1. Import Necessary Libraries

In [1]:

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

- **`pandas`** is used for data manipulation and analysis.
- **`RandomForestClassifier`** from `sklearn.ensemble` is an ensemble learning method that uses multiple decision trees to make more accurate and robust predictions.
- **`LabelEncoder`** is used to convert categorical data into numeric format.
- **`accuracy_score`, `classification_report`, and `confusion_matrix`** are metrics used for model evaluation.
- **`export_graphviz`** helps visualize the decision trees within the random forest.

2. Load the Training Dataset

In [2]:

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

- The `train.csv` file is read into a DataFrame named `train_data`. This dataset contains features and a target variable (`Survived`) used to train the model.

3. Load the Testing Dataset

In [3]:

```
# Load the testing dataset (without target column) and actual results
test_file_path = '../test.csv'  # Replace with the correct path if needed
test_data = pd.read_csv(test_file_path)
```

- The `test.csv` file is read into a DataFrame named `test_data`. This dataset includes features but not the `Survived` column, which the model predicts.

4. Load the Actual Results

In [4]:

```
# Load the actual results for the test data
gender_submission_file_path = '../gender_submission.csv'  # Replace with the correct path if needed
actual_results = pd.read_csv(gender_submission_file_path)
```

- `gender_submission.csv` contains the actual `Survived` values for the test set, used for model evaluation.

5. Preprocess the Training Data

In [5]:

```
# Preprocess the training data
train_data['Age'].fillna(train_data['Age'].median(), inplace=True)
train_data['Cabin'].fillna('Unknown', inplace=True)
train_data['Embarked'].fillna(train_data['Embarked'].mode()[0], inplace=True)
```

- **Missing value handling**:
  - `Age` : Missing values are replaced with the median age.
  - `Cabin` : Missing cabin values are filled with 'Unknown'.
  - `Embarked` : Missing embarked values are filled with the most common port (mode).

6. Label Encoding for Categorical Variables

In [6]:

```
# Initialize and fit LabelEncoders for each column needing encoding
label_encoders = {}
for column in ['gender', 'Cabin', 'Embarked', 'Name', 'Ticket']:
    le = LabelEncoder()
    train_data[column] = le.fit_transform(train_data[column])
    label_encoders[column] = le
```

- Categorical columns (`gender` , `Cabin` , `Embarked` , `Name` , `Ticket`) are converted to numeric using `LabelEncoder` .
- A dictionary (`label_encoders`) stores the encoders for later use on the test data.

7. Select Features and Target Variable for Training

In [7]:

```
# Select features and target variable for training
X_train = train_data[['PassengerId', 'Pclass', 'Name', 'gender', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabi
n', 'Embarked']]
y_train = train_data['Survived']
```

- **`X_train`** : Features used to train the model.
- **`y_train`** : The target variable indicating survival (1 for survived, 0 for not).

In [8]:

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

```
train data head
```

Out[8]:

	PassengerId	Survived	Pclass	Name	gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	108	1	22.0	1	0	523	7.2500	147	2
1	2	1	1	190	0	38.0	1	0	596	71.2833	81	0
2	3	1	3	353	0	26.0	0	0	669	7.9250	147	2
3	4	1	1	272	0	35.0	1	0	49	53.1000	55	2
4	5	0	3	15	1	35.0	0	0	472	8.0500	147	2

In [9]:

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

```
test data head
```

Out[9]:

	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 [10]:

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

```
result data head
```

Out[10]:

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

8. Train the Random Forest Classifier

In [11]:

```
# Train the Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)
```

Out[11]:

```
▾      RandomForestClassifier
RandomForestClassifier(random_state=42)
```

(https://scikit-learn.org/1.4/modules/generated/sklearn.ensemble.RandomForestClassifier.

- A **Random Forest Classifier** with 100 decision trees (`n_estimators=100`) is created and trained using `fit()` .
- `random_state=42` ensures reproducibility by initializing the random number generator.

9. Preprocess the Test Data

In [12]:

```
# Preprocess the test data (similar preprocessing steps as training data)
test_data['Age'].fillna(test_data['Age'].median(), inplace=True)
test_data['Cabin'].fillna('Unknown', inplace=True)
test_data['Embarked'].fillna(test_data['Embarked'].mode()[0], inplace=True)
```

- The same preprocessing steps applied to `train_data` are applied to `test_data` to handle missing values.

10. Transform Test Data Using LabelEncoders

In [13]:

```
# Transform the test data using the fitted LabelEncoders
for column in ['gender', 'Cabin', 'Embarked', 'Name', 'Ticket']:
    if column in label_encoders:
        le = label_encoders[column]
        # Use .fit_transform on training and .transform on test, handling unseen labels safely
        test_data[column] = test_data[column].apply(lambda x: le.transform([x])[0] if x in le.classes_ else -1)
```

- The test data columns are transformed using the `LabelEncoders` created earlier. If a value is not seen during training, it is encoded as `-1` .

11. Select Features for Test Data

In [14]:

```
X_test = test_data[['PassengerId', 'Pclass', 'Name', 'gender', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin'
, 'Embarked']]
```

- **`X_test`** contains the features to be used for predictions.

12. Make Predictions on Test Data

In [15]:

```
# Make predictions on the test data
y_pred = rf_classifier.predict(X_test)
```

- The trained model makes predictions on `X_test`.

13. Evaluate the Model

In [16]:

```
# Evaluate the model using the actual results
y_test = actual_results['Survived']
```

In [17]:

```
# Print evaluation metrics
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy Score: 0.8325358851674641

Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.78      0.86       266
           1       0.71      0.92      0.80       152

    accuracy                           0.83       418
   macro avg       0.83      0.85      0.83       418
weighted avg       0.86      0.83      0.84       418


Confusion Matrix:
 [[208  58]
 [ 12 140]]
```

- **`y_test`** contains the actual `Survived` values from `gender_submission.csv`.
- The model's accuracy, precision, recall, F1-score, and confusion matrix are printed.

14. Export Decision Trees for Visualization

In [18]:

```
# Export the decision tree from the random forest (for one tree)
export_graphviz(rf_classifier.estimators_[0],
                out_file="tree.dot",
                feature_names=X_train.columns,
                class_names=['0', '1'],
                filled=True)
```

In [19]:

```
# Export the decision tree from the random forest (for one tree)
export_graphviz(rf_classifier.estimators_[1],
                out_file="tree_1.dot",
                feature_names=X_train.columns,
                class_names=['0', '1'],
                filled=True)
```

- Two trees from the Random Forest (`0` and `99`) are exported as `.dot` files for visualization.
- **`feature_names`** specify the feature labels for the nodes.
- **`filled=True`** colors the nodes based on the class they represent.

Explanation of Random Forest Algorithm

- A **Random Forest** is an ensemble method that builds multiple decision trees using different subsets of the training data and features. The final output is the mode (classification) or average (regression) of all tree predictions.
- The main benefits include **reduced risk of overfitting** and **higher accuracy** due to aggregation.
- Randomness during training (sampling and feature selection) ensures **diverse trees**, which improves generalization.

Why Random Forest?

- **Resistant to overfitting** compared to individual decision trees.
- **Works well with large datasets** and can handle both numerical and categorical data.
- **Feature importance** is inherently available for model insights.