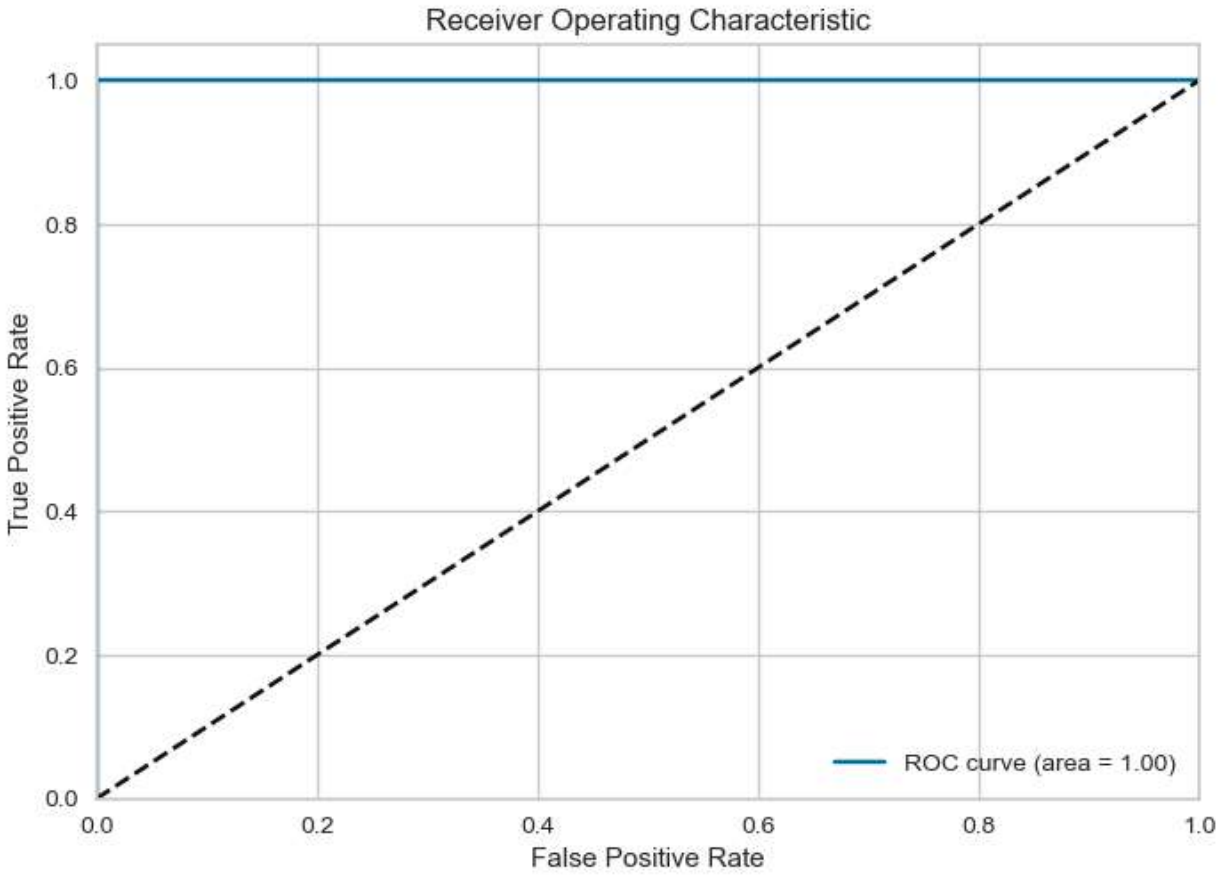
In [12]:
```
cv_scores = cross_val_score(lda, X, y, cv=5)
print('Cross-validation scores: ', cv_scores)
print('Mean cross-validation score: ', np.mean(cv_scores))
```

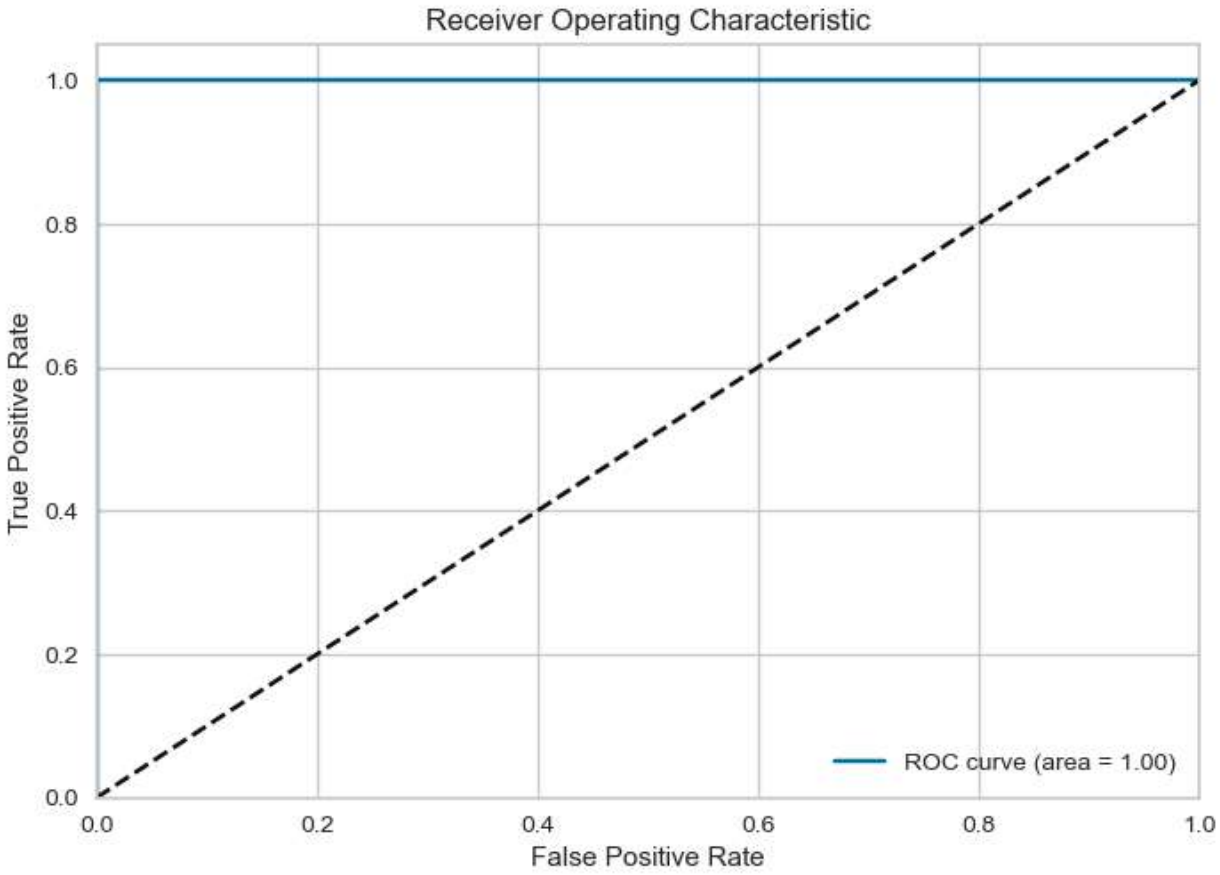
```
Cross-validation scores:  [1.         1.         0.96666667 0.93333333 1.        ]
Mean cross-validation score:  0.9800000000000001
```

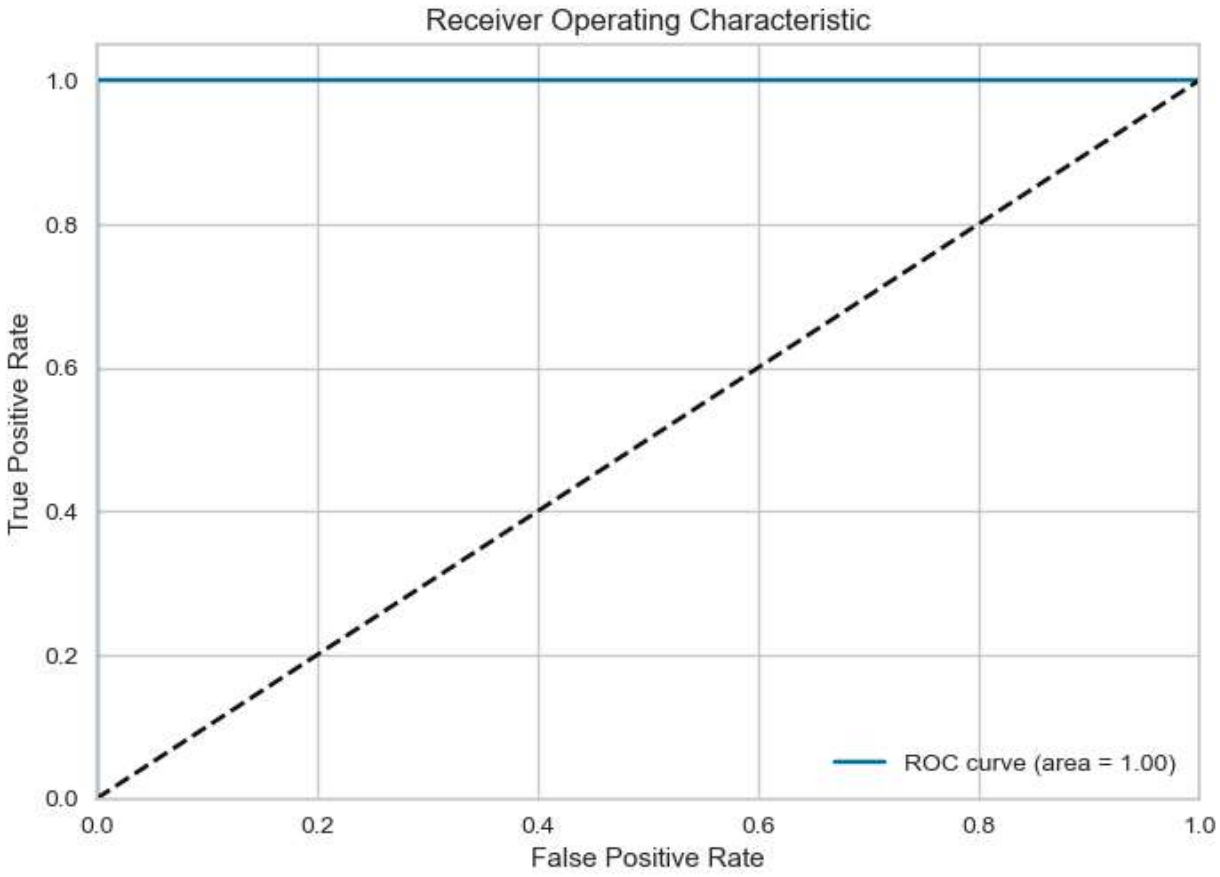
In [13]:
```
#cook's distance
model = sm.OLS(y, sm.add_constant(X))
results = model.fit()
influence = results.get_influence()

(c, p) = influence.cooks_distance
plt.stem(np.arange(len(c)), c, markerfmt=",", use_line_collection=True)
plt.xlabel("Observation Index")
plt.ylabel("Cook's Distance")
plt.show()
```



In [14]:
```
y_test_binarized = label_binarize(y_test, classes=[0, 1, 2])
y_pred_binarized = label_binarize(y_pred, classes=[0, 1, 2])
n_classes = y_test_binarized.shape[1]

fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_pred_binarized[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

for i in range(n_classes):
    plt.figure()
    plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], 'k--')
    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')
    plt.legend(loc="lower right")
    plt.show()
```


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0



Results 📊

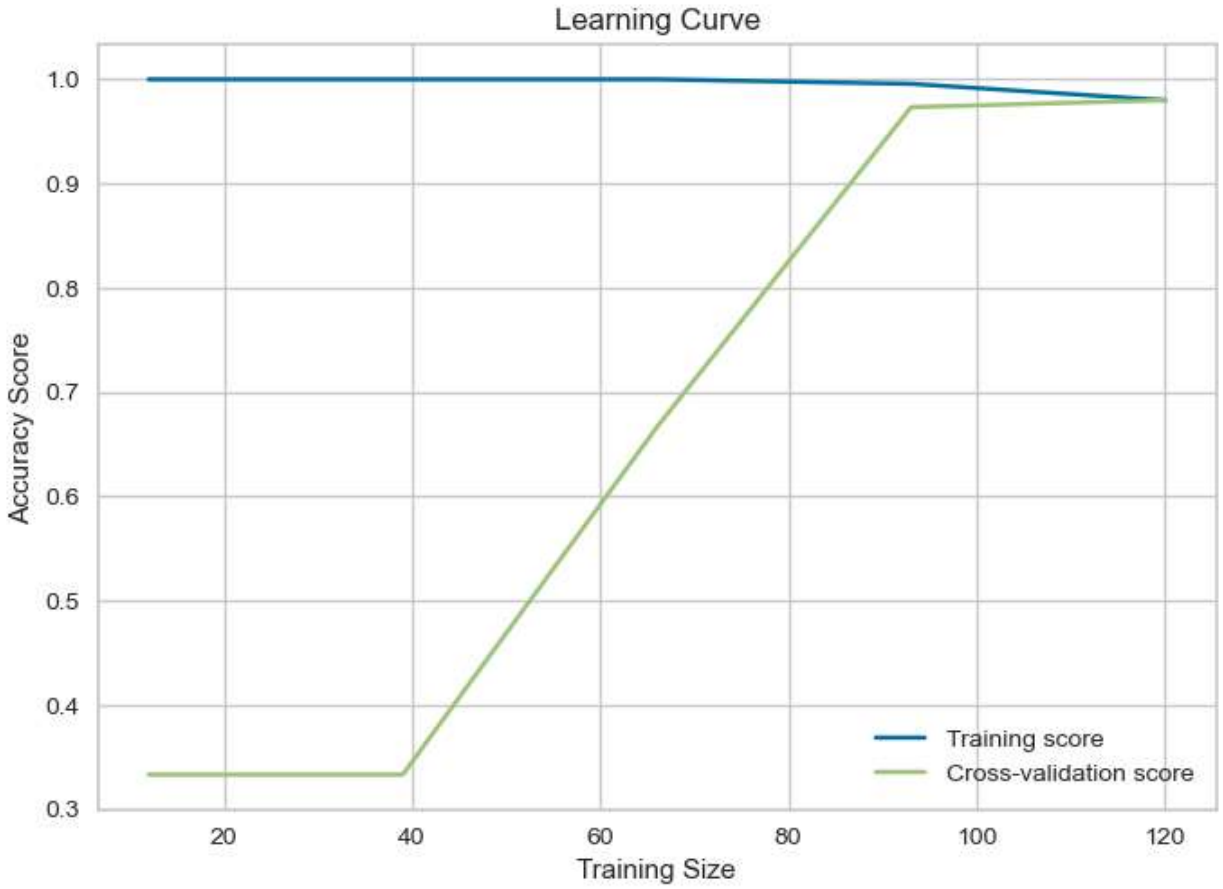
The Receiver Operating Characteristic (ROC) curve is a graphical representation illustrating the trade-off between the true positive rate (sensitivity) and the false positive rate at various classification thresholds.

Observations from the ROC Curve:

- The ROC curve manifests as a perfect step function, characterized by an immediate ascent to a true positive rate of 1.0, followed by a horizontal shift towards the right. This characteristic denotes a perfect classifier, indicating the existence of a threshold where all positive instances are correctly identified without misclassifying any negative instances.
- The Area Under the ROC Curve (AUC) attains a value of 1.00. AUC quantifies the entire two-dimensional area beneath the ROC curve, ranging from (0,0) to (1,1), providing a comprehensive measure of classification performance across all possible thresholds. A value of 1.0 for AUC signifies flawless classification, underlining the model's impeccable discriminative ability.

In [15]:
```
# Learning Curve
train_sizes, train_scores, test_scores = learning_curve(lda, X, y, cv=5)
train_scores_mean = np.mean(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)

plt.plot(train_sizes, train_scores_mean, label='Training score')
plt.plot(train_sizes, test_scores_mean, label='Cross-validation score')
```

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



Results 📊

Learning curves serve as a valuable tool to assess the impact of training data size on model performance and to diagnose whether the model is affected more by variance or bias errors.

Observations from the Learning Curve:

- **Training Score (Blue Line):**
  - Initially starts at a perfect score of 1.0 for a small number of training examples, indicating the model's tendency to overfit the training data when trained on a limited dataset.
  - As the training set size increases, the training score slightly decreases, suggesting a reduction in overfitting as the model is exposed to more data.
- **Cross-Validation Score (Green Line):**
  - Commences at a lower value for a small number of training examples, signaling the model's underfitting of the validation data when trained on a small dataset.
  - With an increase in the training set size, the cross-validation score rises and converges towards the training score. This convergence indicates improved generalization to

unseen data as the model learns from more examples.

The convergence of both training and cross-validation scores to a high value implies that the model strikes a balance between its ability to generalize to unseen data (bias) and its sensitivity to variations in the training data (variance). This suggests that the model neither suffers from high bias nor high variance, indicating a well-balanced performance.

In [16]:
```
importance = lda.coef_
for i, j in enumerate(importance):
    print('Class %d: %s' % ((i+1), j))
```

```
Class 1: [  5.6329093   14.78025075 -17.95274031 -19.82806589]
Class 2: [-1.31638676 -5.17819724  5.00477139  3.53283552]
Class 3: [-4.39344909 -9.71548571 13.15164065 16.62247126]
```

The coefficients obtained from the Linear Discriminant Analysis (LDA) model provide insights into the importance of features for each class. The interpretation of these coefficients is crucial in understanding the impact of each feature on the probability of a particular class.

Class 1 (Setosa):

- The model assigns a substantial negative weight to PetalLengthCm and PetalWidthCm, indicating that shorter and narrower petals significantly decrease the log-odds of being classified as Setosa.
- Conversely, a positive weight is assigned to SepalLengthCm and SepalWidthCm, implying that longer and wider sepals contribute positively to the likelihood of being classified as Setosa.

Class 2 (Versicolor):

- The model assigns a negative weight to SepalLengthCm and SepalWidthCm, suggesting that longer and wider sepals decrease the log-odds of being classified as Versicolor.
- In contrast, a positive weight is assigned to PetalLengthCm and PetalWidthCm, indicating that shorter and narrower petals enhance the likelihood of being classified as Versicolor.

Class 3 (Virginica):

- A strong positive weight is assigned to PetalLengthCm and PetalWidthCm, emphasizing that longer and wider petals significantly increase the log-odds of being classified as Virginica.
- Conversely, a negative weight is assigned to SepalLengthCm and SepalWidthCm, implying that shorter and narrower sepals diminish the likelihood of being classified as Virginica.

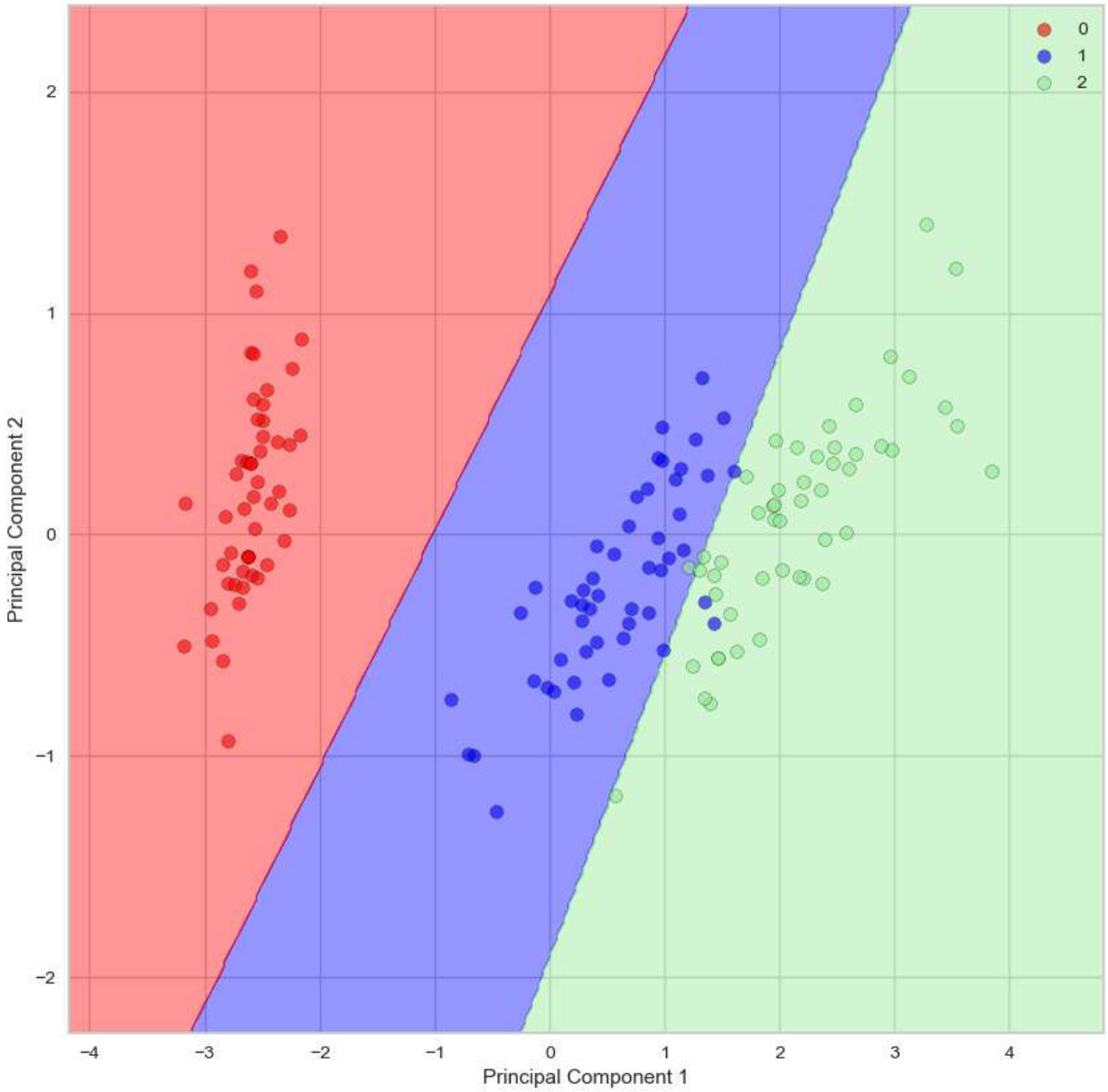
In [17]:
```
# Confidence Interval
confidence_interval = 1.96 * np.std(y_pred) / np.sqrt(len(y_pred))
print("Confidence Interval: ", confidence_interval)
```

```
Confidence Interval:  0.2991572360011568
```

The overall confidence interval is 0.299. This means that we can be confident that the true value of the population parameter lies within this interval around the model's predictions 95% of the

time (assuming a 95% confidence level).

In [18]:
```
pca = PCA(n_components=2)
X_train2 = pca.fit_transform(X_train)
X_test2 = pca.transform(X_test)

lda2 = LinearDiscriminantAnalysis()
lda2.fit(X_train2, y_train)

X_combined_std = np.vstack((X_train2, X_test2))
y_combined = np.hstack((y_train, y_test))

colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
cmap = ListedColormap(colors[:len(np.unique(y_test))])

plt.figure(figsize=(10, 10))


x1_min, x1_max = X_combined_std[:, 0].min() - 1, X_combined_std[:, 0].max() + 1
x2_min, x2_max = X_combined_std[:, 1].min() - 1, X_combined_std[:, 1].max() + 1

xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, 0.02),
                       np.arange(x2_min, x2_max, 0.02))

Z = lda2.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
Z = Z.reshape(xx1.shape)

plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
plt.xlim(xx1.min(), xx1.max())
plt.ylim(xx2.min(), xx2.max())

for idx, cl in enumerate(np.unique(y_combined)):
    plt.scatter(x=X_combined_std[y_combined==cl, 0],
                y=X_combined_std[y_combined==cl, 1],
                alpha=0.6,
                color=cmap(idx),
                edgecolor='black',
                marker='o',
                s=50,
                label=cl)

plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(loc='upper right')
plt.show()
```



The dataset was reduced to two dimensions using Principal Component Analysis (PCA), which is a dimensionality reduction technique that transforms the data to a new coordinate system such that the greatest variance by some projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.

The plot illustrates the decision boundaries demarcating the three classes (Setosa, Versicolor, and Virginica) within the space of the first two principal components.

Distinct colors denote regions where the model assigns a particular class prediction. For instance, observations falling within the red region are predicted as Setosa, those within the blue region as Versicolor, and those within the green region as Virginica.

The model effectively segregates the three classes, exhibiting a clear demarcation for the Setosa class from the others. However, a slight overlap is noticeable between the Versicolor and Virginica classes.

Scatter points represent observations from both the training and test sets. The majority of observations are accurately classified, as evidenced by their alignment within the regions corresponding to their true classes.

SUMMMARY

1. **Model Evaluation**:
   - The Linear Discriminant Analysis (LDA) model exhibited exceptional performance on the Iris dataset, achieving a flawless accuracy of 100% on the test set.
   - Precision, recall, and F1-score metrics, all registering a perfect score of 1.00 across all three classes, signify the model's impeccable ability to precisely classify all samples in the test set, without any false positives or false negatives.
   - Cohen's Kappa coefficient of 1.0 and Matthews Correlation Coefficient of 1.0 further underscore the model's outstanding performance.
2. **Influence Analysis**:
   - Examination through Cook's Distance plot revealed the absence of influential observations that might have disproportionately impacted the model's predictions.
3. **Cross-Validation**:
   - Cross-validation across diverse subsets consistently yielded high accuracies, surpassing 93% in all folds. The mean cross-validation score of 0.98 indicates the model's robustness, correctly classifying an average of 98% of samples.
4. **ROC Curve**:
   - The ROC curve depicted a perfect step function, characteristic of an ideal classifier. An area under the ROC curve (AUC) of 1.00 signifies flawless classification.
5. **Learning Curve**:
   - Analysis of the learning curve indicated the absence of high bias or variance issues. The convergence of training score and cross-validation score to a high value with increasing training set size reflects the model's balanced generalization capability and insensitivity to fluctuations in training data.
6. **Feature Importance**:
   - The LDA model assigned varying weights to features across different classes, with petal measurements demonstrating greater significance in species differentiation compared to sepal measurements.
7. **Decision Region Plot**:
   - Visual representation via the decision region plot illustrated the model's adeptness at delineating the three classes within the first two principal components' space. Notably, the Setosa class appeared distinctly separated, whereas a minor overlap was observed between Versicolor and Virginica classes.

In [ ]: