In [20]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
```

In [21]:
```
# Load dataset
df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/m
```

In [22]:
```
# Explore structure
print(df.info())
print(df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

In [23]:
```
 #Handle missing values
# Age (numerical), Cabin (drop), Embarked (categorical)
df.drop(columns=['Cabin', 'Name', 'Ticket', 'PassengerId'], inplace=True)
```

In [24]:
```
# Define features and target
X = df.drop('Survived', axis=1)
y = df['Survived']
```

In [25]:
```
# Identify column types
numeric_features = ['Age', 'Fare']
categorical_features = ['Sex', 'Embarked', 'Pclass', 'SibSp', 'Parch']

# Create transformers
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])
```

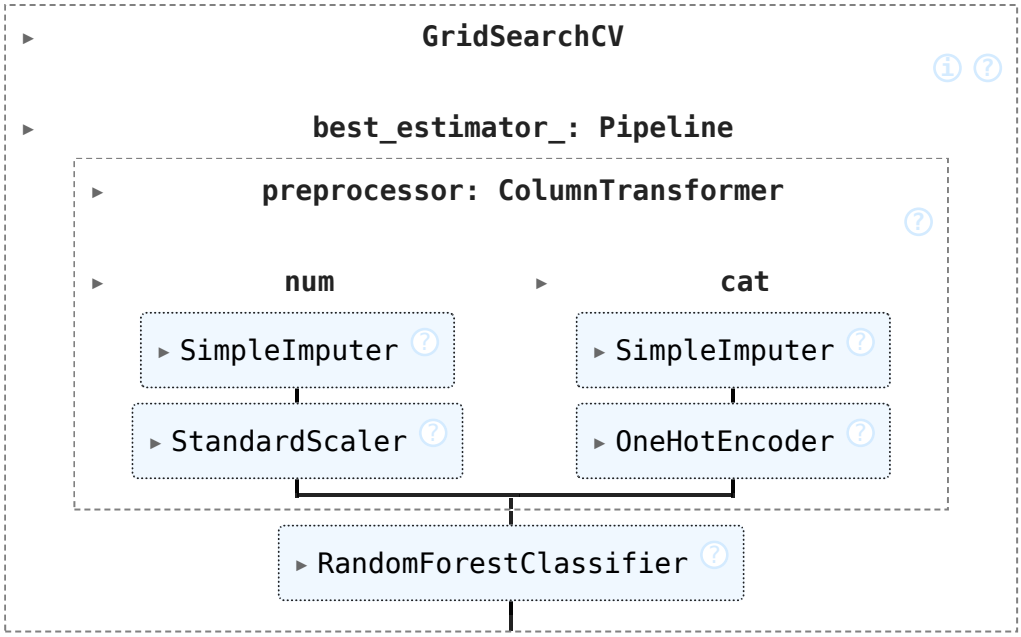
In [26]:
```
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])
```

In [27]:
```
# Combine transformers
preprocessor = ColumnTransformer(transformers=[
    ('num', numeric_transformer, numeric_features),
    ('cat', categorical_transformer, categorical_features)
])
```

In [28]:
```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc
```

In [29]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, cross_val_score

# Append model to preprocessing pipeline
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))
])

# Grid search for hyperparameters
param_grid = {
    'classifier__n_estimators': [50, 100],
    'classifier__max_depth': [None, 5, 10]
}

grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train,y_train)
```

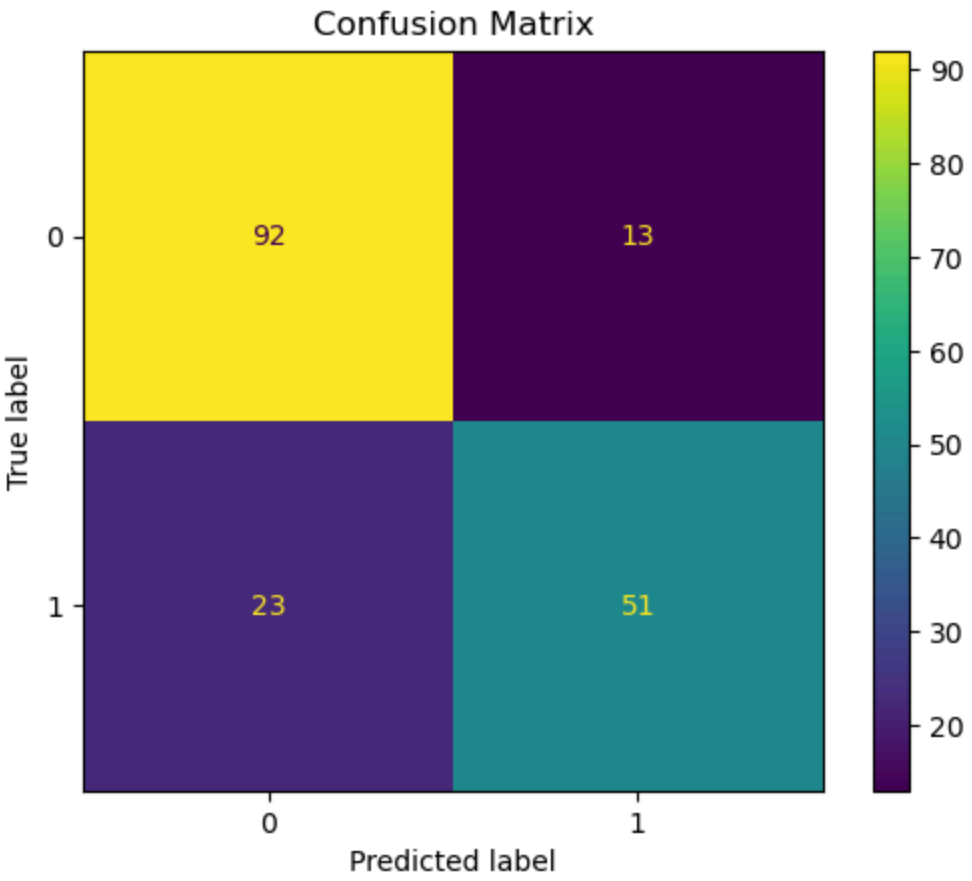
Out[29]:

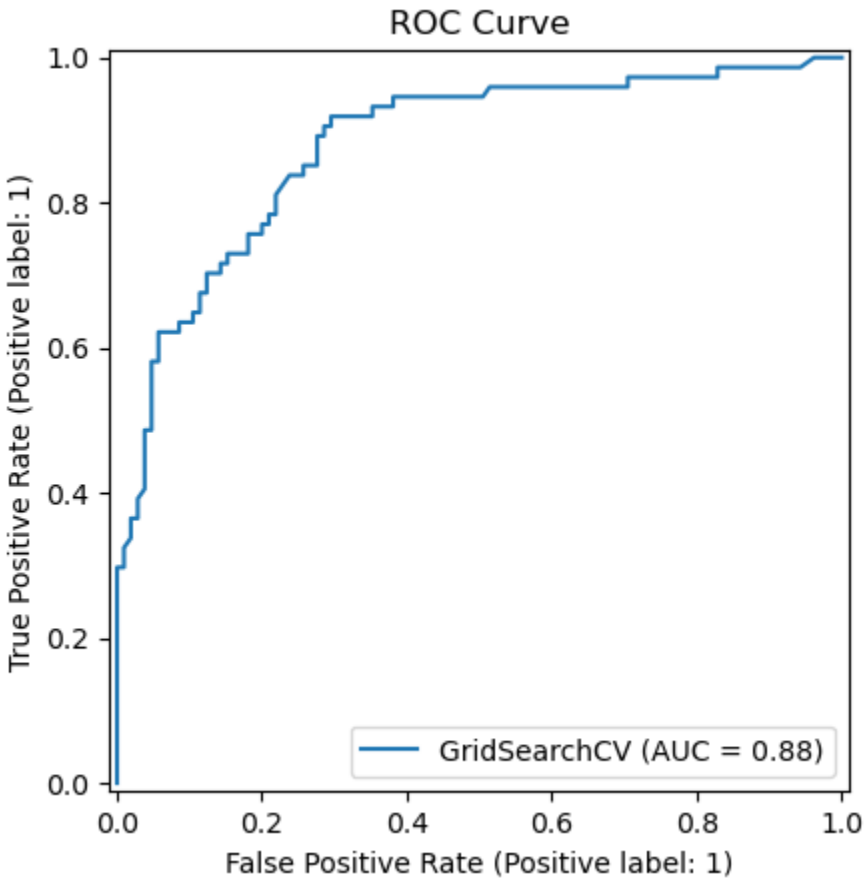


In [30]:

```
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_s
import matplotlib.pyplot as plt

# Predict
y_pred = grid_search.predict(X_test)
y_proba = grid_search.predict_proba(X_test)[:, 1]

# Evaluation
print(classification_report(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, y_proba))

# Confusion Matrix
ConfusionMatrixDisplay.from_estimator(grid_search, X_test, y_test)
plt.title("Confusion Matrix")
plt.show()

# ROC Curve
RocCurveDisplay.from_estimator(grid_search, X_test, y_test)
plt.title("ROC Curve")
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.80      0.88      0.84       105
           1       0.80      0.69      0.74        74

    accuracy                           0.80       179
   macro avg       0.80      0.78      0.79       179
weighted avg       0.80      0.80      0.80       179

ROC-AUC: 0.8794723294723293
```


Confusion Matrix
92
13
23
51
0
1
True label
0
1
Predicted label
90
80
70
60
50
40
30
20


ROC Curve
True Positive Rate (Positive label: 1)
False Positive Rate (Positive label: 1)
GridSearchCV (AUC = 0.88)
0.0
0.2
0.4
0.6
0.8
1.0

In [31]:
```
# This is the best trained pipeline from grid search
final_pipeline = grid_search.best_estimator_

# Predict on new data
new_predictions = final_pipeline.predict(X_test)

# Reuse this pipeline for production/deployment
import joblib
joblib.dump(final_pipeline, 'titanic_model_pipeline.pkl')# Save pipeline
```

Out[31]:
```
['titanic_model_pipeline.pkl']
```

In [33]:
```
# Check what's inside the pipeline
print(final_pipeline)
```

```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
                                                                   SimpleImpute
r(strategy='median')),
                                                                  ('scaler',
                                                                   StandardScal
er())]),
                                                  ['Age', 'Fare']),
                                                 ('cat',
                                                  Pipeline(steps=[('imputer',
                                                                   SimpleImpute
r(strategy='most_frequent')),
                                                                  ('encoder',
                                                                   OneHotEncode
r(handle_unknown='ignore'))]),
                                                  ['Sex', 'Embarked', 'Pclass',
                                                   'SibSp', 'Parch'])])),
                ('classifier',
                 RandomForestClassifier(max_depth=5, n_estimators=50,
                                        random_state=42))])
```

In [ ]: