In [13]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from lazypredict.Supervised import LazyClassifier
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.metrics import roc_curve, auc
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import warnings

# Load the heart disease dataset
df_heart_disease = pd.read_csv("C:/Users/ali/Desktop/osdabig/questionno3/datasetosda/hh/heart2.csv")
df_heart_disease.rename(columns={"target": "Heart_D"}, inplace=True)

# Separate features and target variable
X = df_heart_disease.drop('Heart_D', axis=1)
y = df_heart_disease['Heart_D']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=55)

# LazyClassifier
clf = LazyClassifier(verbose=0, ignore_warnings=True, custom_metric=None)
models, predictions = clf.fit(X_train, X_test, y_train, y_test)
models
```

```
100%|██████████████████████████████████████████████████████████████████████████████████| 29/29 [00:03<00:00,  8.
20it/s]
[LightGBM] [Info] Number of positive: 135, number of negative: 24
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000362 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 192
[LightGBM] [Info] Number of data points in the train set: 159, number of used features: 13
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.849057 -> initscore=1.727221
[LightGBM] [Info] Start training from score 1.727221
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

Out[13]:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
NearestCentroid	0.82	0.85	0.85	0.83	0.06
GaussianNB	0.85	0.83	0.83	0.85	0.05
DecisionTreeClassifier	0.88	0.82	0.82	0.87	0.05
LGBMClassifier	0.90	0.80	0.80	0.89	0.14
ExtraTreesClassifier	0.90	0.80	0.80	0.89	0.47
KNeighborsClassifier	0.90	0.80	0.80	0.89	0.06
XGBClassifier	0.88	0.78	0.78	0.87	0.14
CalibratedClassifierCV	0.88	0.78	0.78	0.87	0.13
RandomForestClassifier	0.88	0.78	0.78	0.87	0.69
SGDClassifier	0.85	0.77	0.77	0.84	0.05
RidgeClassifierCV	0.85	0.77	0.77	0.84	0.05
RidgeClassifier	0.85	0.77	0.77	0.84	0.04
BaggingClassifier	0.85	0.77	0.77	0.84	0.14
LogisticRegression	0.82	0.75	0.75	0.82	0.06
LinearSVC	0.82	0.75	0.75	0.82	0.05
LinearDiscriminantAnalysis	0.82	0.75	0.75	0.82	0.05
ExtraTreeClassifier	0.82	0.75	0.75	0.82	0.05
BernoulliNB	0.82	0.75	0.75	0.82	0.05
QuadraticDiscriminantAnalysis	0.82	0.68	0.68	0.80	0.14
LabelSpreading	0.82	0.68	0.68	0.80	0.05
LabelPropagation	0.82	0.68	0.68	0.80	0.05
SVC	0.82	0.65	0.65	0.79	0.05
AdaBoostClassifier	0.80	0.63	0.63	0.77	0.64
PassiveAggressiveClassifier	0.70	0.53	0.53	0.67	0.05
DummyClassifier	0.75	0.50	0.50	0.64	0.04
Perceptron	0.68	0.45	0.45	0.60	0.11

In [14]:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

def tune_knn_parameters(X, y):
    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=40)

    # Test different values of k
    k_values = range(1, 20)
    for k in k_values:
        knn = KNeighborsClassifier(n_neighbors=k)
        scores = cross_val_score(knn, X_train, y_train, cv=5)
        print(f'k={k}, Mean Accuracy: {np.mean(scores)}')

def lazy_classification(X_train, y_train, X_test, y_test, k_value):
    # Train k-NN with the selected k
    knn = KNeighborsClassifier(n_neighbors=k_value)
    knn.fit(X_train, y_train)

    # Predictions on the test set
    y_pred = knn.predict(X_test)

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix:")
    print(cm)

    # Classification Report
    cr = classification_report(y_test, y_pred)
    print("\nClassification Report:")
    print(cr)

    # Evaluate the model on the test set
```

```
    accuracy = knn.score(X_test, y_test)
    print(f'Test Set Accuracy: {accuracy}')

# Example usage
tune_knn_parameters(X, y)
```

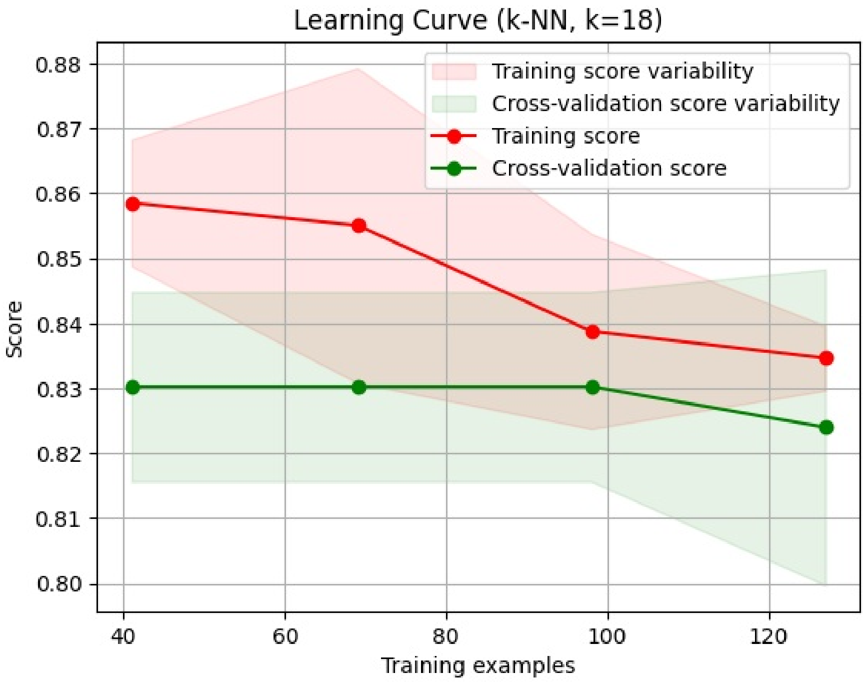
```
k=1, Mean Accuracy: 0.7862903225806452
k=2, Mean Accuracy: 0.6917338709677419
k=3, Mean Accuracy: 0.7983870967741936
k=4, Mean Accuracy: 0.7856854838709678
k=5, Mean Accuracy: 0.830241935483871
k=6, Mean Accuracy: 0.8110887096774194
k=7, Mean Accuracy: 0.8300403225806452
k=8, Mean Accuracy: 0.8173387096774194
k=9, Mean Accuracy: 0.836491935483871
k=10, Mean Accuracy: 0.842741935483871
k=11, Mean Accuracy: 0.842741935483871
k=12, Mean Accuracy: 0.8362903225806452
k=13, Mean Accuracy: 0.842741935483871
k=14, Mean Accuracy: 0.836491935483871
k=15, Mean Accuracy: 0.836491935483871
k=16, Mean Accuracy: 0.836491935483871
k=17, Mean Accuracy: 0.836491935483871
k=18, Mean Accuracy: 0.836491935483871
k=19, Mean Accuracy: 0.836491935483871
```

In [15]:
```
lazy_classification(X_train, y_train, X_test, y_test, 18)
```

```
Confusion Matrix:
[[ 0 10]
 [ 0 30]]

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        10
           1       0.75      1.00      0.86        30

    accuracy                           0.75        40
   macro avg       0.38      0.50      0.43        40
weighted avg       0.56      0.75      0.64        40

Test Set Accuracy: 0.75
```

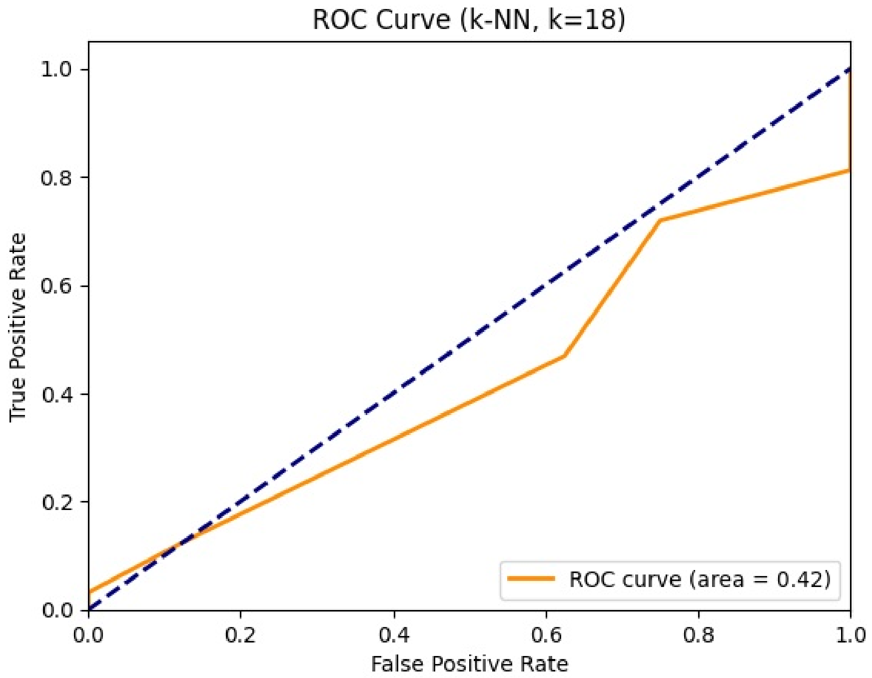
In [16]:
```
import numpy as np

def plot_learning_curve(estimator, title, X, y, cv=5, n_jobs=-1, train_sizes=np.linspace(.1, 1.0, 5)):
    """
   Parameters:
- estimator: The entity employed to fit the data.
- title: The chart's title.
- X: Training vector, where n_samples represents the number of samples, and n_features signifies the number of 
- y: Target corresponding to X for classification or regression.
- cv: Cross-validation generator or an iterable, default=None.
- n_jobs: The number of parallel jobs to execute, default=None.
- train_sizes: Relative or absolute quantities of training examples utilized for generating the learning curve.

Returns:
- plt: Matplotlib plot object.

    """
    plt.figure()
    plt.title(title)
    plt.xlabel("Training examples")
    plt.ylabel("Score")

    train_sizes, train_scores, test_scores, fit_times, _ = \
        learning_curve(estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes, return_times=True)

    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r", label="Training score variability")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1,
                     color="g", label="Cross-validation score variability")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
```

```
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt

# Assuming X_train, y_train, X_test, y_test are your training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Choose a k value for k-NN
k_value = 18
knn = KNeighborsClassifier(n_neighbors=k_value)

# Plot learning curve
plot_learning_curve(knn, f"Learning Curve (k-NN, k={k_value})", X_train, y_train, cv=5, n_jobs=-1)
plt.show()
```



In [17]:

```
# Function for tuning k-NN parameters
def tune_knn_parameters(X, y, k_values=18):
    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=40)

    # Create a list for multiple k values
    if isinstance(k_values, int):
        k_values = [k_values]

    for k in k_values:
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)

        # Plot ROC curve
        plot_roc_curve(knn, X_test, y_test, f"ROC Curve (k-NN, k={k})")

# Function to plot ROC curve
def plot_roc_curve(model, X_test, y_test, title):
    y_prob = model.predict_proba(X_test)[:, 1]
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    roc_auc = auc(fpr, tpr)

    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(title)
    plt.legend(loc="lower right")
    plt.show()

# Assuming X, y are your dataset
tune_knn_parameters(X, y, k_values=18)
```



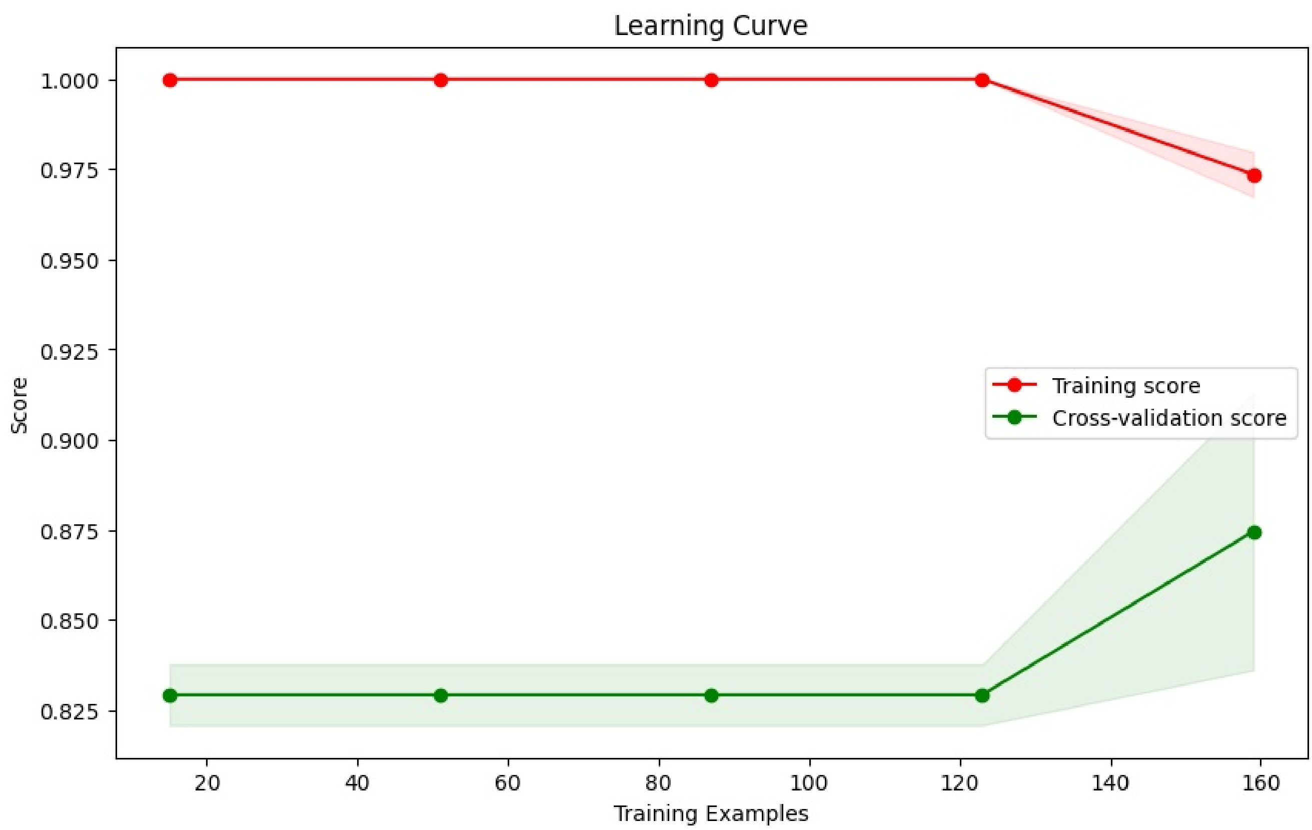
In [18]:
```
# Using Decision Tree Classifier Model
```

In [19]:
```
from sklearn.tree import DecisionTreeClassifier

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=40)

# Decision Tree Classifier
# Tuning the max depth parameter
best_depth = None
best_accuracy_tree = 0

# Add more values for tuning max depth
for depth_value in [None, 5, 10, 15]:
    # Train Decision Tree Classifier with the current max depth
    dtc = DecisionTreeClassifier(max_depth=depth_value, random_state=42)
    scores = cross_val_score(dtc, X_train, y_train, cv=5, scoring='accuracy')

    # Compute mean accuracy
    mean_accuracy_tree = scores.mean()

    print(f"Max Depth={depth_value}, Mean Accuracy: {mean_accuracy_tree}")

    # Update best parameters if needed
    if mean_accuracy_tree > best_accuracy_tree:
        best_accuracy_tree = mean_accuracy_tree
        best_depth = depth_value

# Choose the optimal max depth based on the tuning results
optimal_depth = best_depth

# Perform classification with the optimal max depth
dtc = DecisionTreeClassifier(max_depth=optimal_depth, random_state=42)
dtc.fit(X_train, y_train)

# Predictions on the test set
y_pred_tree = dtc.predict(X_test)

# Decision Tree Classifier - Confusion Matrix
cm_tree = confusion_matrix(y_test, y_pred_tree)
print("\nDecision Tree Classifier - Confusion Matrix:")
print(cm_tree)

# Decision Tree Classifier - Classification Report
cr_tree = classification_report(y_test, y_pred_tree)
print("\nDecision Tree Classifier - Classification Report:")
print(cr_tree)

# Decision Tree Classifier - Evaluate the model on the test set
accuracy_tree = dtc.score(X_test, y_test)
print(f'Decision Tree Classifier - Test Set Accuracy: {accuracy_tree}')
```

```
Max Depth=None, Mean Accuracy: 0.7985887096774194
Max Depth=5, Mean Accuracy: 0.8050403225806452
Max Depth=10, Mean Accuracy: 0.7985887096774194
Max Depth=15, Mean Accuracy: 0.7985887096774194

Decision Tree Classifier - Confusion Matrix:
[[ 6  2]
 [ 2 30]]

Decision Tree Classifier - Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.75      0.75         8
           1       0.94      0.94      0.94        32

    accuracy                           0.90        40
   macro avg       0.84      0.84      0.84        40
weighted avg       0.90      0.90      0.90        40

Decision Tree Classifier - Test Set Accuracy: 0.9
```

In [20]:

```python
# Assuming `model` is your trained model and `X`, `y` are your features and labels
train_sizes, train_scores, test_scores = learning_curve(dtc, X, y, cv=5)

# Calculate mean and standard deviation across cross-validation folds
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

# Plot learning curve
plt.figure(figsize=(10, 6))
plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean + train_scores_std, alpha
plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_scores_mean + test_scores_std, alpha=0.1
plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training score")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation score")

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



In [21]:

```python
# Plot ROC curve
y_prob = dtc.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))  # Adjust the figure size if needed
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
```

```python
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')  # Add a title if desired
plt.legend(loc="lower right")
plt.show()
```