1. Import Necessary Libraries

In [1]:

```
# Import necessary libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
# Import necessary libraries for evaluation
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.tree import export_graphviz
import pandas as pd
```

- `pandas (pd)` : Used for data manipulation and analysis, particularly for loading and working with data in DataFrame format.
- `DecisionTreeClassifier` : A machine learning algorithm from `sklearn` used to classify data by learning simple decision rules inferred from the features.
- `LabelEncoder` : A utility from `sklearn` to convert categorical string labels into numerical values.
- `accuracy_score` , `classification_report` , `confusion_matrix` : These are metrics to evaluate the performance of the classifier.
- `export_graphviz` : A function to export the decision tree structure for visualization.

2. Load the Training Dataset

In [2]:

```
# Load the training dataset
train_file_path = '../train.csv'
train_df = pd.read_csv(train_file_path)
```

- The training data is loaded from a CSV file using `pandas.read_csv` . The data is expected to be a CSV file, and the DataFrame `train_df` stores it.

3. Load the Testing Dataset

In [3]:

```
# Load the testing dataset
test_file_path = '../test.csv'  # Update this path with your actual test file path if needed
test_df = pd.read_csv(test_file_path)
```

- Similarly, the testing dataset is loaded from a CSV file into `test_df` . This dataset will be used to test the trained model.

4. Load the Actual Results for Evaluation

In [4]:

```
# Load the actual results from 'gender_submission.csv'
actual_results_file_path = '../gender_submission.csv'  # Update this path if necessary
actual_results_df = pd.read_csv(actual_results_file_path)
```

- The actual results (`gender_submission.csv`) are loaded to compare against the model's predictions.

5. Data Preprocessing for Training Set

In [5]:

```
# Data preprocessing for training set
# Fill missing age values with the median
train_df['Age'].fillna(train_df['Age'].median(), inplace=True)
```

In [6]:

```
# Fill missing embarked values with the most common port
train_df['Embarked'].fillna(train_df['Embarked'].mode()[0], inplace=True)
```

In [7]:

```
# Drop 'Cabin' due to too many missing values
train_df.drop('Cabin', axis=1, inplace=True)
```

- `train_df['Age'].fillna(train_df['Age'].median(), inplace=True)` : Missing values in the 'Age' column are filled with the median age from the training data. This is done to handle missing values.
- `train_df['Embarked'].fillna(train_df['Embarked'].mode()[0], inplace=True)` : Missing values in the 'Embarked' column are replaced with the most common port (mode).
- `train_df.drop('Cabin', axis=1, inplace=True)` : The 'Cabin' column is dropped because it contains too many missing values, making it unsuitable for analysis.

6. Encode Categorical Variables

In [8]:

```
# Encode categorical variables ('gender' and 'Embarked')
label_encoder = LabelEncoder()
train_df['gender'] = label_encoder.fit_transform(train_df['gender'])
train_df['Embarked'] = label_encoder.fit_transform(train_df['Embarked'])
```

- `LabelEncoder` : Converts categorical string values into numerical values. This is necessary because machine learning algorithms typically work with numerical data.
- `train_df['gender']` and `train_df['Embarked']` are encoded to numeric values (0 or 1).

7. Define Features and Target Variable for Training

In [9]:

```
# Features and target for training
features = ['PassengerId', 'Pclass', 'gender', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
X_train = train_df[features]
y_train = train_df['Survived']
```

- `features` : A list of the columns to be used as features (inputs) for the model. These columns represent characteristics of passengers that could influence their survival.
- `X_train` : Contains the feature values from the training data.
- `y_train` : The target variable, 'Survived', indicates whether the passenger survived (1) or not (0).

In [10]:

```
print('train data head')
train_df.head(5)
```

```
train data head
```

Out[10]:

	PassengerId	Survived	Pclass	Name	gender	Age	SibSp	Parch	Ticket	Fare	Embarked
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	2
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	38.0	1	0	PC 17599	71.2833	0
2	3	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282	7.9250	2
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	35.0	1	0	113803	53.1000	2
4	5	0	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	2

In [11]:

```
print('test data head')
test_df.head(5)
```

```
test data head
```

Out[11]:

	PassengerId	Pclass	Name	gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

In [12]:

```
print('result data head')
actual_results_df.head(5)
```

```
result data head
```

Out[12]:

	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	1

8. Initialize and Train the Decision Tree Classifier

In [13]:

```
# Initialize and train the Decision Tree Classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
```

Out[13]:

```
▾      DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

(https://scikit-learn.org/1.4/modules/generated/sklearn.tree.DecisionTreeClassifier.html)

- `DecisionTreeClassifier` : The decision tree model is initialized. `random_state=42` ensures reproducibility of the model's results.
- `clf.fit(X_train, y_train)` : The classifier is trained on the feature set `X_train` and the target variable `y_train` .

9. Data Preprocessing for the Test Set

In [14]:

```
# Data preprocessing for the test set
# Fill missing age values with the median from the training set
test_df['Age'].fillna(train_df['Age'].median(), inplace=True)
```

In [15]:

```
# Fill missing embarked values with the most common port from the training set
test_df['Embarked'].fillna(train_df['Embarked'].mode()[0], inplace=True)
```

In [16]:

```
# Drop 'Cabin' as done with the training set
test_df.drop('Cabin', axis=1, inplace=True)
```

In [17]:

```
# Encode categorical variables in the test set
test_df['gender'] = label_encoder.fit_transform(test_df['gender'])
test_df['Embarked'] = label_encoder.fit_transform(test_df['Embarked'])
```

- The test set is preprocessed in a similar manner to the training set:
  - Missing values in 'Age' are filled with the median from the training set.
  - Missing values in 'Embarked' are filled with the mode from the training set.
  - The 'Cabin' column is dropped.
  - Categorical columns ('gender' and 'Embarked') are encoded using the same `LabelEncoder` used for the training set to maintain consistency.

10. Select Features for the Test Set

In [18]:

```
# Select features for the test set
X_test = test_df[features]
```

- The features for the test set (`X_test`) are selected in the same way as for the training set.

11. Make Predictions on the Test Set

In [19]:

```
# Predict on the test set
y_pred = clf.predict(X_test)
```

- `y_pred` : The model makes predictions on the test data (`X_test`). These predictions are the model's estimations of whether each passenger survived or not.

12. Create a DataFrame for Output

In [20]:

```
# Create a DataFrame for the output
output = pd.DataFrame({
    'PassengerId': test_df['PassengerId'],
    'Survived': y_pred
})
```

- An output DataFrame is created, combining the `PassengerId` from the test set and the predicted survival labels (`y_pred`).

13. Display and Save the Prediction Results

In [21]:

```
# Display the prediction results
print(output)
```

```
     PassengerId  Survived
0            892         0
1            893         0
2            894         0
3            895         0
4            896         0
..           ...       ...
413         1305         0
414         1306         1
415         1307         0
416         1308         0
417         1309         1

[418 rows x 2 columns]
```

In [22]:

```
# Save the output to a CSV file
output.to_csv('titanic_predictions.csv', index=False)
```

- The `output` DataFrame is printed to the console and saved as a CSV file (`titanic_predictions.csv`) for later use.

14. Align the Actual Results with the Predictions

In [23]:

```
# Ensure that the 'PassengerId' is used to align predictions and actual results
y_test = actual_results_df.set_index('PassengerId')['Survived'].reindex(X_test['PassengerId']).values
```

- `y_test` : The actual survival outcomes are retrieved using the `PassengerId` as the index and reindexed to align with the test set's predictions.

In [24]:

```
# Evaluate the model
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

```
Accuracy Score: 0.7990430622009569
```

In [25]:

```
# Generate the classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

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

           0       0.82      0.88      0.85       266
           1       0.76      0.65      0.70       152

    accuracy                           0.80       418
   macro avg       0.79      0.77      0.78       418
weighted avg       0.80      0.80      0.80       418
```

- `accuracy_score` : Computes the accuracy, i.e., the proportion of correct predictions out of all predictions.
- `classification_report` : Provides a detailed report with precision, recall, and F1-score for each class (0 or 1), along with the overall accuracy.

16. Generate the Decision Tree Visualization

In [26]:

```
# Export the decision tree to a .dot file for visualization
export_graphviz(clf,
                out_file="tree.dot",
                feature_names=features,
                class_names=['0', '1'],
                filled=True)

# Note: You can use tools like Graphviz or online viewers to visualize the 'tree.dot' file.
```

- `export_graphviz` : Exports the decision tree as a `.dot` file, which can be used with Graphviz to visualize the tree. The `filled=True` option colors the nodes based on the predicted class.

17. Print the Confusion Matrix

In [27]:

```
# Print the confusion matrix
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
 [[235  31]
 [ 53  99]]
```

In [28]:

```
# Export the decision tree to a .dot file for visualization
export_graphviz(clf,
                out_file="tree.dot",
                feature_names=features,
                class_names=['0', '1'],
                filled=True)

# Note: You can use tools like Graphviz or online viewers to visualize the 'tree.dot' file.
```

- `confusion_matrix` : Computes a confusion matrix, which compares the actual vs predicted values. It shows how many true positives, true negatives, false positives, and false negatives the model produced.

Conclusion

This code demonstrates the process of training a decision tree model on a Titanic dataset to predict survival outcomes based on various passenger features. After training, the model's performance is evaluated using accuracy, classification reports, and confusion matrices, and the decision tree is exported for visualization.