Titanic Survival Prediction Using Multiple Algorithms

Step 1: Import Necessary Libraries

In [2]:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report ,confusion_matrix
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import export_graphviz
```

Step 2: Load the Datasets

In [3]:

```
train_file_path = '../train.csv'
test_file_path = '../test.csv'
actual_results_file_path = '../gender_submission.csv'

train_df = pd.read_csv(train_file_path)
test_df = pd.read_csv(test_file_path)
actual_results_df = pd.read_csv(actual_results_file_path)
```

Step 3: Data Preprocessing

Fill missing values

In [4]:

```
for dataset in [train_df, test_df]:
    dataset['Age'].fillna(dataset['Age'].median(), inplace=True)
    dataset['Embarked'].fillna(dataset['Embarked'].mode()[0], inplace=True)
    dataset['Fare'].fillna(dataset['Fare'].median(), inplace=True)
```

Drop unwanted columns

In [5]:

```
columns_to_drop = ['Cabin', 'Name', 'Ticket']
train_df.drop(columns=columns_to_drop, axis=1, inplace=True)
test_df.drop(columns=columns_to_drop, axis=1, inplace=True)
```

Encode categorical variables

In [6]:

```
label_encoder = LabelEncoder()
for col in ['gender', 'Embarked']:
    train_df[col] = label_encoder.fit_transform(train_df[col])
    test_df[col] = label_encoder.transform(test_df[col])
```

Step 4: Define Features and Target

In [7]:

```
features = ['Pclass', 'gender', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
X_train = train_df[features]
y_train = train_df['Survived']
X_test = test_df[features]
y_test = actual_results_df['Survived']
```

Step 5: Train and Evaluate Models

Helper function for model evaluation

In [8]:

```
# def evaluate_model(model, X_train, y_train, X_test, y_test):
#     predictions = model.predict(X_test)
#     accuracy = accuracy_score(y_test, predictions)
#     print(f"Accuracy: {accuracy:.2f}")
#     print("Classification Report:\n", classification_report(y_test, predictions))
#     return model, predictions
```

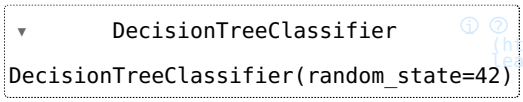
Model 1: Decision Tree Classifier

In [9]:

```
print("\n--- Decision Tree Classifier ---")
decision_tree_clf = DecisionTreeClassifier(random_state=42)
decision_tree_clf.fit(X_train, y_train)
```

```
--- Decision Tree Classifier ---
```

Out[9]:



In [10]:

```
predictions = decision_tree_clf.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", classification_report(y_test, predictions))
```

```
Accuracy: 0.75
Classification Report:
               precision    recall  f1-score   support

           0       0.85      0.74      0.79       266
           1       0.63      0.77      0.69       152

    accuracy                           0.75       418
   macro avg       0.74      0.75      0.74       418
weighted avg       0.77      0.75      0.75       418
```

Export the decision tree to a .dot file for visualization

In [11]:

```
export_graphviz(decision_tree_clf,
                out_file="decision_tree.dot",
                feature_names=features,
                class_names=['0', '1'],
                filled=True)
```

Model 2: Random Forest Classifier

In [12]:

```
print("\n--- Random Forest Classifier ---")
random_forest_clf = RandomForestClassifier(n_estimators=100, random_state=42)
random_forest_clf.fit(X_train, y_train)
```

```
--- Random Forest Classifier ---
```

Out[12]:

In [13]:

```
predictions = random_forest_clf.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", classification_report(y_test, predictions))
```

```
Accuracy: 0.81
Classification Report:
               precision    recall  f1-score   support

           0       0.86      0.85      0.85       266
           1       0.74      0.75      0.74       152

    accuracy                           0.81       418
   macro avg       0.80      0.80      0.80       418
weighted avg       0.81      0.81      0.81       418
```

In [14]:

```
export_graphviz(random_forest_clf.estimators_[0],
                out_file="randomforest_tree.dot",
                feature_names=X_train.columns,
                class_names=['0', '1'],
                filled=True)
```

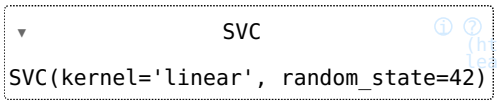
Model 3: Support Vector Machine (SVM)

In [15]:

```
print("\n--- Support Vector Machine (SVM) ---")
svm_clf = SVC(kernel='linear', random_state=42)
svm_clf.fit(X_train, y_train)
```

```
--- Support Vector Machine (SVM) ---
```

Out[15]:



In [16]:

```
predictions = svm_clf.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", classification_report(y_test, predictions))
```

```
Accuracy: 1.00
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       266
           1       1.00      1.00      1.00       152

    accuracy                           1.00       418
   macro avg       1.00      1.00      1.00       418
weighted avg       1.00      1.00      1.00       418
```

Step 6: Visualize Decision Boundary (SVM with Two Features)

Use only 'Age' and 'Fare' for visualization

In [17]:

```
X_train_visual = X_train[['Age', 'Fare']]
y_train_visual = y_train

# Train a new SVM model for visualization
svm_visual_clf = SVC(kernel='linear', random_state=42)
svm_visual_clf.fit(X_train_visual.values, y_train_visual)
```

Out[17]:

```
SVC(kernel='linear', random_state=42)
```

Function to plot decision boundary

In [18]:

```
plt.figure(figsize=(10, 6))
xx, yy = np.meshgrid(
    np.linspace(X_train_visual['Age'].min(), X_train_visual['Age'].max(), 100),
    np.linspace(X_train_visual['Fare'].min(), X_train_visual['Fare'].max(), 100)
)
Z = svm_visual_clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, levels=[-1, 0, 1], alpha=0.5, colors=['red', 'blue', 'green'])
plt.scatter(X_train_visual['Age'], X_train_visual['Fare'], c=y_train_visual, cmap='coolwarm', edgecolor='k')
plt.xlabel('Age')
plt.ylabel('Fare')
plt.title('SVM Decision Boundary')
plt.show()
```