In [27]:

```
!pip install graphviz
```

```
Requirement already satisfied: graphviz in /Users/ghitabouzida/opt/anacon
da3/lib/python3.9/site-packages (0.20.1)
```

In [30]:

```
import pandas as pd
# Assuming the data is stored in a CSV file called 'titanic_data.csv'
data = pd.read_csv('titanic_dataset.csv')
# Let's first check for missing data
print(data.isnull().sum())
# Drop unnecessary columns (such as PassengerId, Name, Ticket, and Cabi
data = data.drop(columns=['PassengerId', 'Name', 'Ticket', 'Cabin'])
# Handle missing values in Age and Embarked columns (you may choose to
data['Age'].fillna(data['Age'].mean(), inplace=True)
data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)
# Convert categorical features into numerical representations using one
data = pd.get_dummies(data, columns=['Sex', 'Embarked'], drop_first=Tru
# Separate the target variable (Survived) from the features
X = data.drop(columns=['Survived'])
y = data['Survived']
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [31]:

```
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets (80% for training, 20%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
```

In [48]:

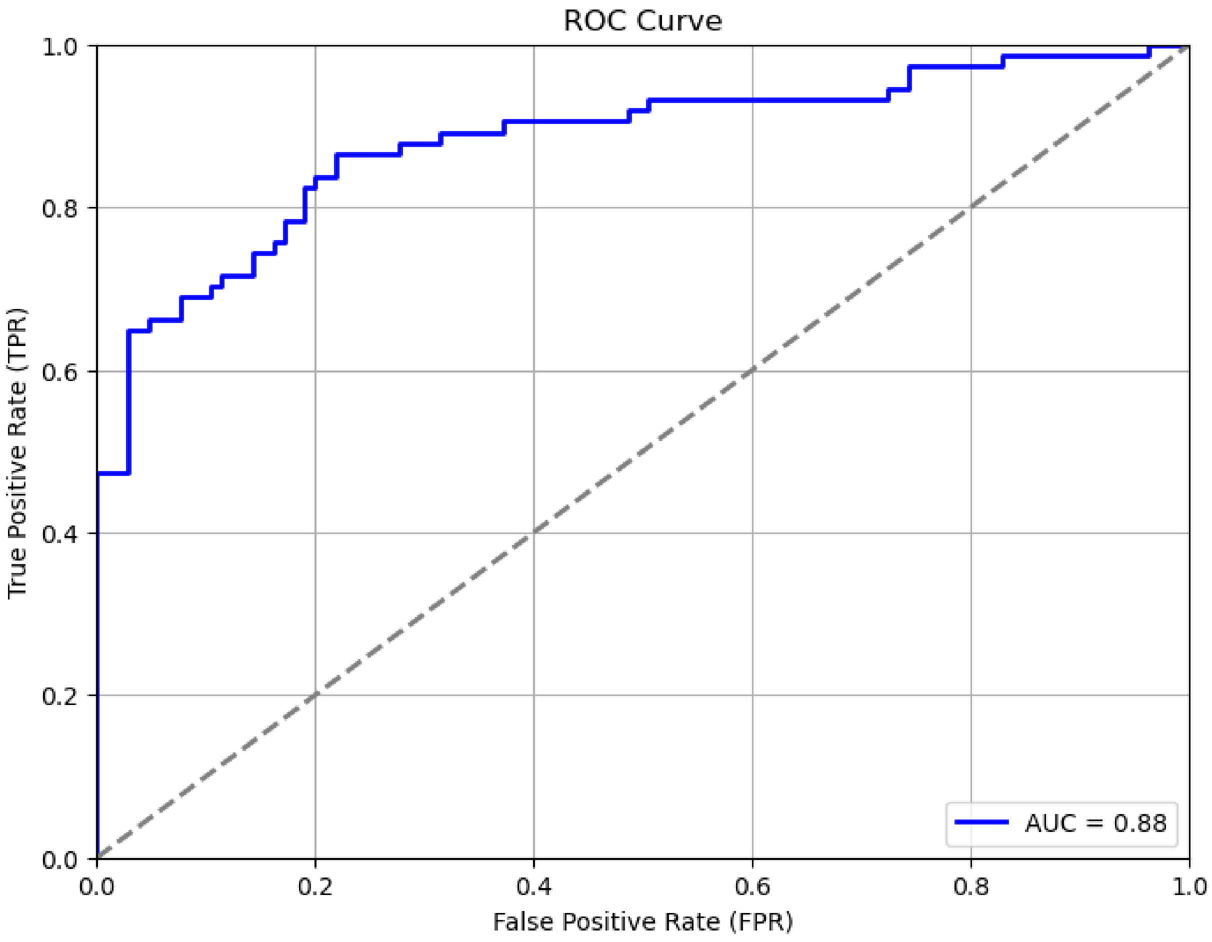
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, conf
import numpy as np
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree


# Assuming the data is stored in a CSV file called 'titanic_data.csv'
data = pd.read_csv('titanic_dataset.csv')

# Drop unnecessary columns (such as PassengerId, Name, Ticket, and Cabi
data = data.drop(columns=['PassengerId', 'Name', 'Ticket', 'Cabin'])

# Handle missing values in Age and Embarked columns (you may choose to
data['Age'].fillna(data['Age'].mean(), inplace=True)
data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

# Convert categorical features into numerical representations using one
data = pd.get_dummies(data, columns=['Sex', 'Embarked'], drop_first=Tru

# Separate the target variable (Survived) from the features
X = data.drop(columns=['Survived'])
y = data['Survived']

# Split the data into training and testing sets (80% for training, 20%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create a pipeline with feature scaling and logistic regression
pipe = make_pipeline(StandardScaler(), LogisticRegression(random_state=

# Fit the pipeline on the training data (feature scaling is applied)
pipe.fit(X_train, y_train)

# Make predictions on the test set
y_pred = pipe.predict(X_test)

# Evaluate the model
from sklearn.metrics import accuracy_score, classification_report, conf

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Get a classification report
print(classification_report(y_test, y_pred))

# Get the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
```

```python
print("Confusion Matrix:")
print(conf_matrix)

```

```
Accuracy: 0.8100558659217877
              precision    recall  f1-score   support

           0       0.83      0.86      0.84       105
           1       0.79      0.74      0.76        74

    accuracy                           0.81       179
   macro avg       0.81      0.80      0.80       179
weighted avg       0.81      0.81      0.81       179

Confusion Matrix:
[[90 15]
 [19 55]]
```

In [45]:

```
# Get the predicted probabilities for the positive class (survived) frc
y_pred_prob = pipe.predict_proba(X_test)[:, 1]

# Calculate the ROC curve values: FPR, TPR, and the corresponding thres
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Calculate the AUC score
auc_score = roc_auc_score(y_test, y_pred_prob)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'AUC = {auc_score:.2f}')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



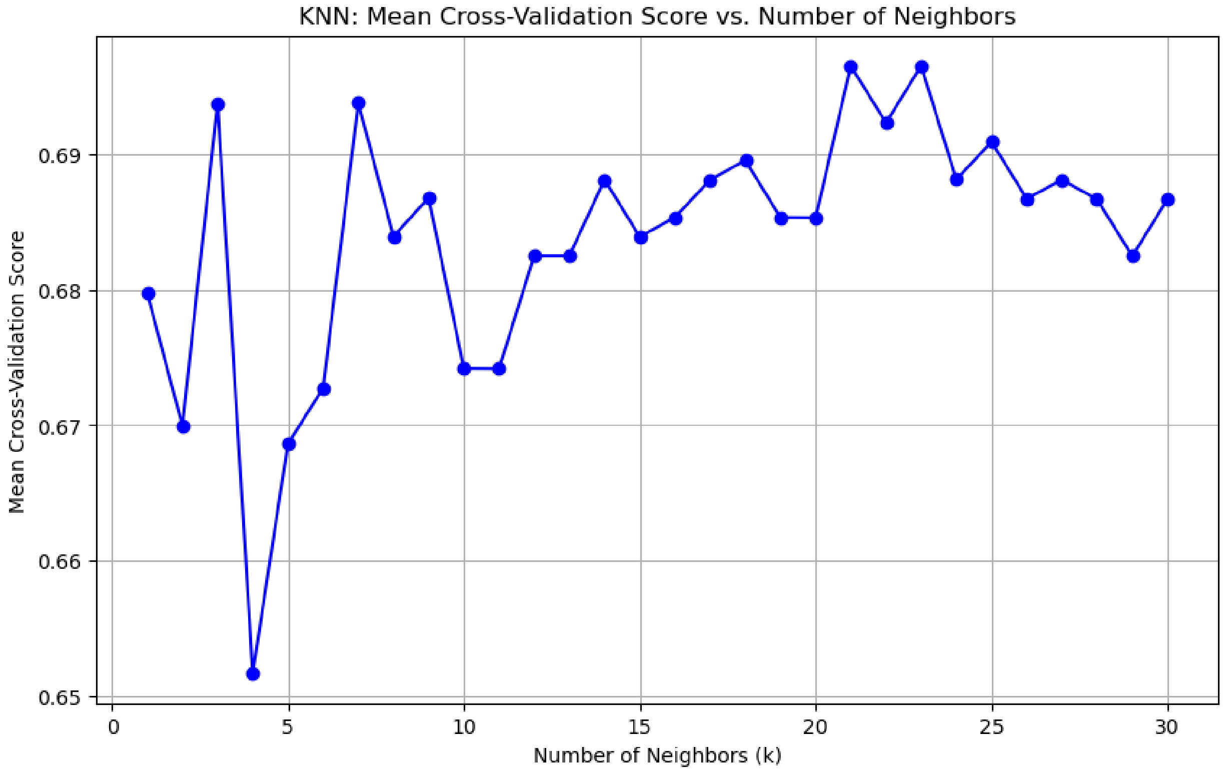
In [47]:

```python
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

# Your KNN code here...
# Create a KNN classifier
knn_model = KNeighborsClassifier()

# Fit the KNN model to the training data
knn_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_knn = knn_model.predict(X_test)

# Evaluate the KNN model
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print("KNN Accuracy:", accuracy_knn)

# Get a classification report for KNN
print("Classification Report (KNN):")
print(classification_report(y_test, y_pred_knn))

# Get the confusion matrix for KNN
conf_matrix_knn = confusion_matrix(y_test, y_pred_knn)
print("Confusion Matrix (KNN):")
print(conf_matrix_knn)
```

```
KNN Accuracy: 0.6927374301675978
Classification Report (KNN):
              precision    recall  f1-score   support

           0       0.71      0.80      0.75       105
           1       0.66      0.54      0.59        74

    accuracy                           0.69       179
   macro avg       0.68      0.67      0.67       179
weighted avg       0.69      0.69      0.69       179

Confusion Matrix (KNN):
[[84 21]
 [34 40]]
```

In [49]:

```
# Define a range of k values to try (e.g., from 1 to 30)
k_values = np.arange(1, 31)

# Initialize lists to store the mean cross-validation scores for each k
cv_scores = []

# Perform k-fold cross-validation for each k value
for k in k_values:
    knn_model = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn_model, X_train, y_train, cv=5)
    cv_scores.append(scores.mean())

# Find the optimal k with the highest mean cross-validation score
optimal_k = k_values[np.argmax(cv_scores)]

# Plot the mean cross-validation scores for each k value
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.plot(k_values, cv_scores, marker='o', linestyle='-', color='b')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Mean Cross-Validation Score')
plt.title('KNN: Mean Cross-Validation Score vs. Number of Neighbors')
plt.grid(True)
plt.show()

print("Optimal Number of Neighbors (k):", optimal_k)

```



```
Optimal Number of Neighbors (k): 23
```

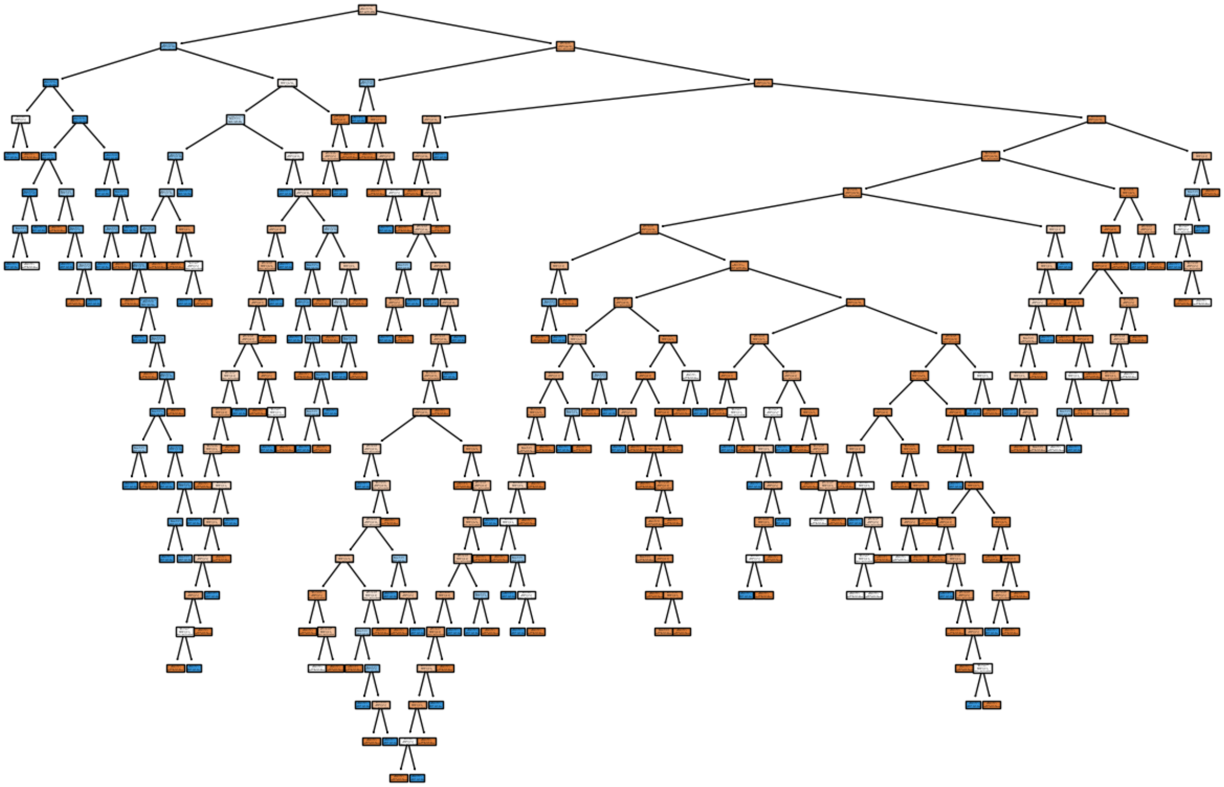
In [50]:

```python
from sklearn.tree import DecisionTreeClassifier

# Create a Decision Tree classifier
decision_tree_model = DecisionTreeClassifier(random_state=42)

# Fit the Decision Tree model to the training data
decision_tree_model.fit(X_train, y_train)

# Make predictions on the test set using the Decision Tree model
y_pred_decision_tree = decision_tree_model.predict(X_test)

# Evaluate the Decision Tree model
accuracy_decision_tree = accuracy_score(y_test, y_pred_decision_tree)
print("Decision Tree Accuracy:", accuracy_decision_tree)

# Get a classification report for Decision Tree
print("Classification Report (Decision Tree):")
print(classification_report(y_test, y_pred_decision_tree))

# Get the confusion matrix for Decision Tree
conf_matrix_decision_tree = confusion_matrix(y_test, y_pred_decision_tr
print("Confusion Matrix (Decision Tree):")
print(conf_matrix_decision_tree)

```

```
Decision Tree Accuracy: 0.7932960893854749
Classification Report (Decision Tree):
              precision    recall  f1-score   support

           0       0.81      0.84      0.83       105
           1       0.76      0.73      0.74        74

    accuracy                           0.79       179
   macro avg       0.79      0.78      0.79       179
weighted avg       0.79      0.79      0.79       179

Confusion Matrix (Decision Tree):
[[88 17]
 [20 54]]
```

In [55]:

```
plt.rcParams['figure.figsize'] = (15, 10)
plot_tree(decision_tree_model, feature_names=X.columns, class_names=['N
plt.show()
```



In [56]:

```
# Create a Decision Tree classifier with modified parameters
decision_tree_model_modified = DecisionTreeClassifier(max_depth=5, min_

# Fit the modified Decision Tree model to the training data
decision_tree_model_modified.fit(X_train, y_train)

# Make predictions on the test set using the modified Decision Tree mod
y_pred_decision_tree_modified = decision_tree_model_modified.predict(X_

# Evaluate the modified Decision Tree model
accuracy_decision_tree_modified = accuracy_score(y_test, y_pred_decisio
print("Modified Decision Tree Accuracy:", accuracy_decision_tree_modifi
```

```
Modified Decision Tree Accuracy: 0.7988826815642458
```

In [57]:

```
from sklearn.ensemble import RandomForestClassifier

# Create a Random Forest classifier with default parameters (n_estimato
random_forest_model = RandomForestClassifier(random_state=42)

# Fit the Random Forest model to the training data
random_forest_model.fit(X_train, y_train)

# Make predictions on the test set using the Random Forest model
y_pred_random_forest = random_forest_model.predict(X_test)

# Evaluate the Random Forest model
accuracy_random_forest = accuracy_score(y_test, y_pred_random_forest)
print("Random Forest Accuracy (Default n_estimators=100):", accuracy_ra
```

```
Random Forest Accuracy (Default n_estimators=100): 0.8044692737430168
```

In [58]:

```
# Create a Random Forest classifier with modified number of estimators
random_forest_model_modified = RandomForestClassifier(n_estimators=200,

# Fit the modified Random Forest model to the training data
random_forest_model_modified.fit(X_train, y_train)

# Make predictions on the test set using the modified Random Forest mod
y_pred_random_forest_modified = random_forest_model_modified.predict(X_

# Evaluate the modified Random Forest model
accuracy_random_forest_modified = accuracy_score(y_test, y_pred_random_
print("Random Forest Accuracy (Modified n_estimators=200):", accuracy_r

```

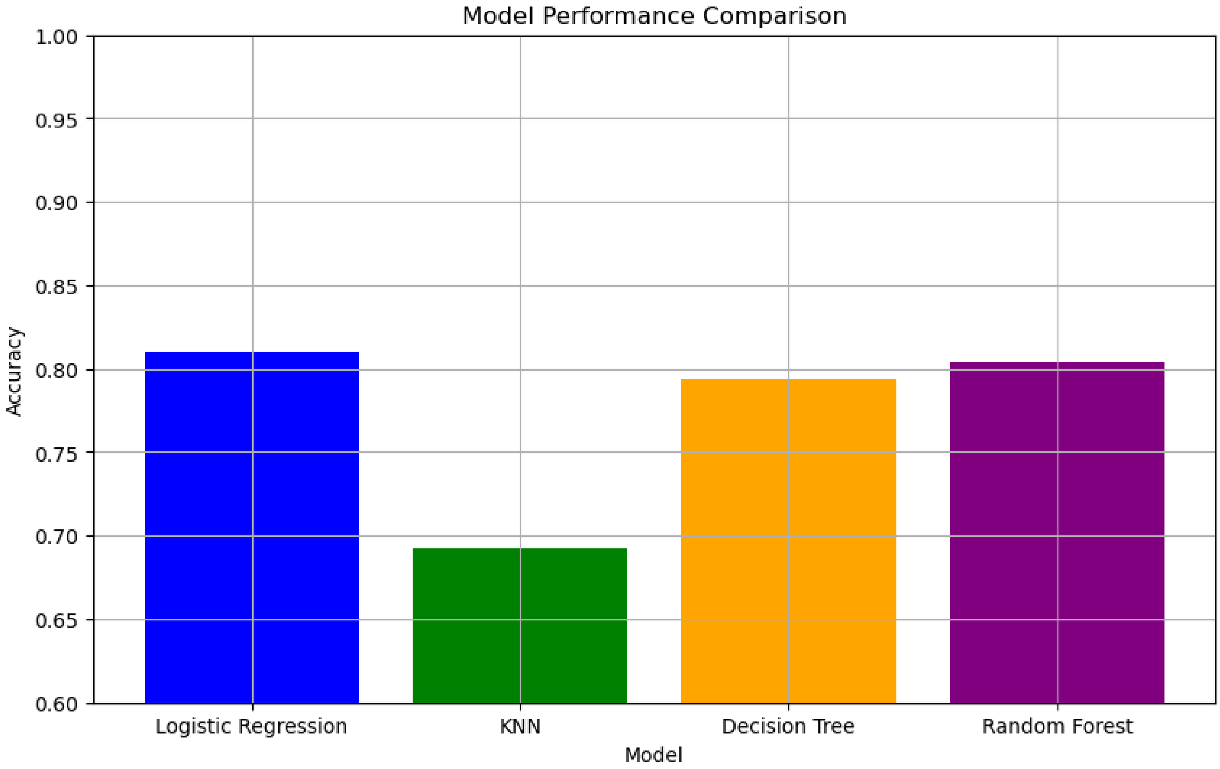
```
Random Forest Accuracy (Modified n_estimators=200): 0.8100558659217877
```

In [65]:

```
import matplotlib.pyplot as plt

# List of model names and their respective accuracy scores
model_names = ['Logistic Regression', 'KNN', 'Decision Tree', 'Random F
acc = [accuracy_knn, accuracy_decision_tree, accuracy_random_forest]
```

In [69]:

```
import matplotlib.pyplot as plt

# List of model names and their respective accuracy scores
model_names = ['Logistic Regression', 'KNN', 'Decision Tree', 'Random F
accuracy_scores = [accuracy, accuracy_knn, accuracy_decision_tree, accu

# Plot the summary graph
plt.figure(figsize=(10, 6))
plt.bar(model_names, accuracy_scores, color=['blue', 'green', 'orange',
plt.ylim(0.6, 1.0)  # Set the y-axis limits to better compare the accur
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Model Performance Comparison')
plt.grid(True)
plt.show()

```



In [ ]:

```

```