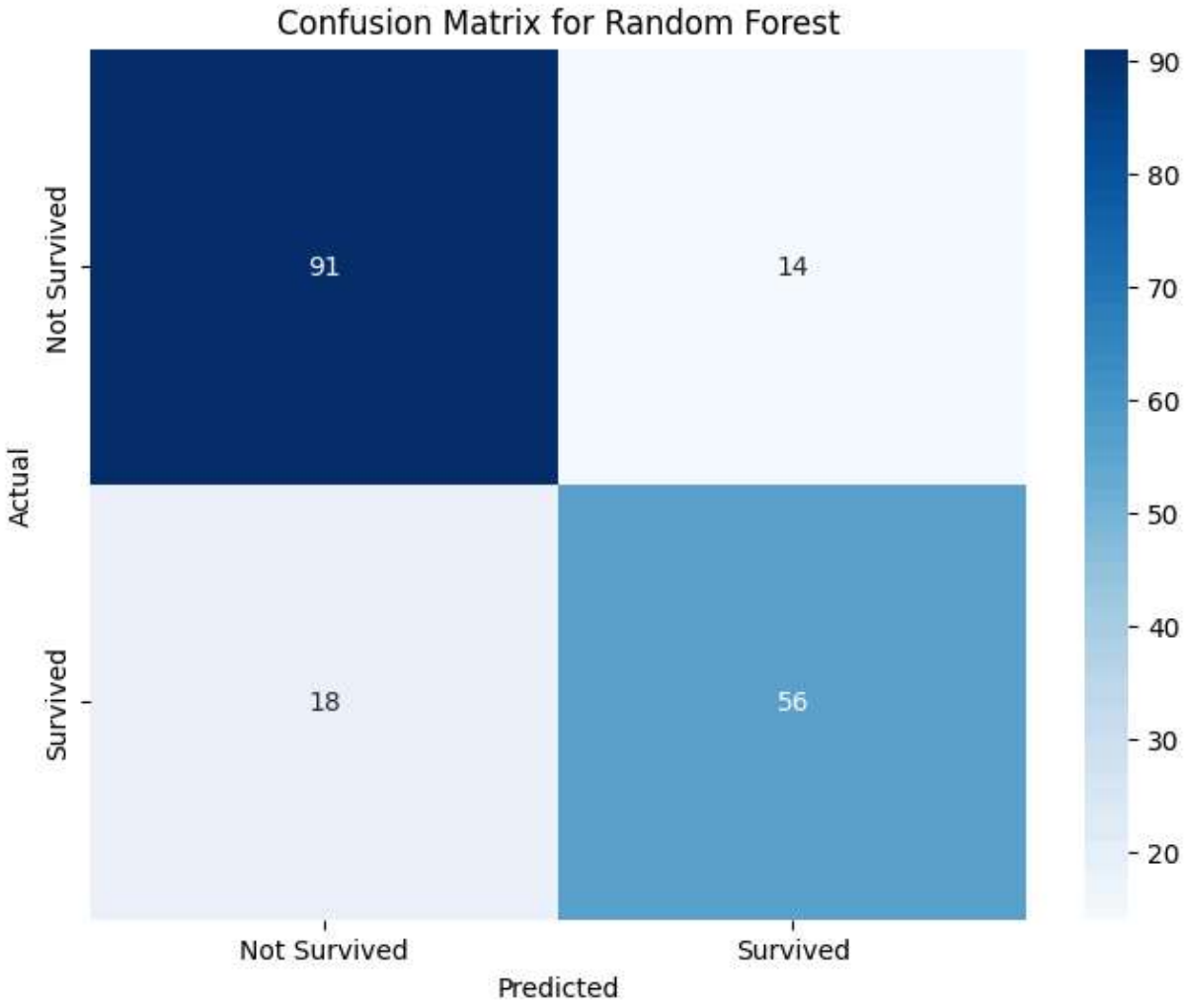
In [2]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns


url = 'https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.cs
df = pd.read_csv(url)


df['Age'] = df['Age'].fillna(df['Age'].median())
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()[0])


df['FamilySize'] = df['SibSp'] + df['Parch']
df['Title'] = df['Name'].apply(lambda x: x.split(',')[1].split('.')[0].strip())


df.drop(['Name', 'Ticket', 'Cabin'], axis=1, inplace=True)


features = ['Pclass', 'Sex', 'Age', 'FamilySize', 'Fare', 'Embarked']
X = df[features]
y = df['Survived']


preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['Age', 'FamilySize', 'Fare']),
        ('cat', OneHotEncoder(), ['Pclass', 'Sex', 'Embarked'])
    ])


models = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(),
    'SVM': SVC(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'Naive Bayes': GaussianNB(),
    'KNN': KNeighborsClassifier()
}
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


results = pd.DataFrame(columns=['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Sco

model_results = []

for name, model in models.items():
    pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                               ('model', model)])
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    model_results.append({
        'Model': name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1
    })

results = pd.DataFrame(model_results)

print(results)


best_model_name = 'Random Forest'
best_model = models[best_model_name]

pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('model', best_model)])
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)

conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Not Survived', 'Survived'],
            yticklabels=['Not Survived', 'Survived'])
plt.title(f'Confusion Matrix for {best_model_name}')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
                Model  Accuracy  Precision    Recall  F1 Score
0  Logistic Regression  0.798883   0.779412  0.716216  0.746479
1        Random Forest  0.832402   0.805556  0.783784  0.794521
2                  SVM  0.815642   0.815385  0.716216  0.762590
3    Gradient Boosting  0.815642   0.815385  0.716216  0.762590
4          Naive Bayes  0.776536   0.712500  0.770270  0.740260
5                  KNN  0.810056   0.785714  0.743243  0.763889
```



In [4]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns


url = 'https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.cs
df = pd.read_csv(url)
df['Age'] = df['Age'].fillna(df['Age'].median())
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()[0])
df['FamilySize'] = df['SibSp'] + df['Parch']
df['Title'] = df['Name'].apply(lambda x: x.split(',')[1].split('.')[0].strip())
df.drop(['Name', 'Ticket', 'Cabin'], axis=1, inplace=True)
features = ['Pclass', 'Sex', 'Age', 'FamilySize', 'Fare', 'Embarked']
```

```
X = df[features]
y = df['Survived']
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['Age', 'FamilySize', 'Fare']),
        ('cat', OneHotEncoder(), ['Pclass', 'Sex', 'Embarked'])
    ])
models = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(),
    'SVM': SVC(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'Naive Bayes': GaussianNB(),
    'KNN': KNeighborsClassifier()
}
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
best_model_name = 'Random Forest'
best_model = models[best_model_name]
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('model', best_model)])
pipeline.fit(X_train, y_train)
def predict_survival():
    print("Enter the details for the passenger:")


    pclass = int(input("Pclass (1, 2, or 3): "))
    sex = input("Sex (male or female): ").strip().lower()
    age = float(input("Age: "))
    family_size = int(input("Family Size (SibSp + Parch): "))
    fare = float(input("Fare: "))
    embarked = input("Embarked (C, Q, or S): ").strip().upper()


    input_data = {
        'Pclass': pclass,
        'Sex': sex,
        'Age': age,
        'FamilySize': family_size,
        'Fare': fare,
        'Embarked': embarked
    }

    input_df = pd.DataFrame([input_data], columns=features)


    prediction = pipeline.predict(input_df)


    if prediction[0] == 1:
        print('Survived')
    else:
        print('Did not survive')


predict_survival()
results = pd.DataFrame(columns=['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Sco
```

```
model_results = []

for name, model in models.items():
    pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                               ('model', model)])
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    model_results.append({
        'Model': name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1
    })

results = pd.DataFrame(model_results)

print(results)


conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Not Survived', 'Survived'],
            yticklabels=['Not Survived', 'Survived'])
plt.title(f'Confusion Matrix for {best_model_name}')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

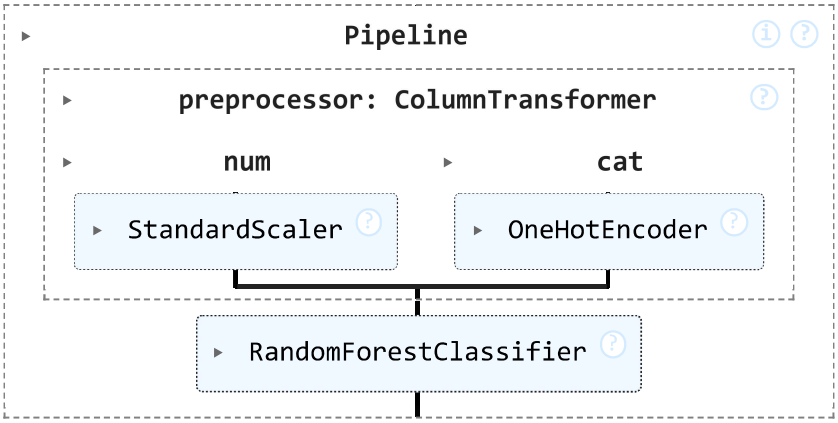
```
Enter the details for the passenger:
Did not survive
                Model  Accuracy  Precision    Recall  F1 Score
0  Logistic Regression  0.798883   0.779412  0.716216  0.746479
1        Random Forest  0.821229   0.783784  0.783784  0.783784
2                  SVM  0.815642   0.815385  0.716216  0.762590
3    Gradient Boosting  0.815642   0.815385  0.716216  0.762590
4          Naive Bayes  0.776536   0.712500  0.770270  0.740260
5                  KNN  0.810056   0.785714  0.743243  0.763889
```

In [16]:
```
from sklearn.model_selection import GridSearchCV
param_grid_rf = {
    'model__n_estimators': [100, 200, 300],
    'model__max_depth': [None, 10, 20, 30],
    'model__min_samples_split': [2, 5, 10]
}
grid_search_rf = GridSearchCV(Pipeline(steps=[('preprocessor', preprocessor),
                                              ('model', RandomForestClassifier())]),
                              param_grid_rf, cv=5, scoring='accuracy')
grid_search_rf.fit(X_train, y_train)
print(f"Best parameters: {grid_search_rf.best_params_}")
print(f"Best score: {grid_search_rf.best_score_}")
```

```
Best parameters: {'model__max_depth': 30, 'model__min_samples_split': 10, 'model__n_
estimators': 300}
Best score: 0.8272431793558553
```

In [21]:
```
best_rf_model = RandomForestClassifier(n_estimators=300, max_depth=30, min_samples_
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('model', best_rf_model)])
pipeline.fit(X_train, y_train)
```

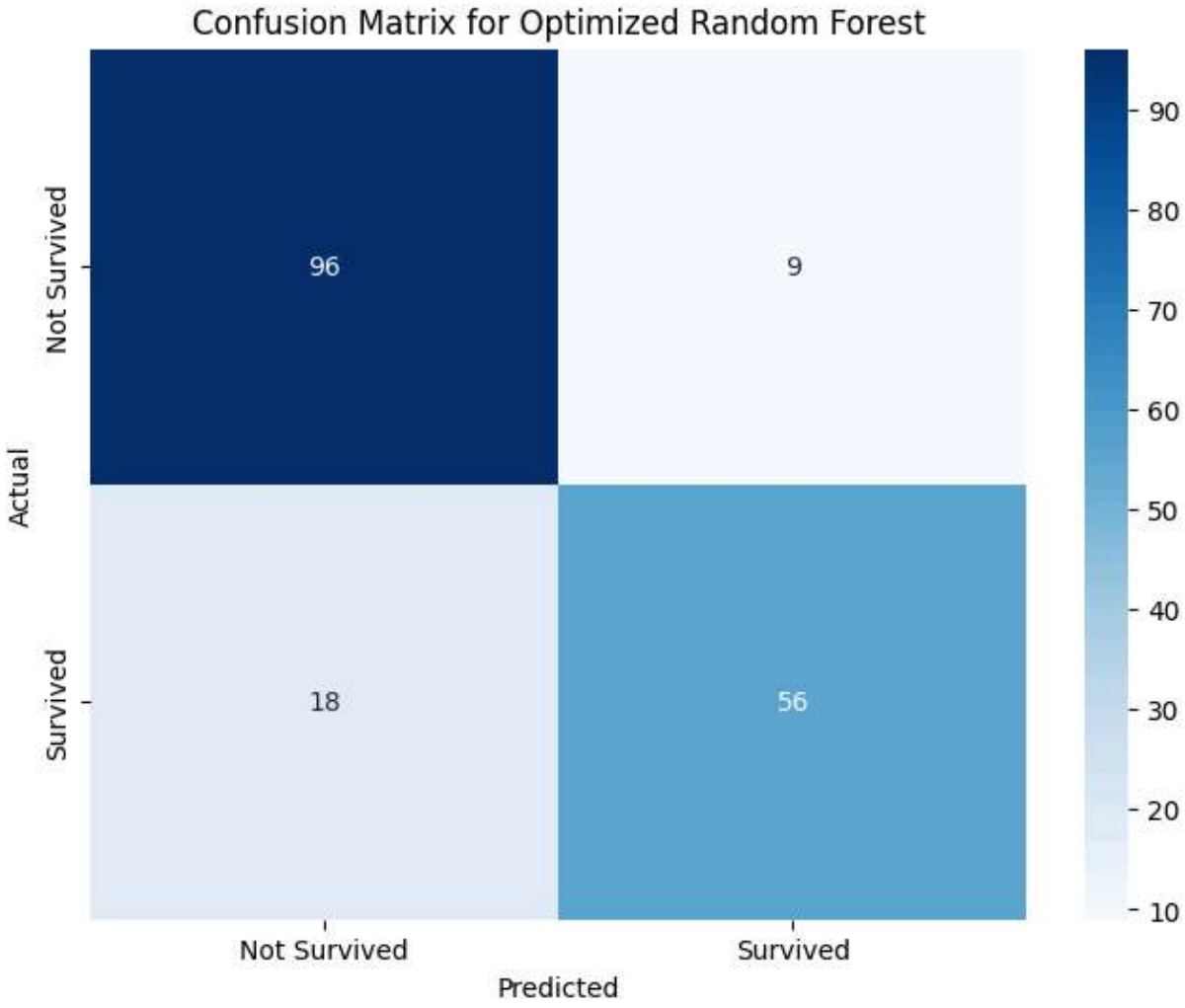
Out[21]:



In [22]:

```
y_pred = pipeline.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Updated Random Forest Model Performance:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")


conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Not Survived', 'Survived'],
            yticklabels=['Not Survived', 'Survived'])
plt.title('Confusion Matrix for Optimized Random Forest')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Updated Random Forest Model Performance:
Accuracy: 0.8492
Precision: 0.8615
Recall: 0.7568
F1 Score: 0.8058
```


Confusion Matrix for Optimized Random Forest
Not Survived
Survived
Actual
96
9
18
56
Not Survived
Survived
Predicted
90
80
70
60
50
40
30
20
10

In [ ]: