IRID_data with traditional ML methods

In [ ]:

In [17]:
```
# Import necessary libraries
# Import the `pandas` library to load the dataset
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier

# Import necessary libraries for visualization
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.inspection import permutation_importance
import numpy as np
```

In [1]:

In [2]:
```
# Read datasets from the file
iris_file = pd.read_csv("iris.csv")
iris_file.dtypes
```

Out[2]:
```
SepalLength    float64
SepalWidth     float64
PetalLength    float64
PetalWidth     float64
Name            object
dtype: object
```

In [3]:
```
# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
```

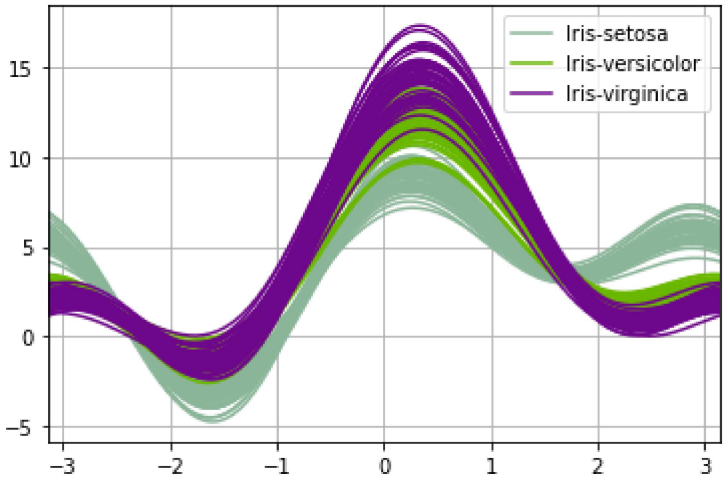
In [4]:
```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [5]:
```
# Import the plotting library
```

```
from pandas.plotting import andrews_curves
```

In [6]:
```
andrews_curves(iris_file, "Name")
```

Out[6]: <AxesSubplot:>

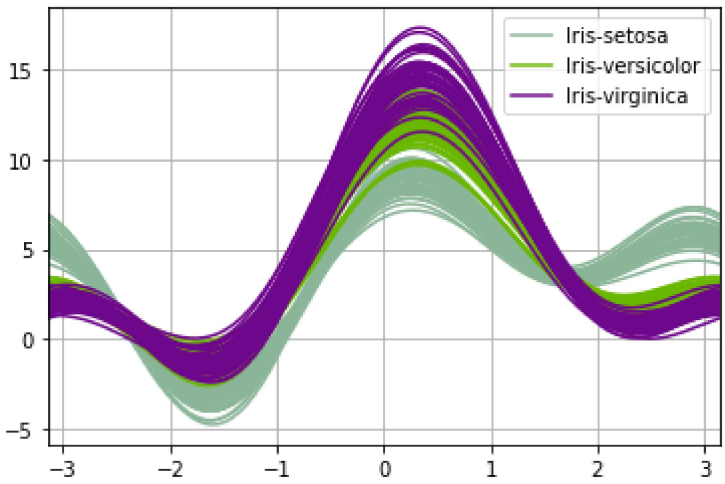


In [7]:
```
# Read datasets directly from the URL
csv_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

# The CSV file at the UCI repository does not contain the header
header = ["SepalLength", "SepalWidth", "PetalLength", "PetalWidth", "Name"]

iris_url = pd.read_csv(csv_url, names = header)
iris_url.dtypes
```

Out[7]:
```
SepalLength    float64
SepalWidth     float64
PetalLength    float64
PetalWidth     float64
Name            object
dtype: object
```

In [8]:
```
andrews_curves(iris_url, "Name")
```

Out[8]: <AxesSubplot:>



In [9]:
```
# 1. Logistic Regression
log_reg = LogisticRegression(max_iter=200)
log_reg.fit(X_train, y_train)
y_pred_log_reg = log_reg.predict(X_test)
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_log_reg))
print("Classification Report:\n", classification_report(y_test, y_pred_log_reg))
```

```
Logistic Regression Accuracy: 1.0
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 [10]:
```
# 2. Random Forest
random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
random_forest.fit(X_train, y_train)
y_pred_rf = random_forest.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Classification Report:\n", classification_report(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 1.0
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 [11]:
```
# 3. Support Vector Machine (SVM)
svm = SVC(kernel='linear')
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)
print("Support Vector Machine (SVM) Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Classification Report:\n", classification_report(y_test, y_pred_svm))
```

```
Support Vector Machine (SVM) Accuracy: 0.9666666666666667
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.89      0.94         9
           2       0.92      1.00      0.96        11

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.97        30
weighted avg       0.97      0.97      0.97        30
```

In [14]:
```
# 4. K-Nearest Neighbors (KNN)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
print("K-Nearest Neighbors (KNN) Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Classification Report:\n", classification_report(y_test, y_pred_knn))
```

```
K-Nearest Neighbors (KNN) Accuracy: 1.0
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 [15]:
```
# 5. Naive Bayes
naive_bayes = GaussianNB()
naive_bayes.fit(X_train, y_train)
y_pred_nb = naive_bayes.predict(X_test)
print("Naive Bayes Accuracy:", accuracy_score(y_test, y_pred_nb))
print("Classification Report:\n", classification_report(y_test, y_pred_nb))
```

```
Naive Bayes Accuracy: 1.0
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
```

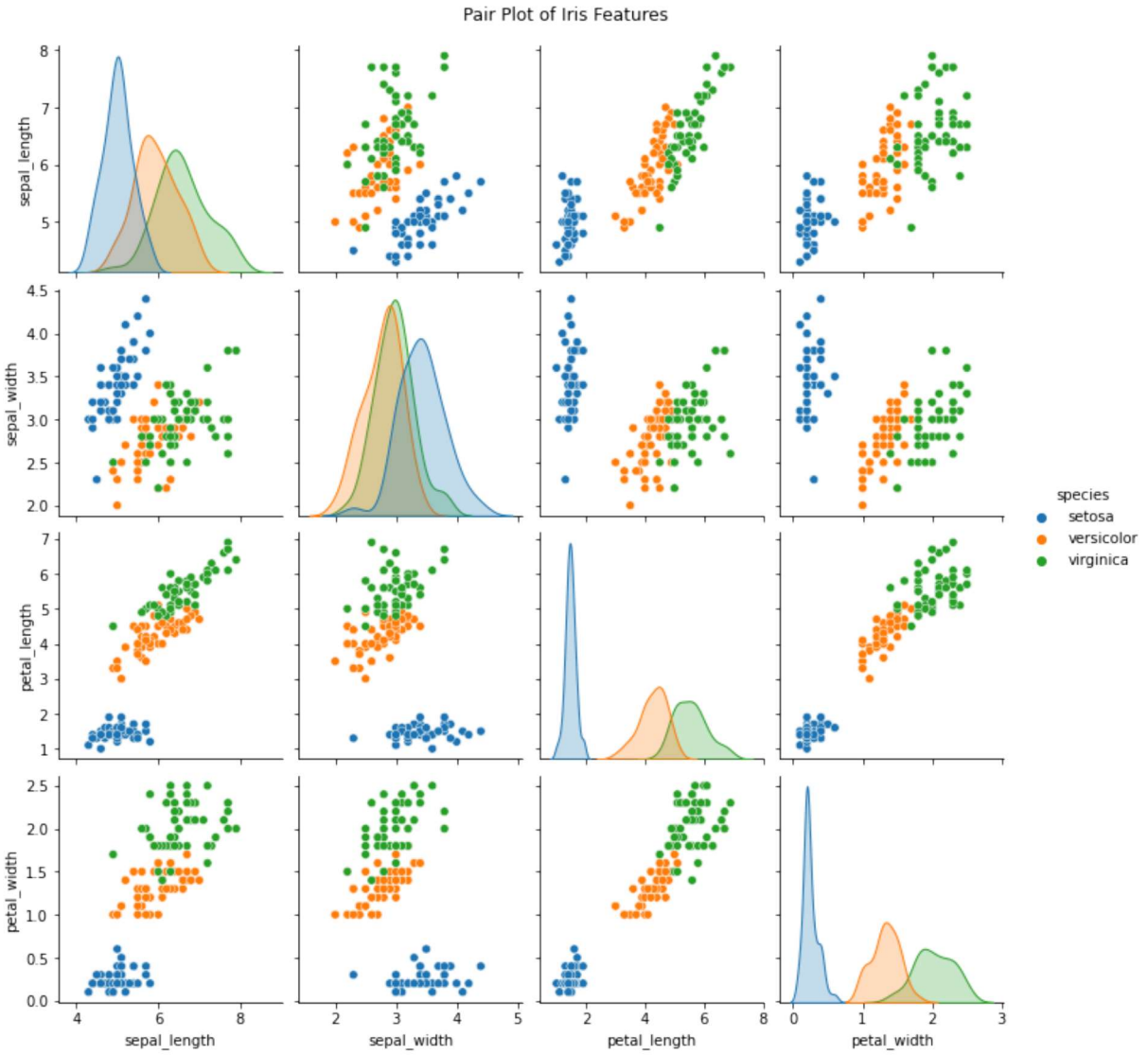
In [16]:
```
# 6. Gradient Boosting (XGBoost)
xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_state=4
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)
```

```
print("Gradient Boosting (XGBoost) Accuracy:", accuracy_score(y_test, y_pred_xgb))
print("Classification Report:\n", classification_report(y_test, y_pred_xgb))
```

```
Gradient Boosting (XGBoost) Accuracy: 1.0
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
```

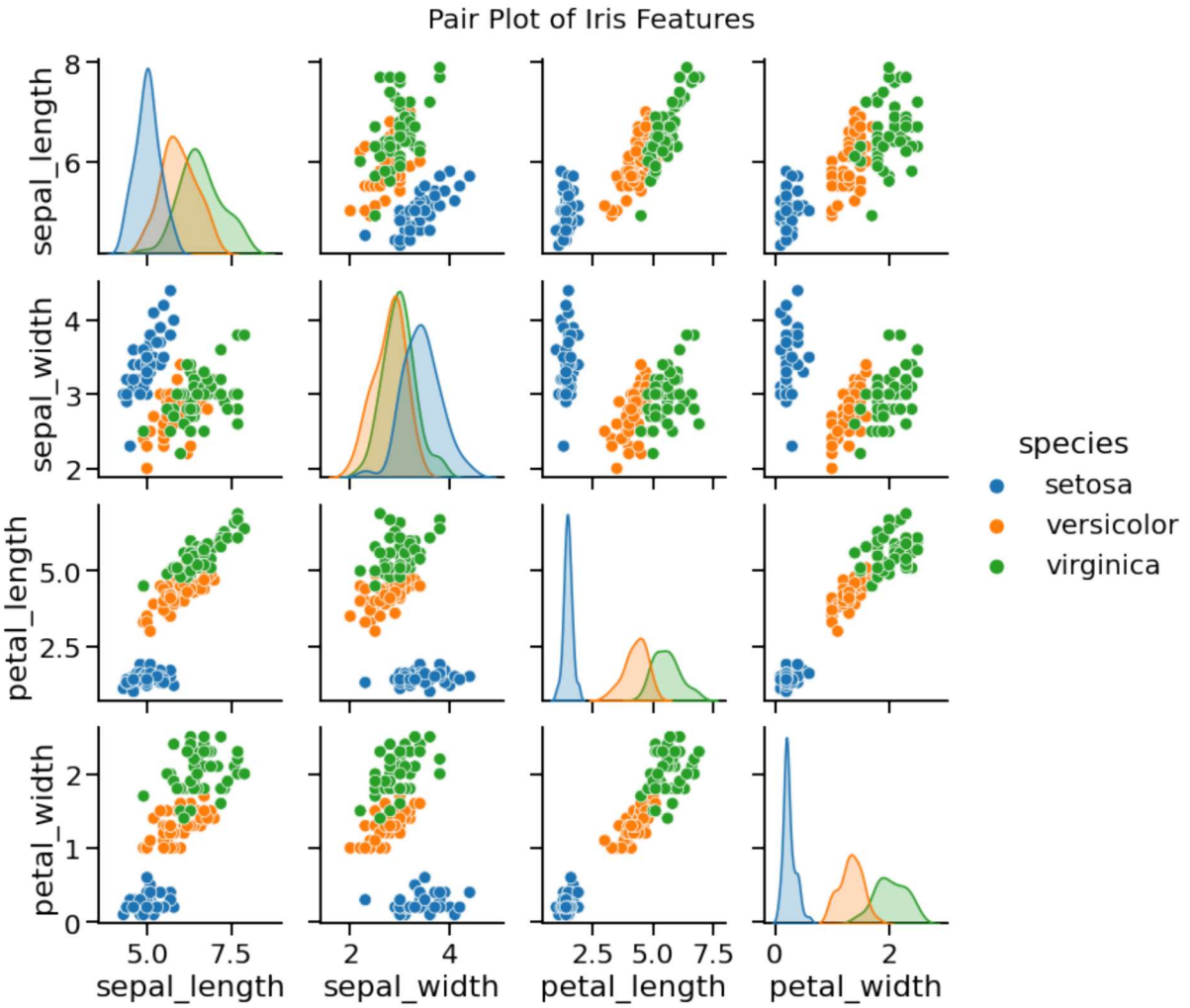
In [18]:
```
# Load the Iris dataset for visualization
iris = load_iris()
X = iris.data
y = iris.target
iris_df = sns.load_dataset("iris")

# 1. Pair Plot
plt.figure(figsize=(10, 6))
sns.pairplot(iris_df, hue="species", diag_kind="kde")
plt.suptitle("Pair Plot of Iris Features", y=1.02)
plt.show()
```

```
<Figure size 720x432 with 0 Axes>
```

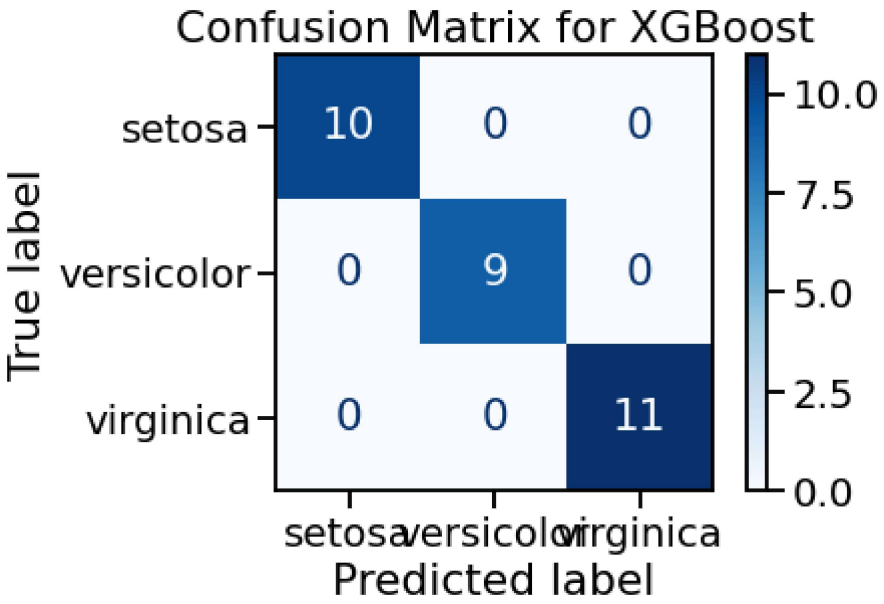


In [19]:
```
# Load the Iris dataset for visualization
iris_df = sns.load_dataset("iris")

# Set Seaborn context for larger labels and titles
sns.set_context("talk", font_scale=1.2)  # 'talk' context with a slightly larger fo

# Create a high-resolution pair plot
plt.figure(figsize=(14, 8))  # Define figure size for clear visibility
pair_plot = sns.pairplot(iris_df, hue="species", diag_kind="kde")

# Adjust the overall title of the plot
pair_plot.fig.suptitle("Pair Plot of Iris Features", y=1.02, fontsize=20)

# Show the plot
plt.show()

# Save the pair plot with high resolution
pair_plot.fig.savefig("iris_pairplots_high_res.jpg", dpi=300)  # Save with 300 DPI
```

```
<Figure size 1008x576 with 0 Axes>
```

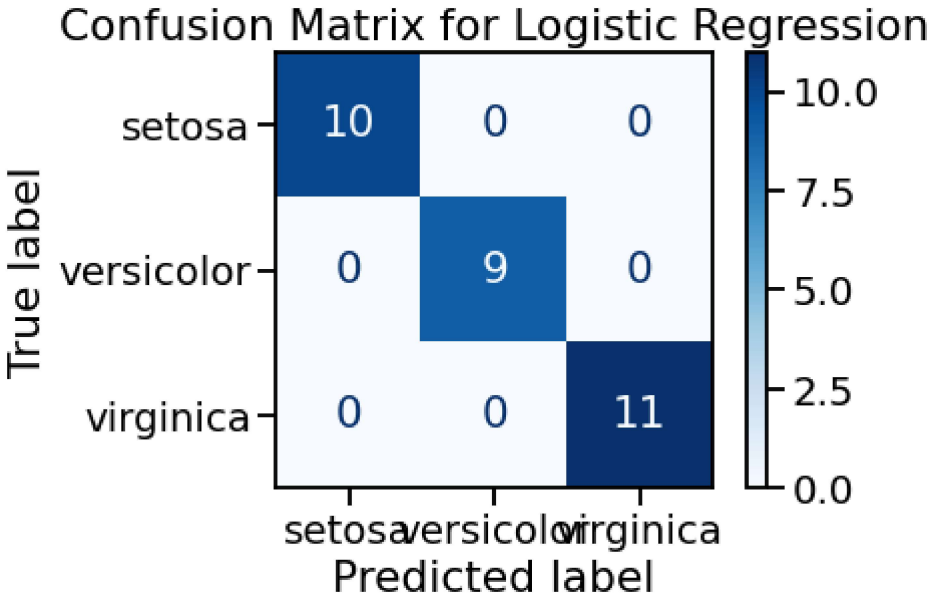


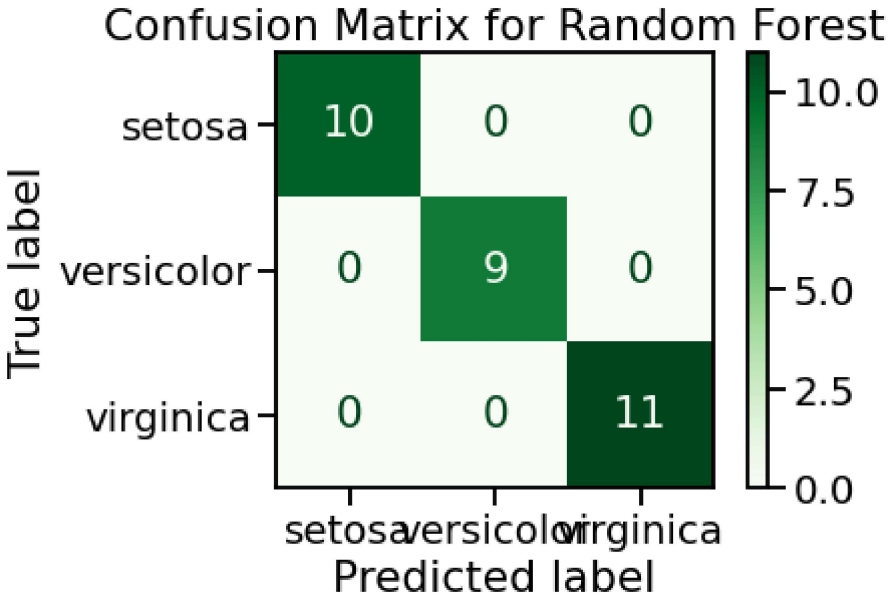
In [20]:
```
# 2. Confusion Matrix for XGBoost
y_pred_xgb = xgb.predict(X_test)
cm = confusion_matrix(y_test, y_pred_xgb)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=iris.target_names
disp.plot(cmap='Blues')
plt.title("Confusion Matrix for XGBoost")
plt.show()
```

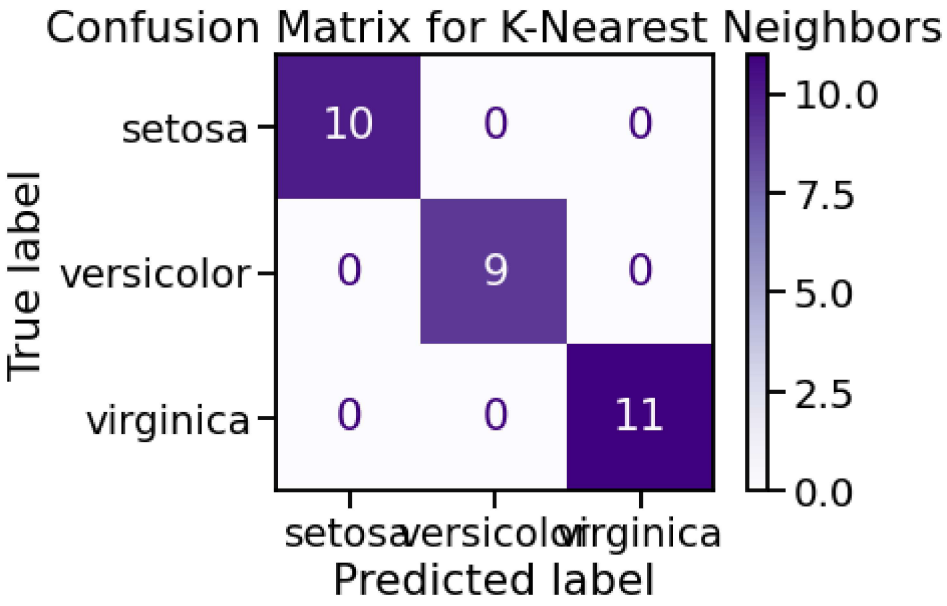


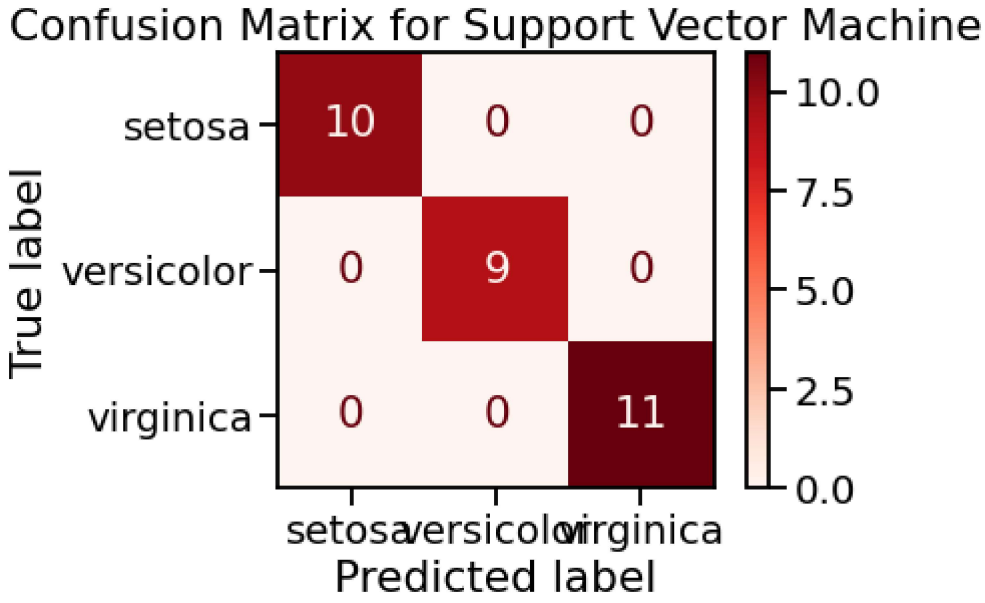
In [24]:
```python
# Import necessary libraries for visualization
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier

# Load dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Initialize and fit models
logreg = LogisticRegression(max_iter=200)
logreg.fit(X_train, y_train)

random_forest = RandomForestClassifier(random_state=42)
random_forest.fit(X_train, y_train)

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)

svm = SVC(probability=True)
svm.fit(X_train, y_train)

xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
xgb.fit(X_train, y_train)

# Function to plot confusion matrices
def plot_confusion_matrix(model, X_test, y_test, model_name, cmap):
```

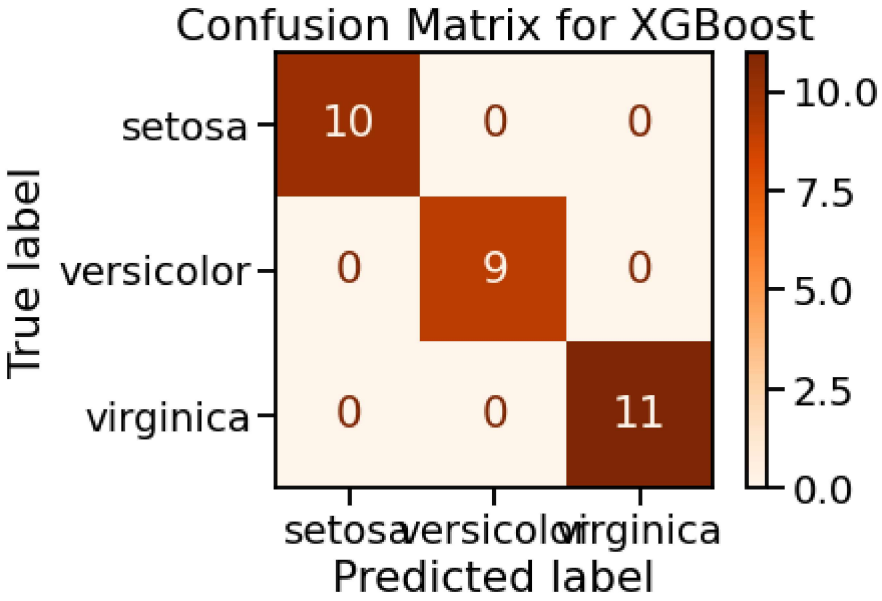
```
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=iris.target_n
    disp.plot(cmap=cmap)
    plt.title(f"Confusion Matrix for {model_name}")
    plt.show()

# Plot confusion matrices for each model
plot_confusion_matrix(logreg, X_test, y_test, "Logistic Regression", 'Blues')
plot_confusion_matrix(random_forest, X_test, y_test, "Random Forest", 'Greens')
plot_confusion_matrix(knn, X_test, y_test, "K-Nearest Neighbors", 'Purples')
plot_confusion_matrix(svm, X_test, y_test, "Support Vector Machine", 'Reds')
plot_confusion_matrix(xgb, X_test, y_test, "XGBoost", 'Oranges')
```



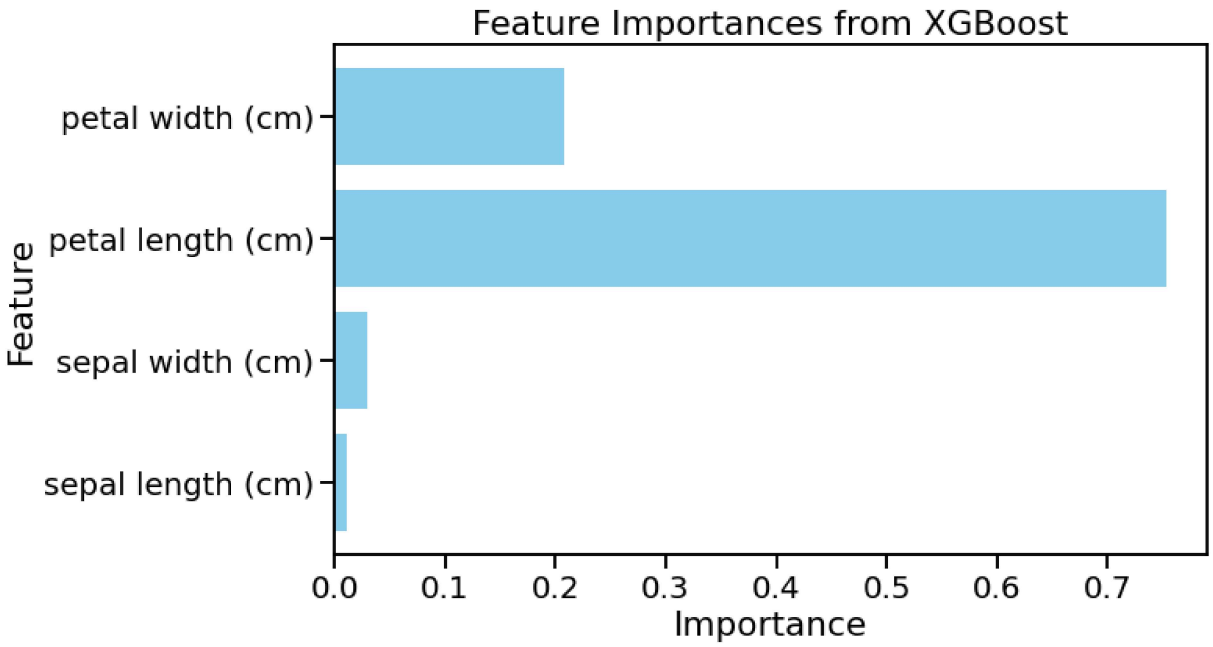



Confusion Matrix for K-Nearest Neighbors
setosa
versicolor
virginica
True label
10
0
0
0
9
0
0
0
11
10.0
7.5
5.0
2.5
0.0
setosa
versicolor
virginica
Predicted label

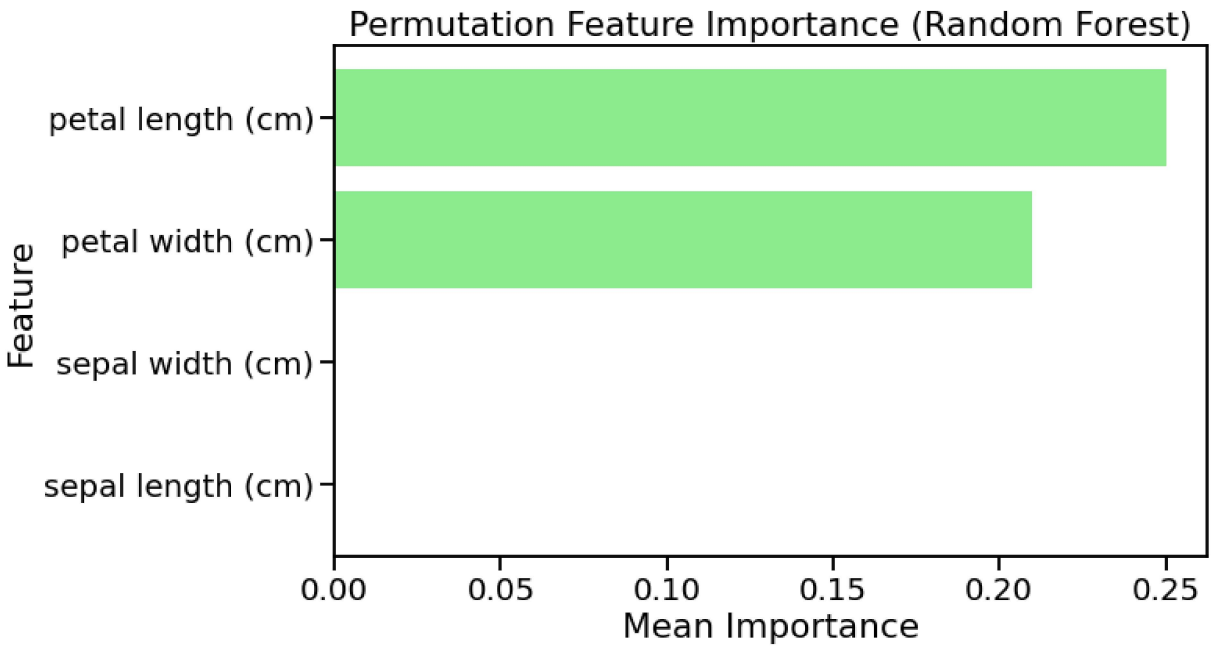

Confusion Matrix for Support Vector Machine
setosa
versicolor
virginica
True label
10
0
0
0
9
0
0
0
11
10.0
7.5
5.0
2.5
0.0
setosa
versicolor
virginica
Predicted label


Confusion Matrix for XGBoost
setosa
versicolor
virginica
True label
10
0
0
0
9
0
0
0
11
10.0
7.5
5.0
2.5
0.0
setosa
versicolor
virginica
Predicted label

In [21]:
```
# 3. Feature Importance Plot for XGBoost
# Get feature importances from XGBoost model
feature_importances = xgb.feature_importances_
feature_names = iris.feature_names
plt.figure(figsize=(10, 6))
plt.barh(feature_names, feature_importances, color='skyblue')
plt.title("Feature Importances from XGBoost")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()
```

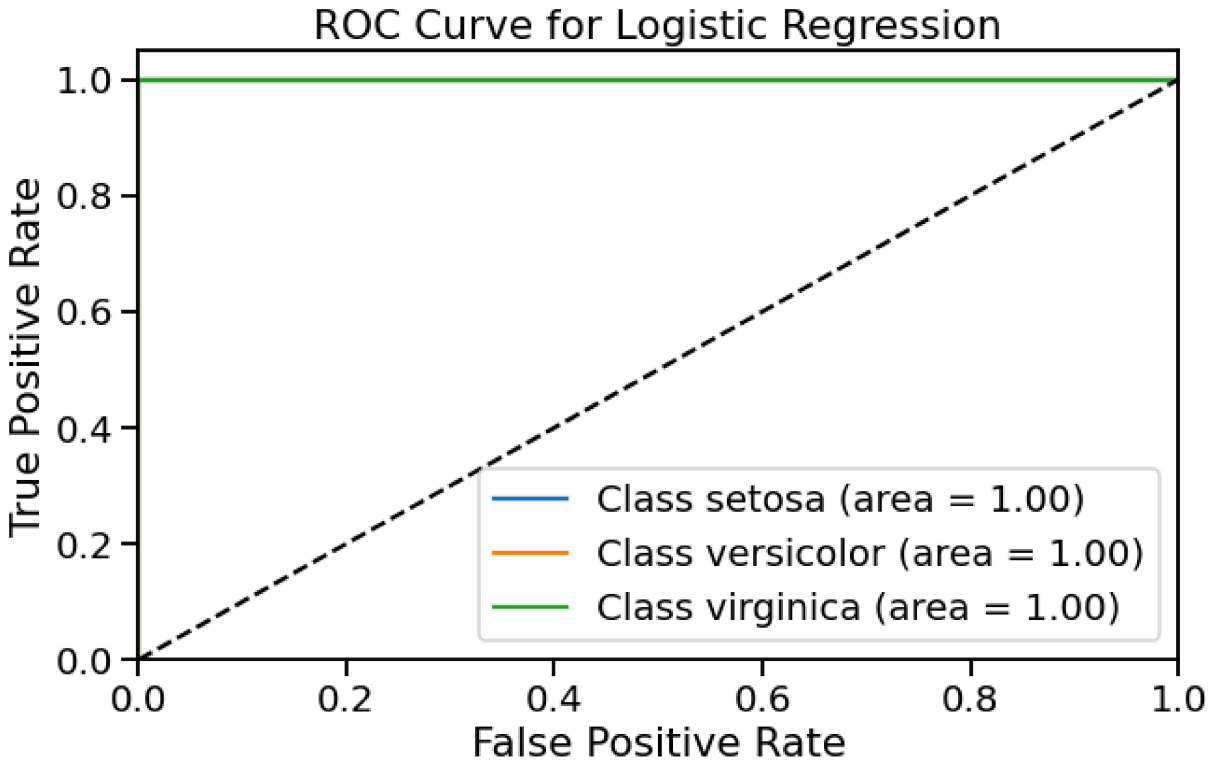


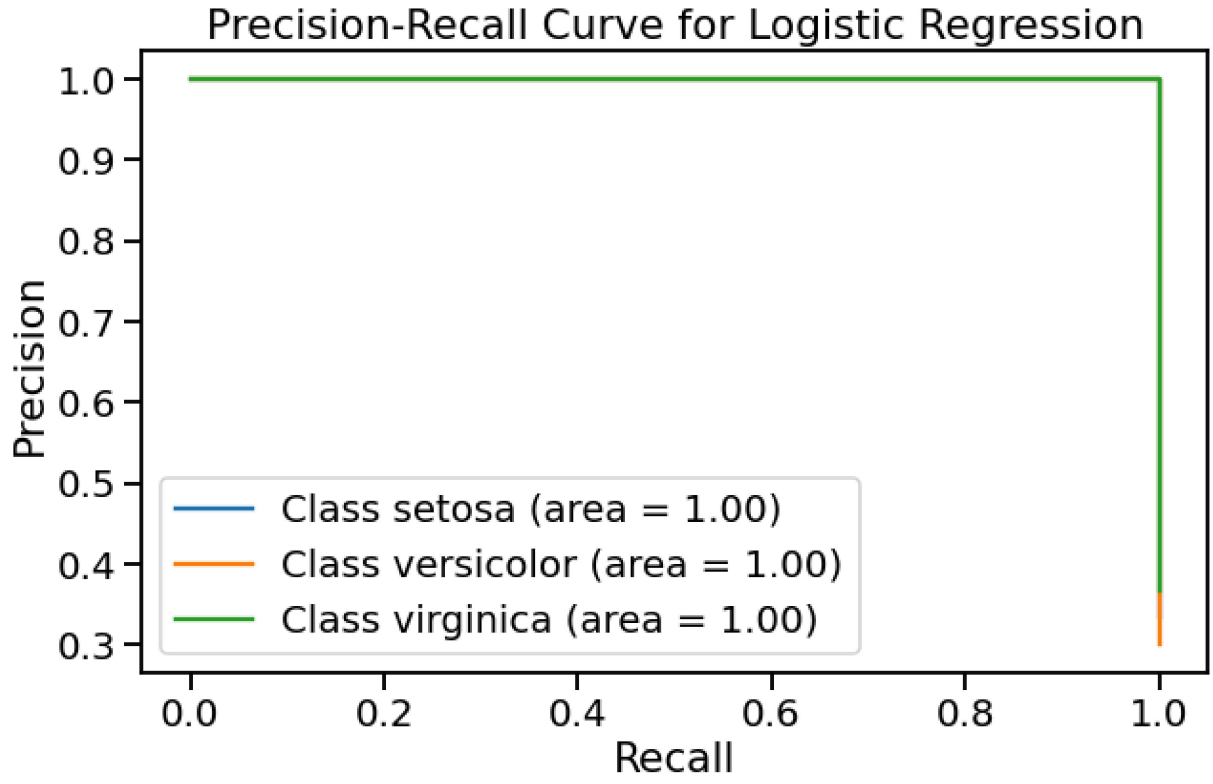
In [22]:
```
# 4. Feature Importance using Permutation (for Random Forest)
# Permutation importance is model-agnostic and gives insight into how each feature
perm_importance = permutation_importance(random_forest, X_test, y_test, n_repeats=1
sorted_idx = perm_importance.importances_mean.argsort()

plt.figure(figsize=(10, 6))
plt.barh(np.array(iris.feature_names)[sorted_idx], perm_importance.importances_mean
plt.title("Permutation Feature Importance (Random Forest)")
plt.xlabel("Mean Importance")
plt.ylabel("Feature")
plt.show()
```

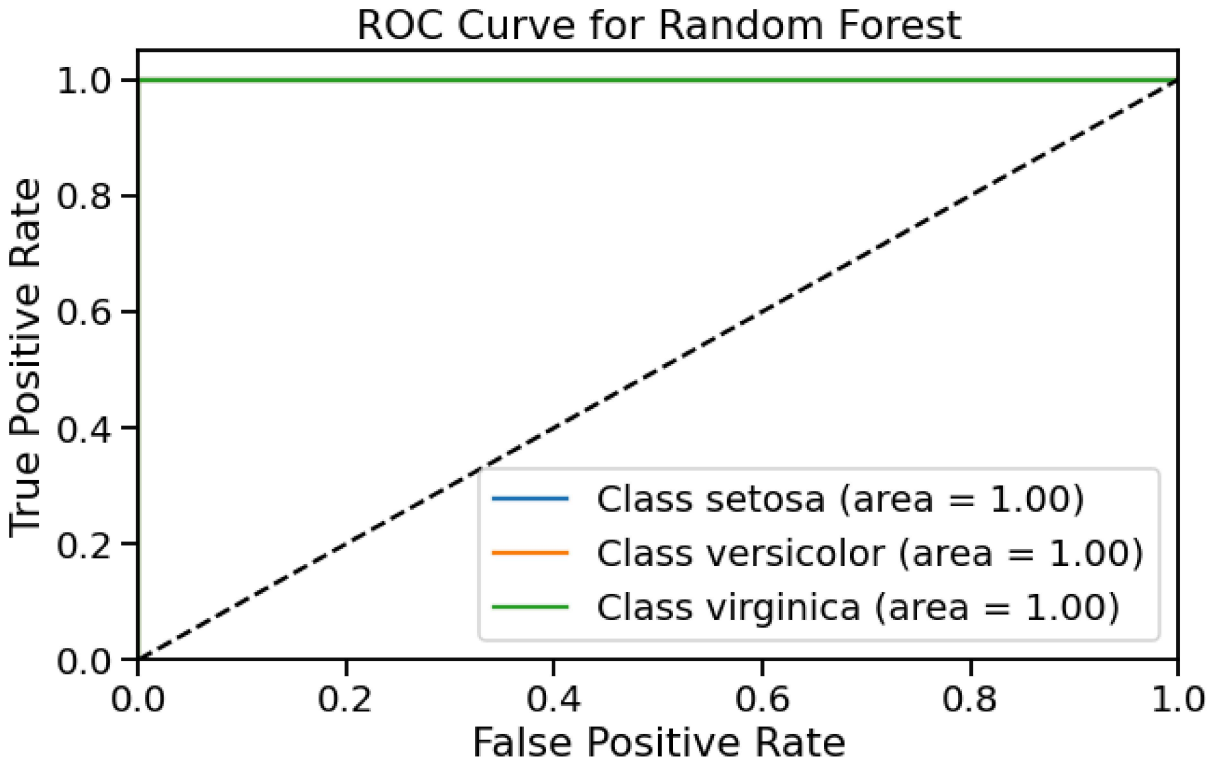


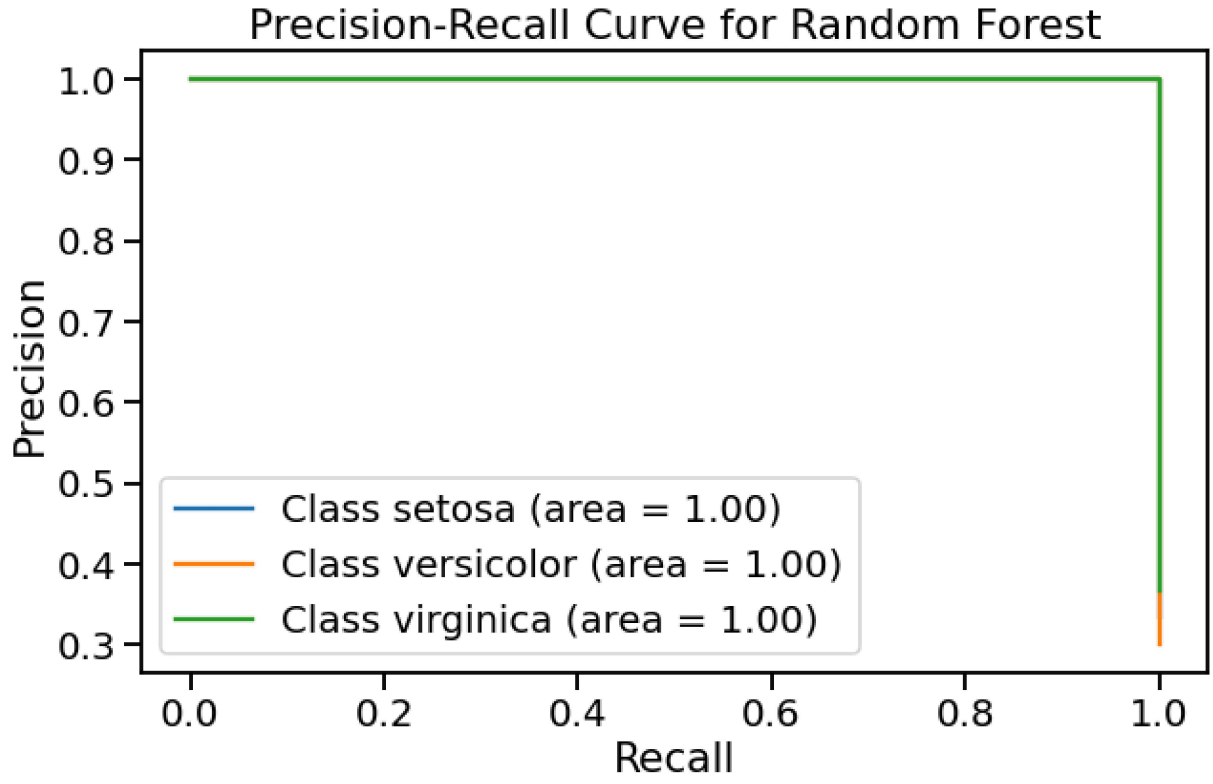
In [26]:
```python
# Import necessary libraries for additional visualizations
from sklearn.metrics import precision_recall_curve, roc_curve, roc_auc_score, auc
from sklearn.preprocessing import label_binarize

# Function to plot additional visualizations
def plot_additional_metrics(model, X_test, y_test, model_name):
    # Predict probabilities for ROC curve and Precision-Recall curve
    y_prob = model.predict_proba(X_test)
    y_pred = model.predict(X_test)

    # Binarize the output for multi-class ROC and Precision-Recall curves
    y_test_bin = label_binarize(y_test, classes=[0, 1, 2])

    # Plot ROC Curve
    plt.figure(figsize=(10, 6))
    fpr = {}
    tpr = {}
    roc_auc = {}
    for i in range(y_test_bin.shape[1]):
        fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_prob[:, i])
        roc_auc[i] = roc_auc_score(y_test_bin[:, i], y_prob[:, i])
        plt.plot(fpr[i], tpr[i], label=f'Class {iris.target_names[i]} (area = {roc_

    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(f'ROC Curve for {model_name}')
    plt.legend(loc='lower right')
    plt.show()

    # Plot Precision-Recall Curve
    plt.figure(figsize=(10, 6))
    precision = {}
    recall = {}
```

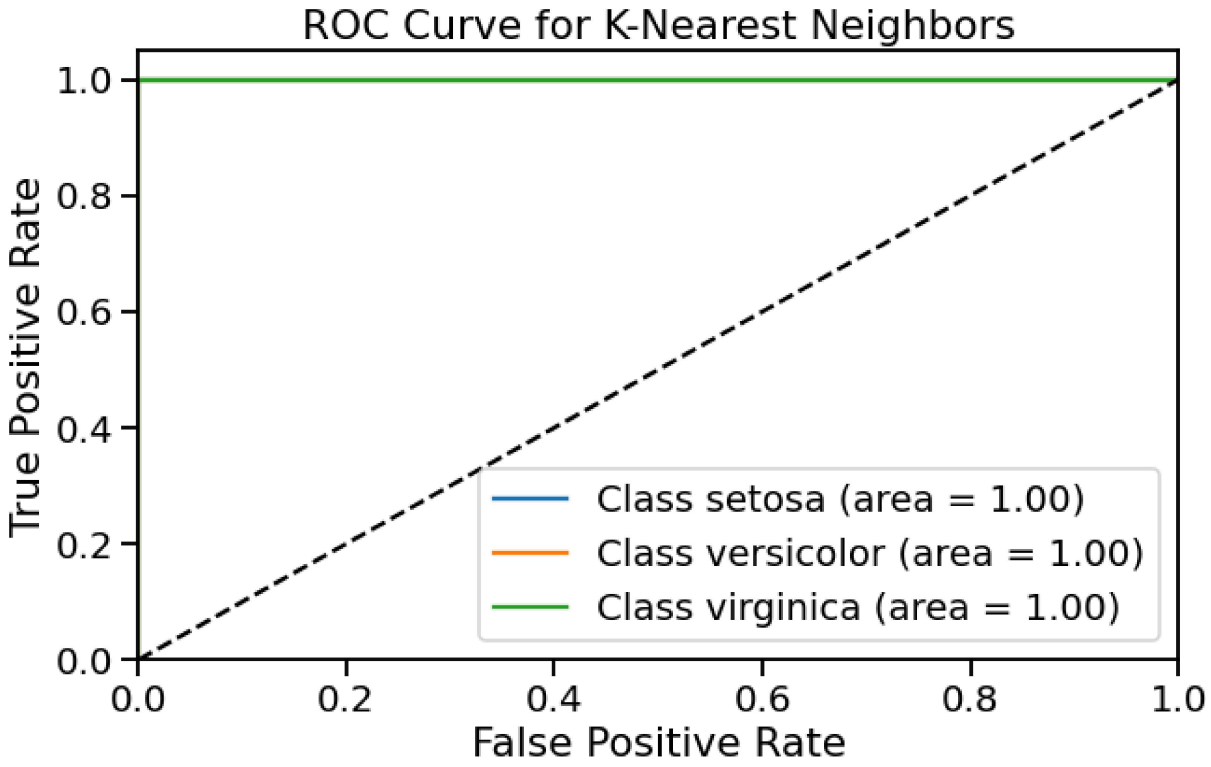
```
    pr_auc = {}
    for i in range(y_test_bin.shape[1]):
        precision[i], recall[i], _ = precision_recall_curve(y_test_bin[:, i], y_pro
        pr_auc[i] = auc(recall[i], precision[i])
        plt.plot(recall[i], precision[i], label=f'Class {iris.target_names[i]} (are

    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title(f'Precision-Recall Curve for {model_name}')
    plt.legend(loc='lower left')
    plt.show()

# Load dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Initialize and fit models
logreg = LogisticRegression(max_iter=200)
logreg.fit(X_train, y_train)

random_forest = RandomForestClassifier(random_state=42)
random_forest.fit(X_train, y_train)

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)

svm = SVC(probability=True)
svm.fit(X_train, y_train)

xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
xgb.fit(X_train, y_train)

# Plot additional metrics for each model
plot_additional_metrics(logreg, X_test, y_test, "Logistic Regression")
plot_additional_metrics(random_forest, X_test, y_test, "Random Forest")
plot_additional_metrics(knn, X_test, y_test, "K-Nearest Neighbors")
plot_additional_metrics(svm, X_test, y_test, "Support Vector Machine")
plot_additional_metrics(xgb, X_test, y_test, "XGBoost")
```

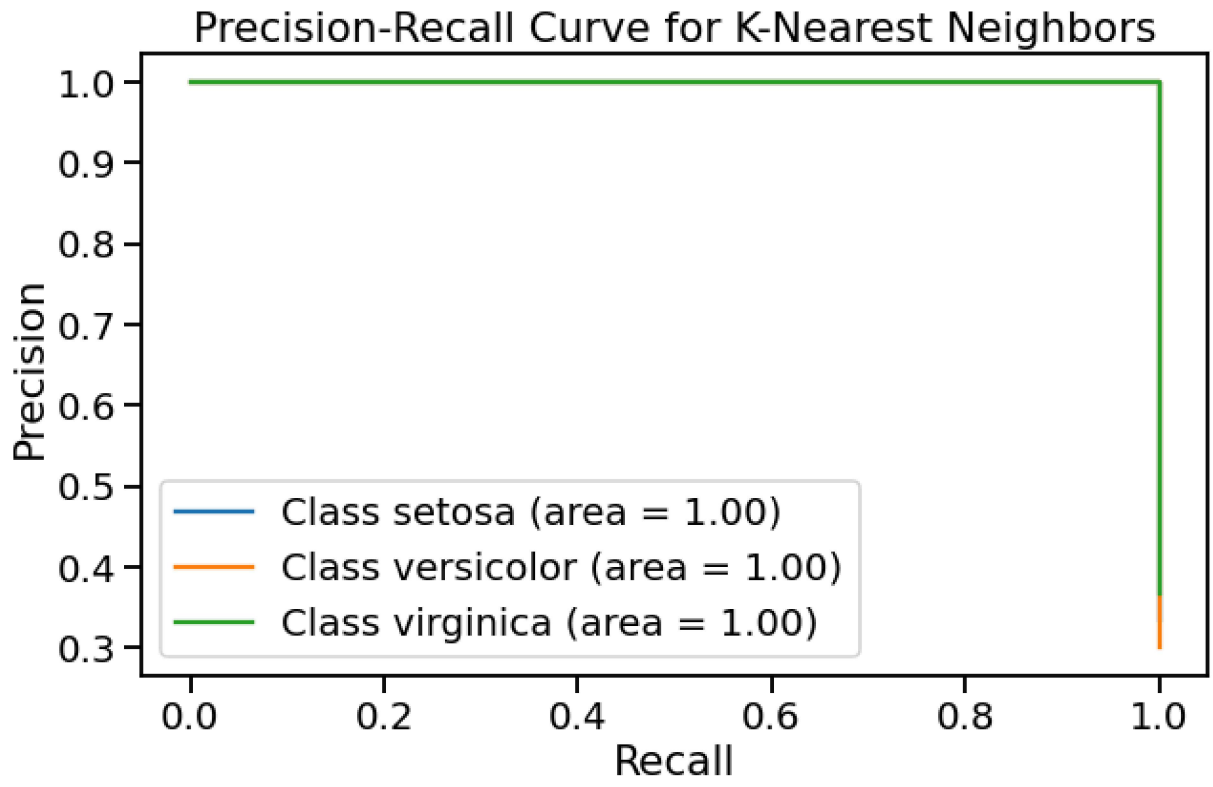

ROC Curve for Logistic Regression
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class setosa (area = 1.00)
Class versicolor (area = 1.00)
Class virginica (area = 1.00)

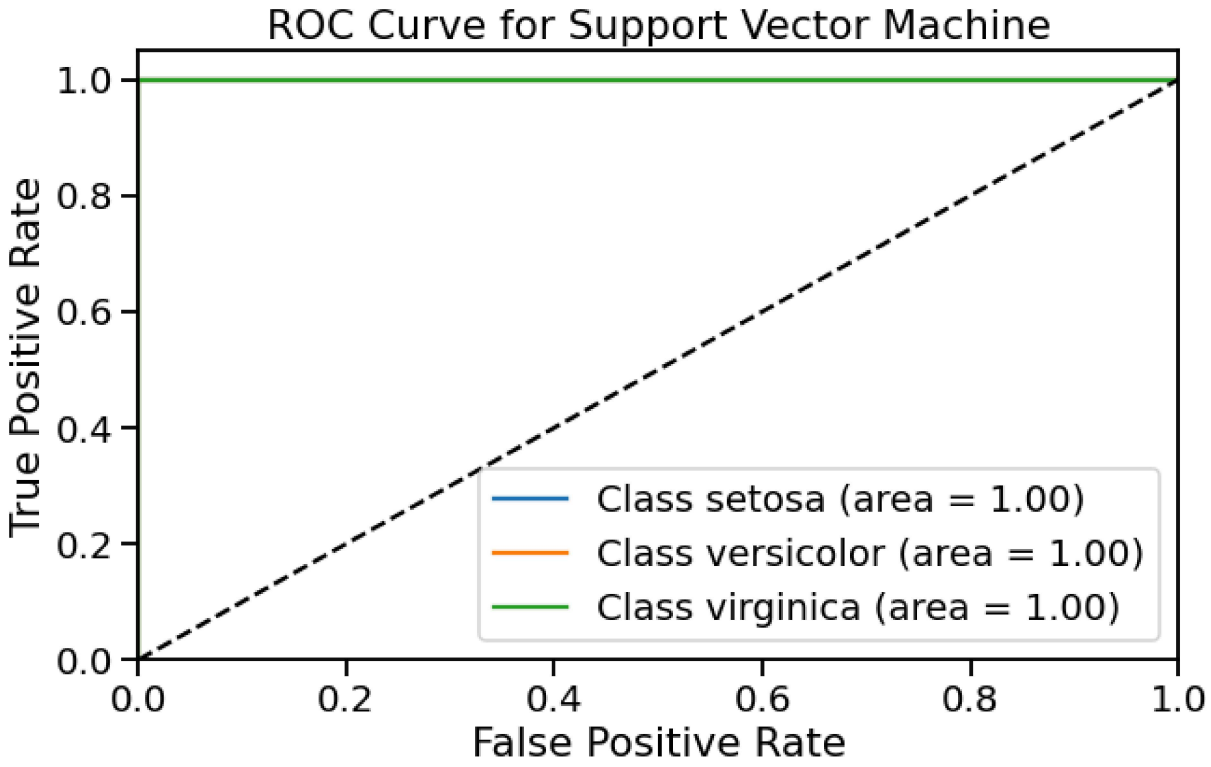

Precision-Recall Curve for Logistic Regression
Precision
Recall
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Class setosa (area = 1.00)
Class versicolor (area = 1.00)
Class virginica (area = 1.00)

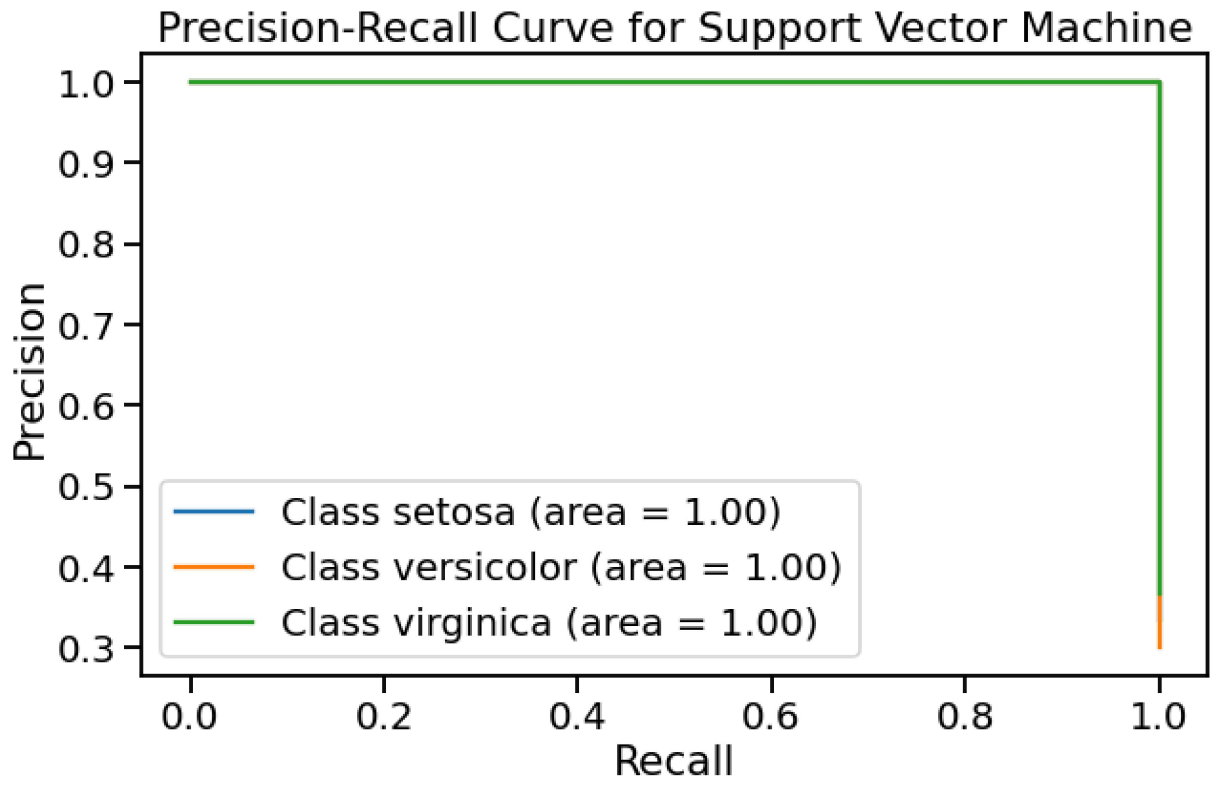

ROC Curve for Random Forest
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class setosa (area = 1.00)
Class versicolor (area = 1.00)
Class virginica (area = 1.00)

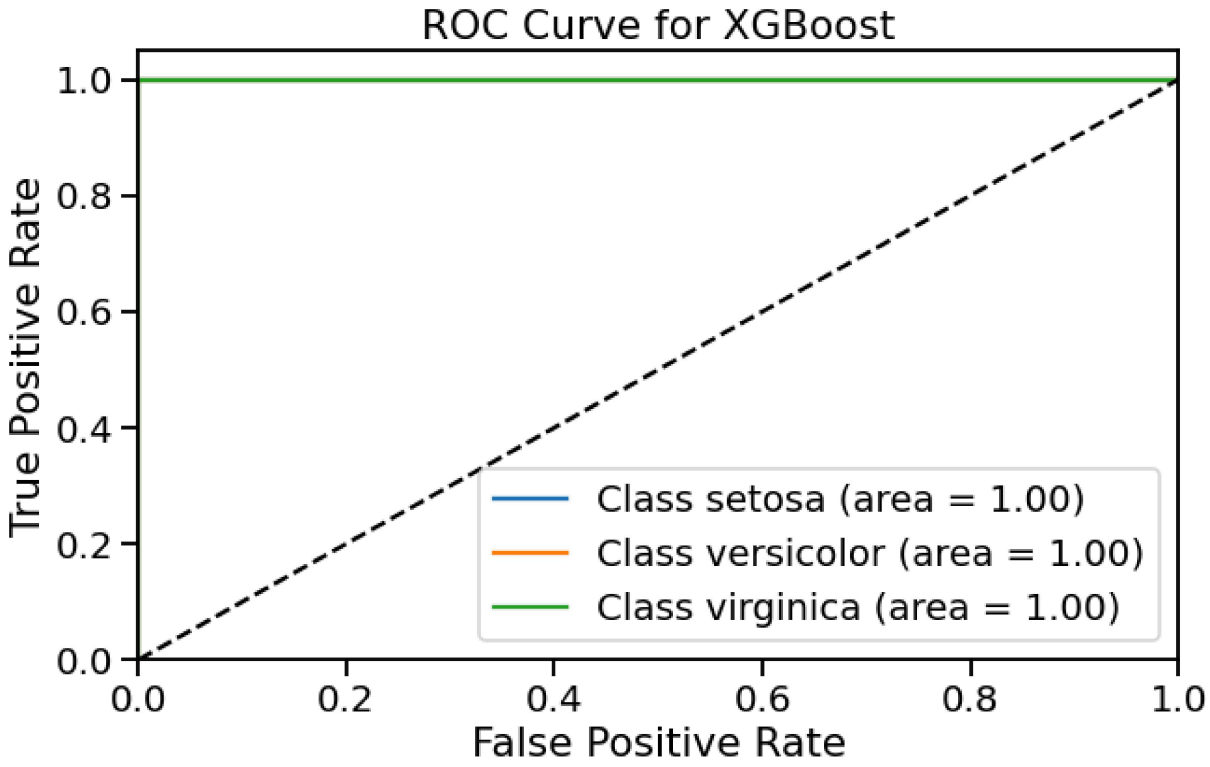

Precision-Recall Curve for Random Forest
Precision
Recall
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Class setosa (area = 1.00)
Class versicolor (area = 1.00)
Class virginica (area = 1.00)

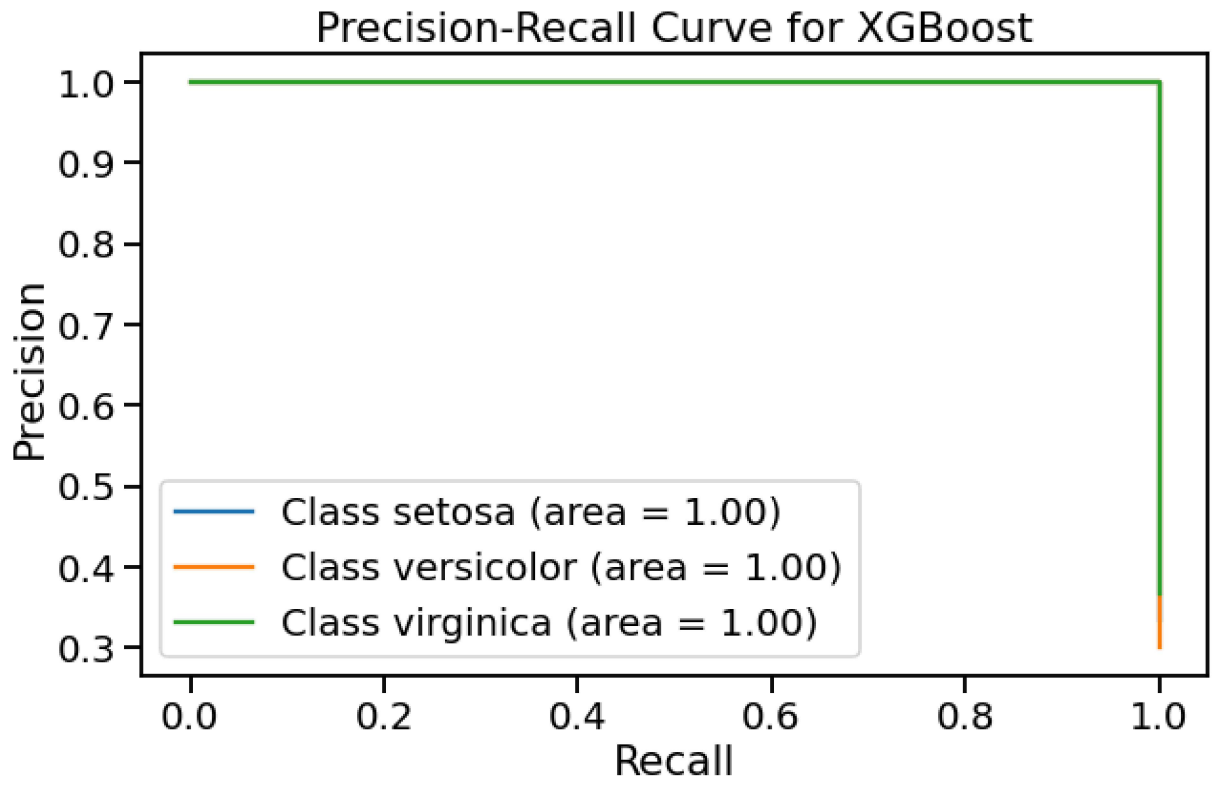

ROC Curve for K-Nearest Neighbors
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class setosa (area = 1.00)
Class versicolor (area = 1.00)
Class virginica (area = 1.00)


Precision-Recall Curve for K-Nearest Neighbors
Precision
Recall
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Class setosa (area = 1.00)
Class versicolor (area = 1.00)
Class virginica (area = 1.00)


ROC Curve for Support Vector Machine
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class setosa (area = 1.00)
Class versicolor (area = 1.00)
Class virginica (area = 1.00)

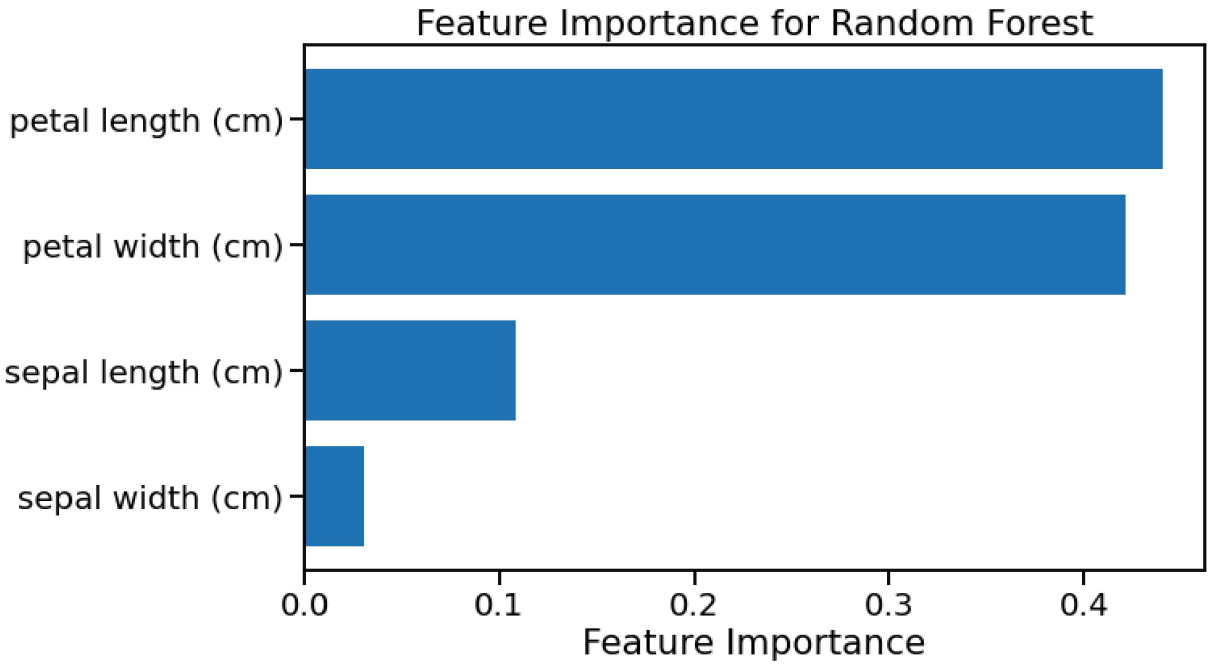

Precision-Recall Curve for Support Vector Machine
Precision
Recall
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Class setosa (area = 1.00)
Class versicolor (area = 1.00)
Class virginica (area = 1.00)

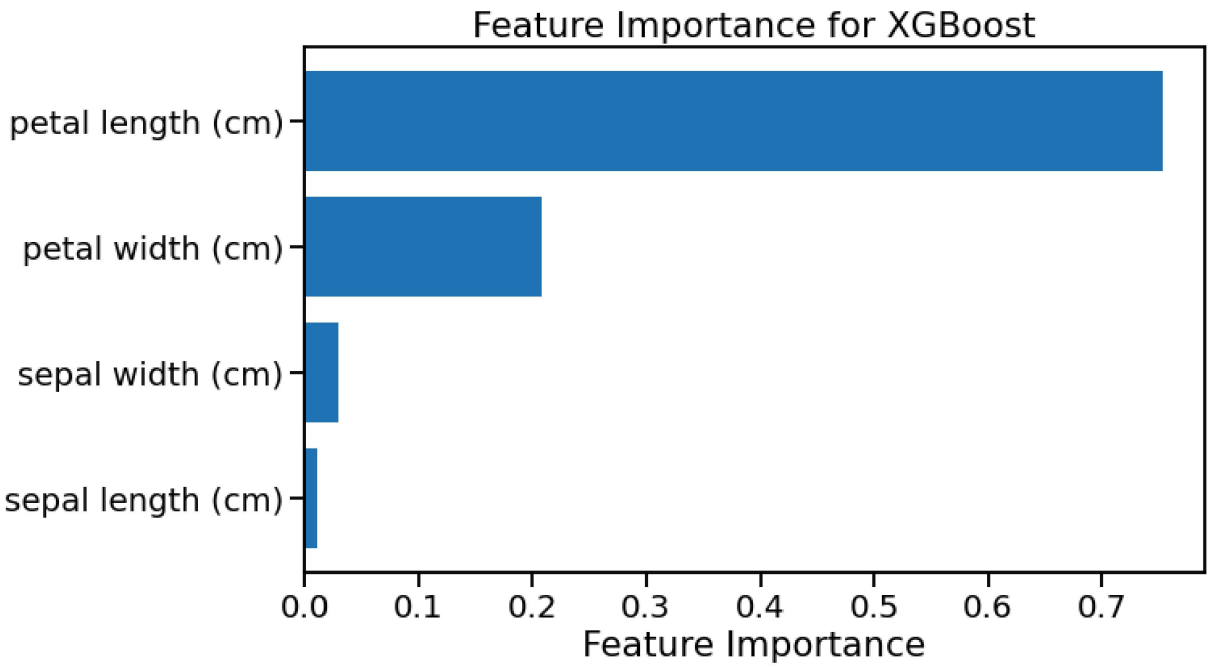




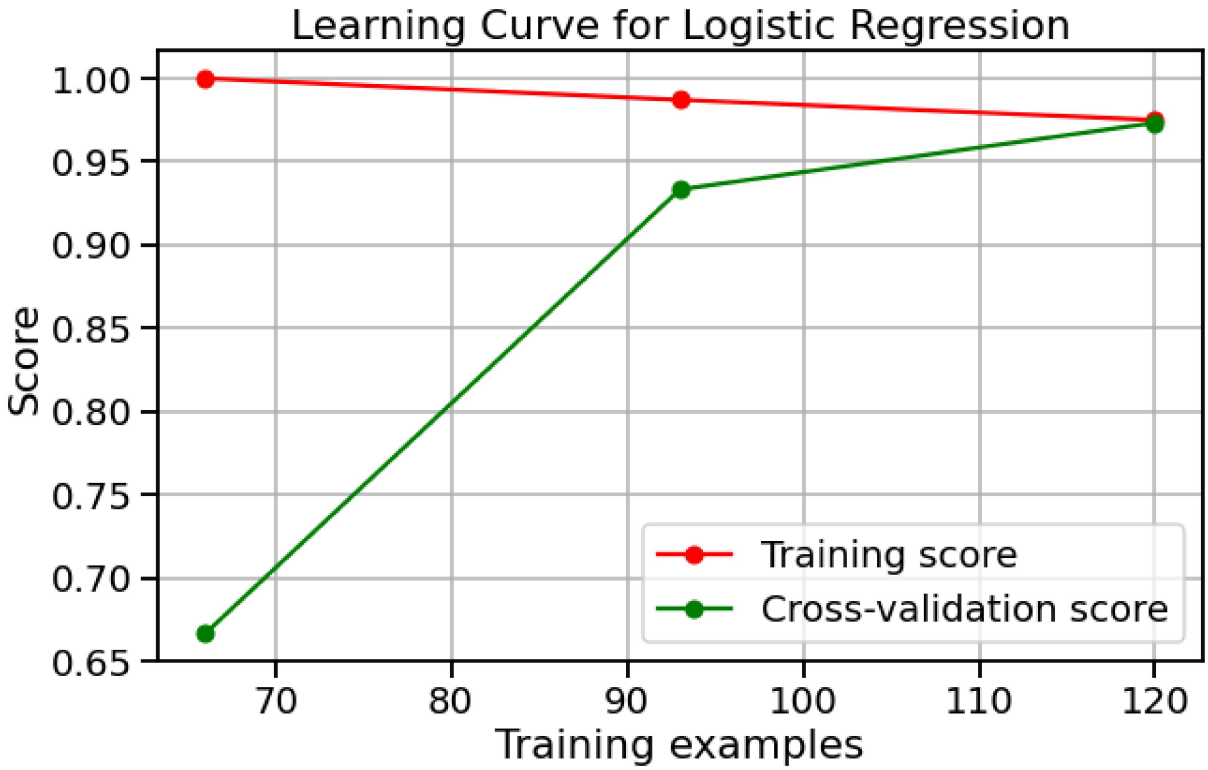
In [27]:
```python
import numpy as np

def plot_feature_importance(model, feature_names, model_name):
    if hasattr(model, 'feature_importances_'):
        importances = model.feature_importances_
    elif hasattr(model, 'booster_'):
        importances = model.booster_.feature_importances_
    else:
```

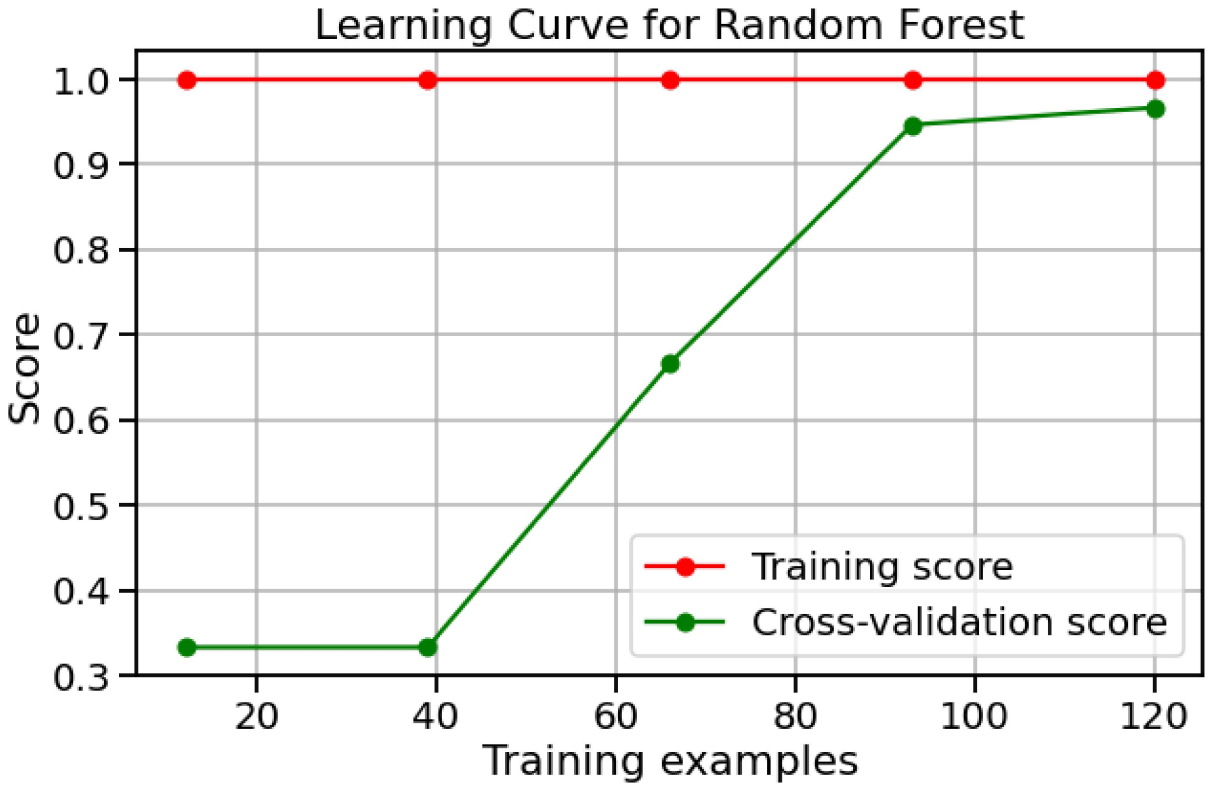
```
        print(f"{model_name} does not have feature importances.")
        return

    indices = np.argsort(importances)

    plt.figure(figsize=(10, 6))
    plt.title(f"Feature Importance for {model_name}")
    plt.barh(range(len(indices)), importances[indices], align='center')
    plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
    plt.xlabel('Feature Importance')
    plt.show()

# Assuming feature names
feature_names = iris.feature_names

# Plot feature importance for Random Forest and XGBoost
plot_feature_importance(random_forest, feature_names, "Random Forest")
plot_feature_importance(xgb, feature_names, "XGBoost")
```

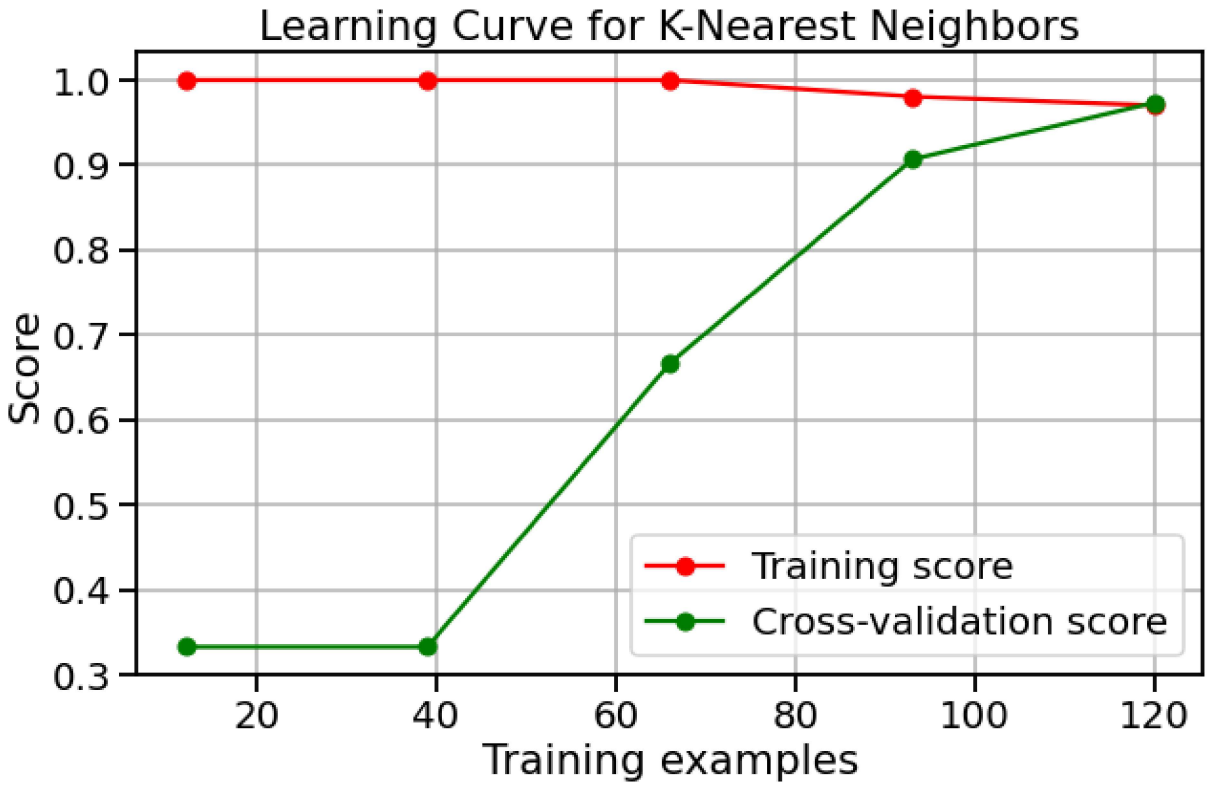


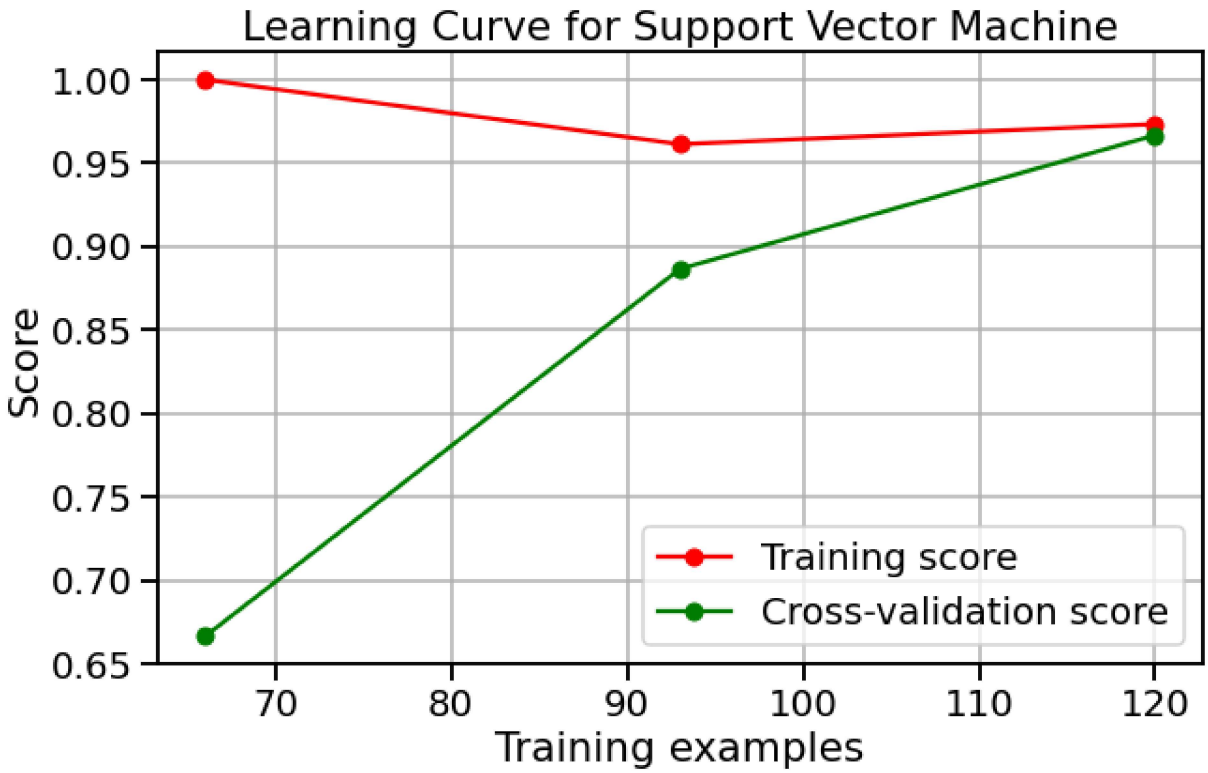


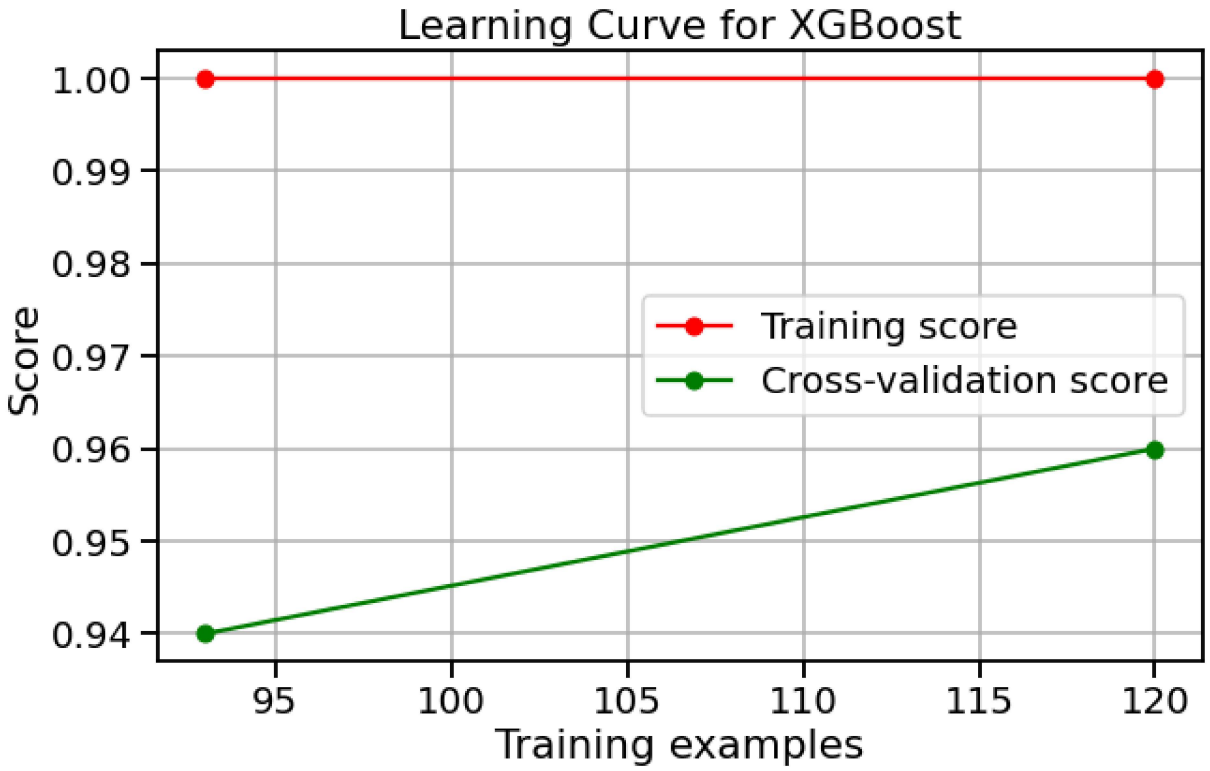
In [28]:
```python
from sklearn.model_selection import learning_curve

def plot_learning_curve(model, X, y, model_name):
    train_sizes, train_scores, test_scores = learning_curve(
        model, X, y, cv=5, n_jobs=-1, train_sizes=np.linspace(0.1, 1.0, 5)
    )

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

    plt.figure(figsize=(10, 6))
    plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training score')
    plt.plot(train_sizes, test_mean, 'o-', color='g', label='Cross-validation score
    plt.xlabel('Training examples')
    plt.ylabel('Score')
    plt.title(f'Learning Curve for {model_name}')
    plt.legend(loc='best')
    plt.grid()
    plt.show()

# Plot learning curves for each model
plot_learning_curve(logreg, X, y, "Logistic Regression")
plot_learning_curve(random_forest, X, y, "Random Forest")
plot_learning_curve(knn, X, y, "K-Nearest Neighbors")
plot_learning_curve(svm, X, y, "Support Vector Machine")
plot_learning_curve(xgb, X, y, "XGBoost")
```


Learning Curve for Logistic Regression
Score
Training examples
Training score
Cross-validation score
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
70
80
90
100
110
120


Learning Curve for Random Forest
Score
Training examples
Training score
Cross-validation score
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
20
40
60
80
100
120

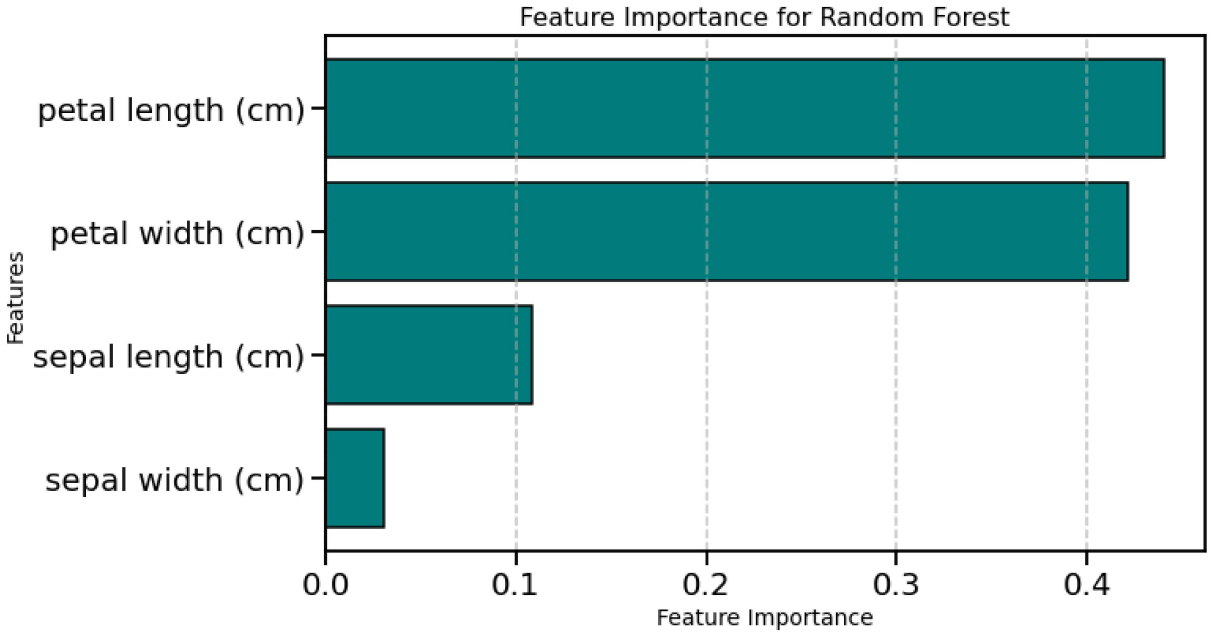

Learning Curve for K-Nearest Neighbors
Score
Training examples
Training score
Cross-validation score
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
20
40
60
80
100
120

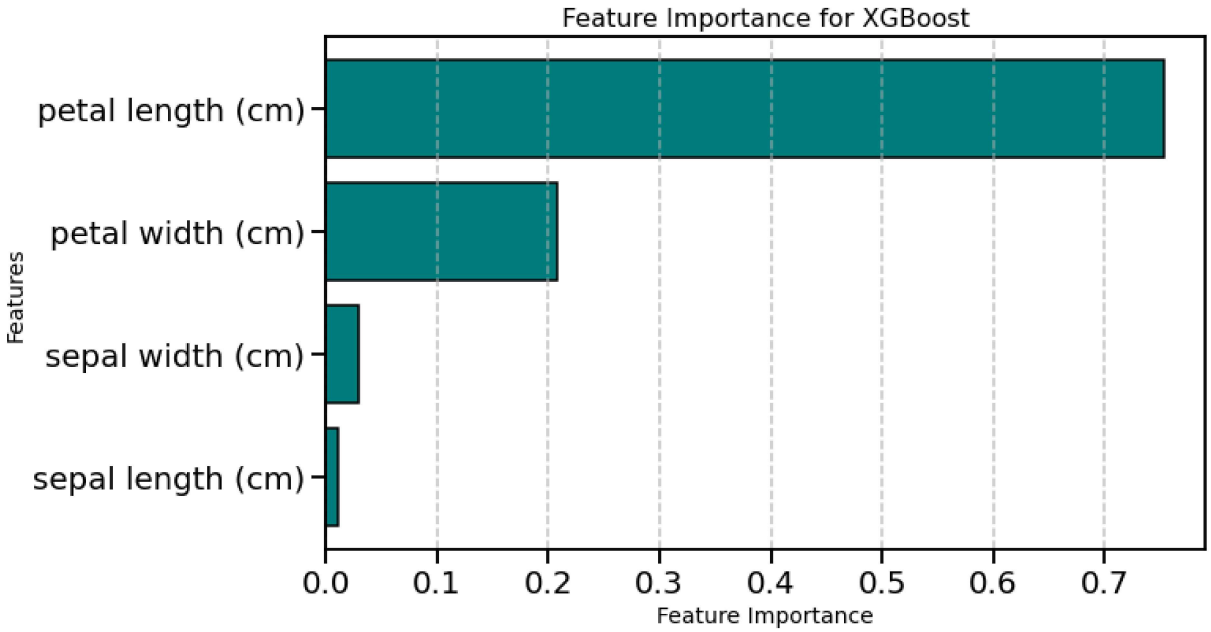

Learning Curve for Support Vector Machine
Score
Training examples
Training score
Cross-validation score
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
70
80
90
100
110
120



In [29]:

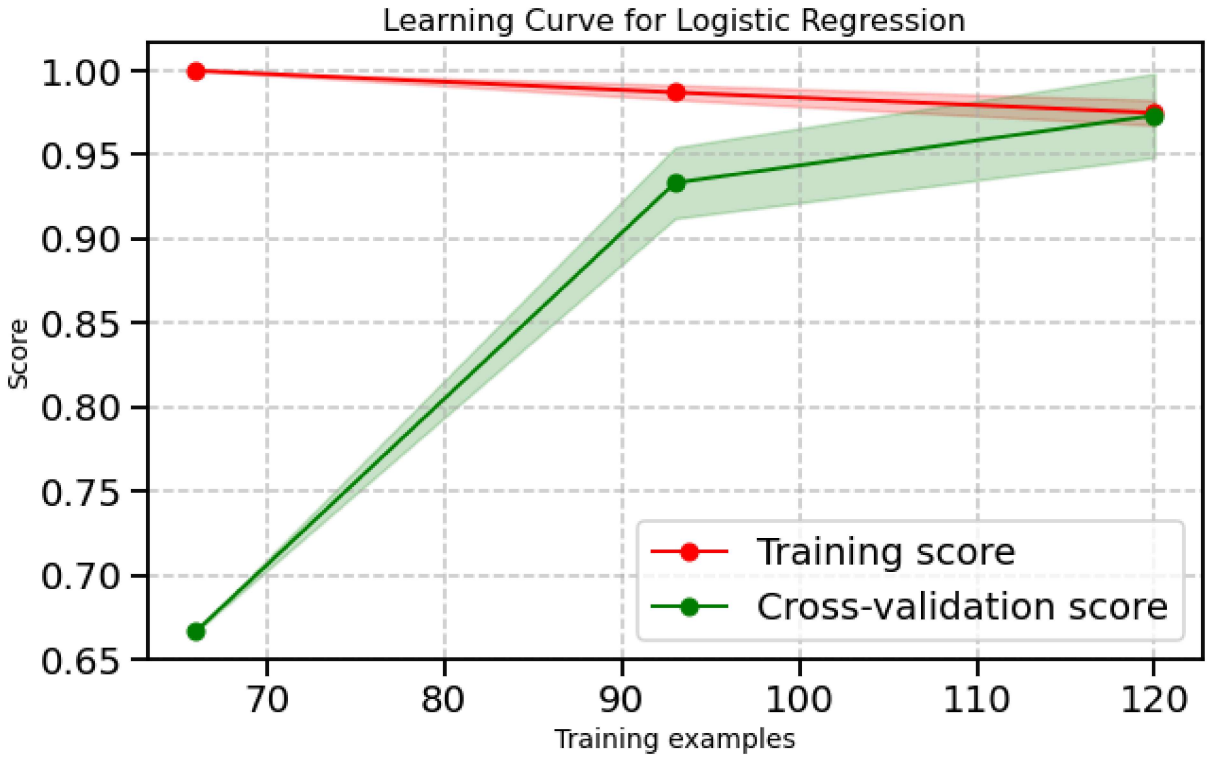
```python
import numpy as np
import matplotlib.pyplot as plt

def plot_feature_importance(model, feature_names, model_name):
    if hasattr(model, 'feature_importances_'):
        importances = model.feature_importances_
    elif hasattr(model, 'booster_'):
        importances = model.booster_.feature_importances_
    else:
        print(f"{model_name} does not have feature importances.")
        return

    indices = np.argsort(importances)

    plt.figure(figsize=(10, 6))
    plt.title(f"Feature Importance for {model_name}", fontsize=16)
    plt.barh(range(len(indices)), importances[indices], align='center', color='teal
    plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
    plt.xlabel('Feature Importance', fontsize=14)
    plt.ylabel('Features', fontsize=14)
    plt.grid(axis='x', linestyle='--', alpha=0.7)
    plt.show()

# Assuming feature names
feature_names = iris.feature_names

# Plot feature importance for Random Forest and XGBoost
plot_feature_importance(random_forest, feature_names, "Random Forest")
plot_feature_importance(xgb, feature_names, "XGBoost")
```

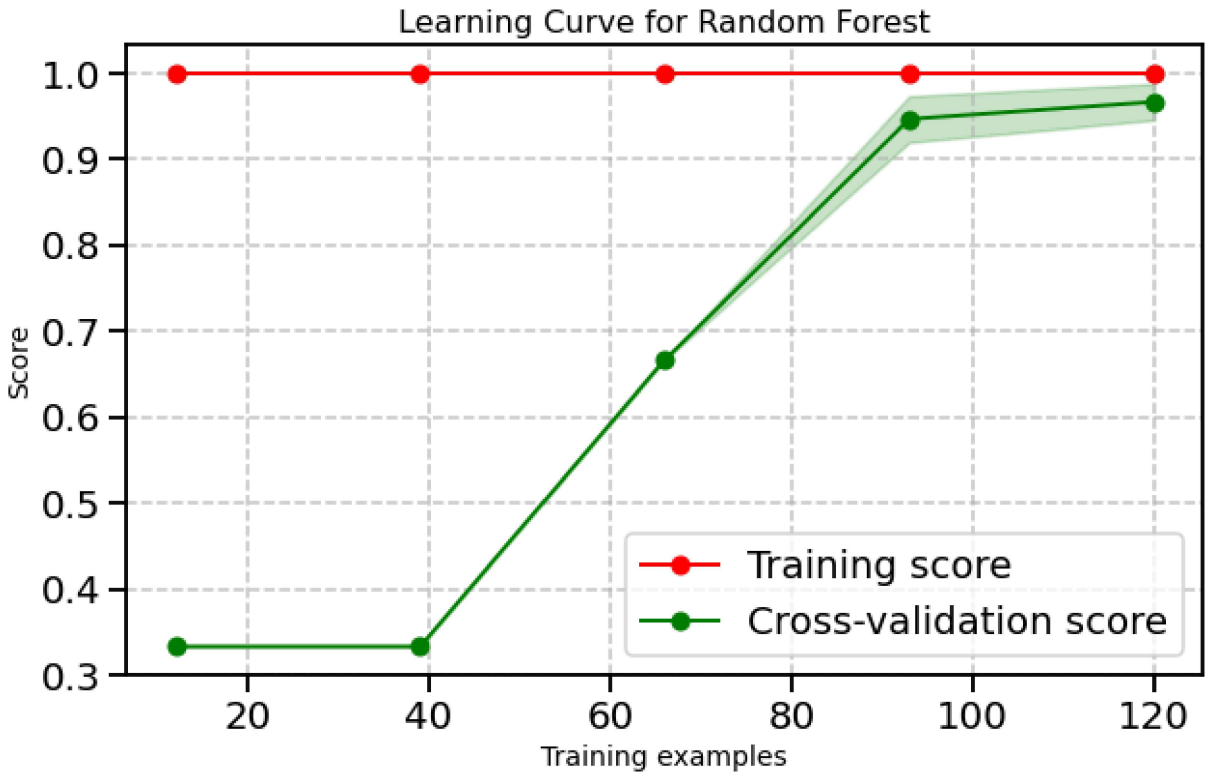


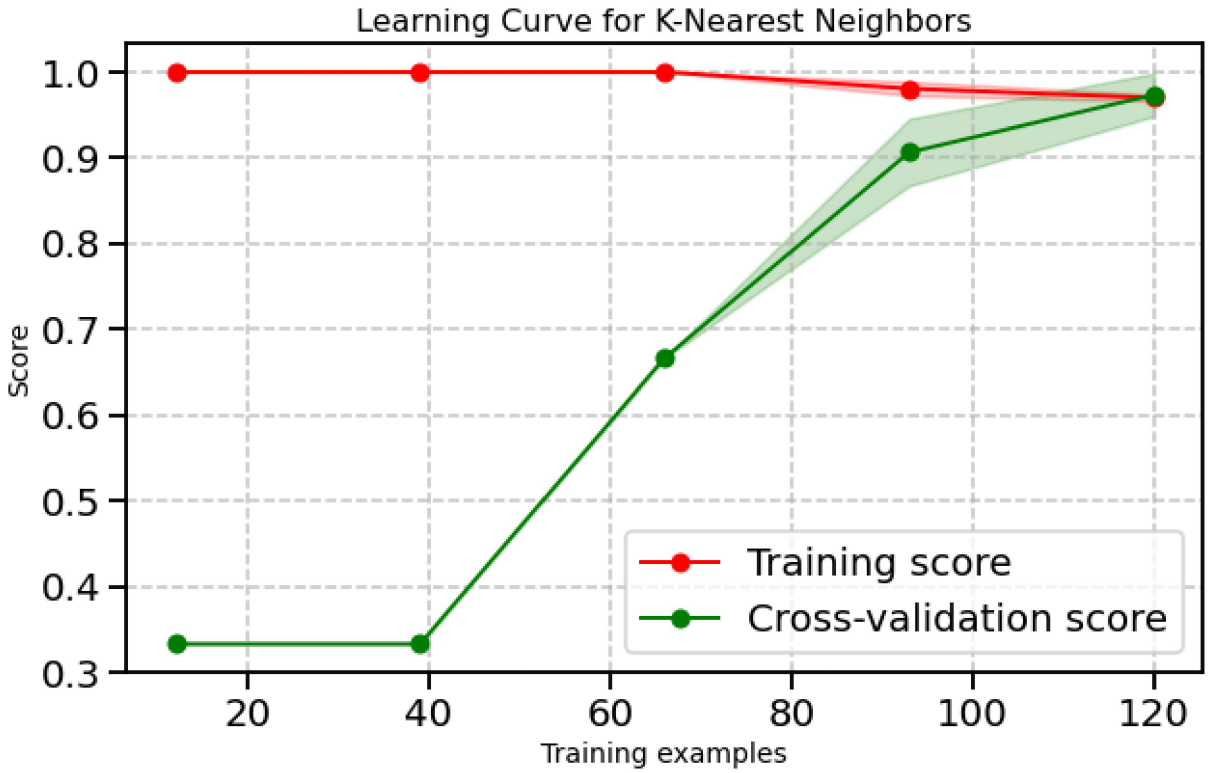


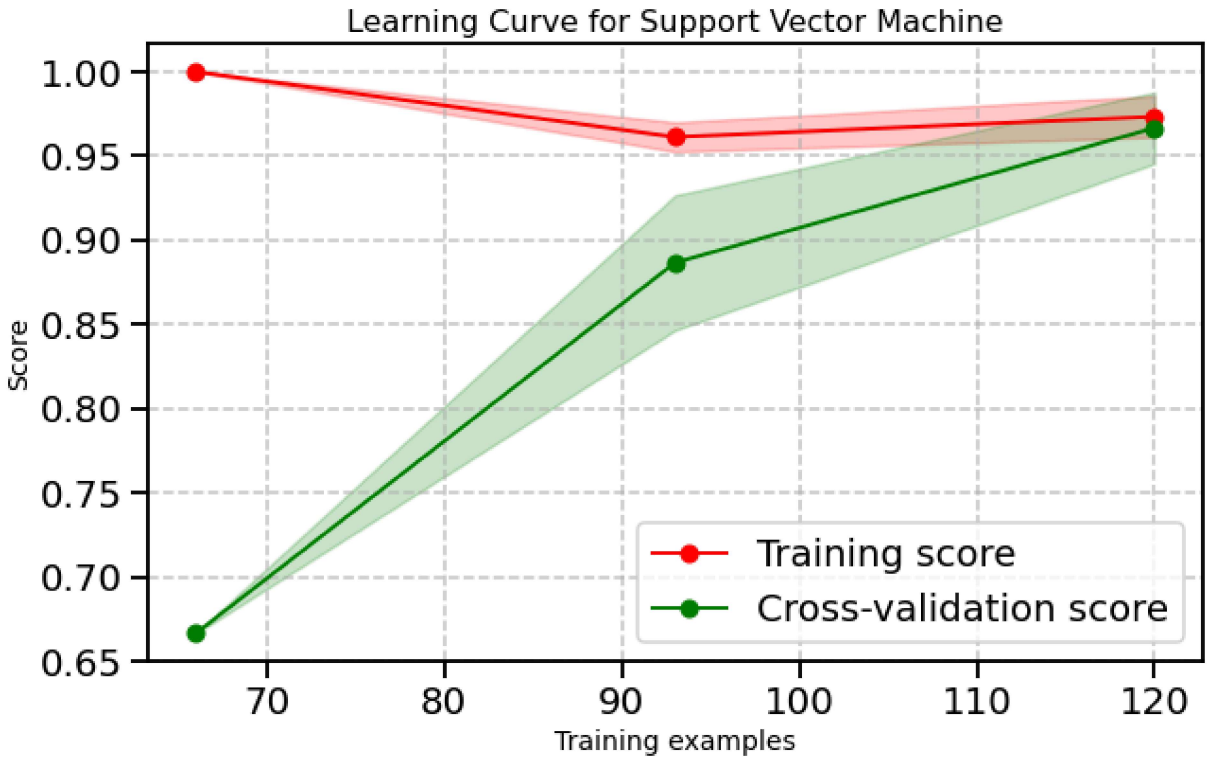
In [30]:
```
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt
import numpy as np

def plot_learning_curve(model, X, y, model_name):
    train_sizes, train_scores, test_scores = learning_curve(
        model, X, y, cv=5, n_jobs=-1, train_sizes=np.linspace(0.1, 1.0, 5)
    )

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

    plt.figure(figsize=(10, 6))
    plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training score', line
    plt.plot(train_sizes, test_mean, 'o-', color='g', label='Cross-validation score
```

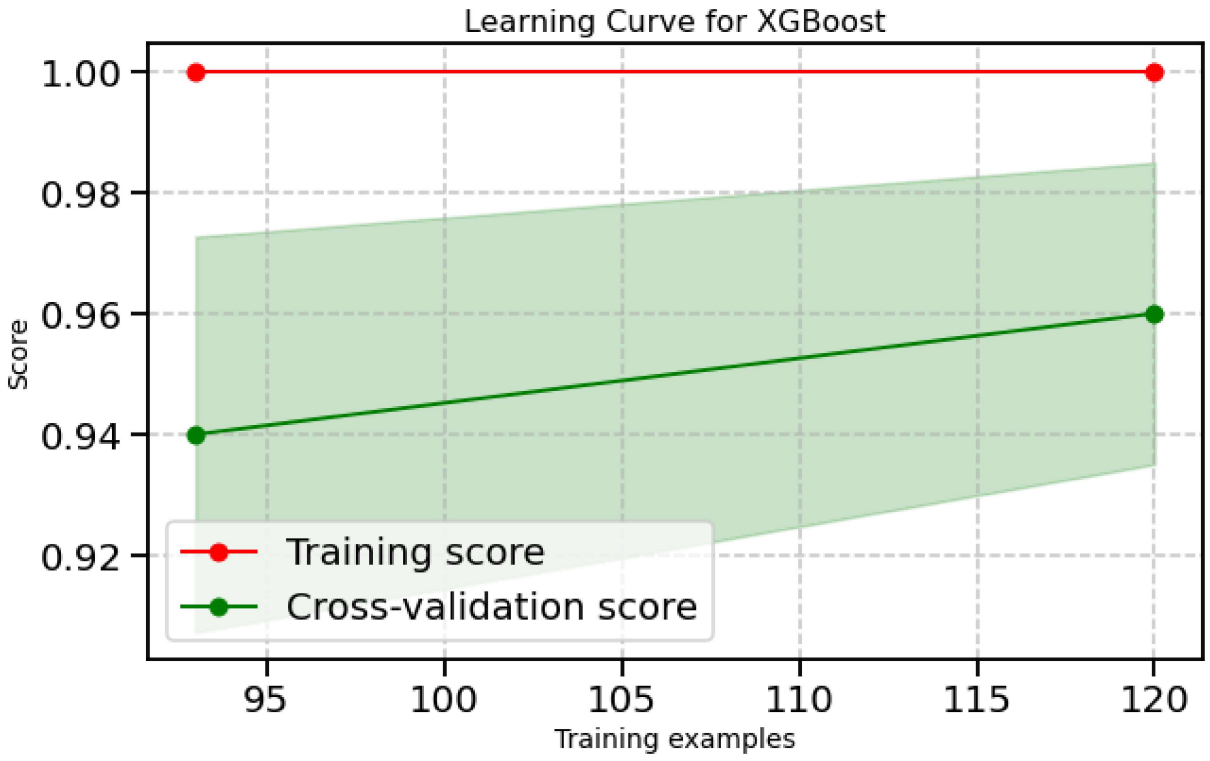
```
    plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, a
    plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha
    plt.xlabel('Training examples', fontsize=14)
    plt.ylabel('Score', fontsize=14)
    plt.title(f'Learning Curve for {model_name}', fontsize=16)
    plt.legend(loc='best')
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.show()

# Plot learning curves for each model
plot_learning_curve(logreg, X, y, "Logistic Regression")
plot_learning_curve(random_forest, X, y, "Random Forest")
plot_learning_curve(knn, X, y, "K-Nearest Neighbors")
plot_learning_curve(svm, X, y, "Support Vector Machine")
plot_learning_curve(xgb, X, y, "XGBoost")
```




Learning Curve for Random Forest
Score
Training examples
Training score
Cross-validation score
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
20
40
60
80
100
120


Learning Curve for K-Nearest Neighbors
Score
Training examples
Training score
Cross-validation score
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
20
40
60
80
100
120


Learning Curve for Support Vector Machine
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
Score
70
80
90
100
110
120
Training examples
Training score
Cross-validation score


Learning Curve for XGBoost
1.00
0.98
0.96
0.94
0.92
Score
95
100
105
110
115
120
Training examples
Training score
Cross-validation score

In [ ]: