```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
d=pd.read_csv('/content/train.csv')
A=pd.read_csv('/content/gender_submission.csv')
```

```
# prompt: merage these fole and store in df

df = pd.merge(d, A, on='PassengerId', how='left')
df.head(3)
```

	PassengerId	Survived_x	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Survived_y
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	NaN

```
df.shape
```

```
(891, 13)
```

```
df.isnull().sum()
```

	0
PassengerId	0
Survived_x	0
Pclass	0
Name	0
Sex	0
Age	0
SibSp	0
Parch	0
Ticket	0
Fare	0
Embarked	2

dtype: int64

```
df.columns
```

```
Index(['PassengerId', 'Survived_x', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked', 'Survived_y'],
      dtype='object')
```

```
df.head(3)
```

	PassengerId	Survived_x	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	Sex_male	Embarked_Q	Embarked_S
0	1	0	3	Braund, Mr. Owen Harris	22.0	1	0	A/5 21171	7.2500	True	False	True
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	38.0	1	0	PC 17599	71.2833	False	False	False

```
# prompt: Convert categorical variables into numerical values. 0 and 1
# convert the sex_male column to only sex and gave 1 for male and 0 for female

df['Sex'] = df['Sex_male'].astype(int)
df.drop('Sex_male', axis=1, inplace=True)
```

```
df.head(3)
```

	PassengerId	Survived_x	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	Embarked_Q	Embarked_S	Sex
0	1	0	3	108	22.0	1	0	523	7.2500	False	True	1
1	2	1	1	190	38.0	1	0	596	71.2833	False	False	0
2	3	1	3	353	26.0	0	0	669	7.9250	False	True	0

```
# prompt: also do for embaraked colum convert into numerical
# there is two column embaraked_q and embaraked_y so make it one column and then do it

# Assuming the provided code is already executed and df is available.

# Create the 'Embarked' column based on 'Embarked_Q' and 'Embarked_Y'
df['Embarked'] = 0  # Initialize the 'Embarked' column
df.loc[df['Embarked_Q'] == 1, 'Embarked'] = 1  # If 'Embarked_Q' is 1, set 'Embarked' to 1
df.loc[df['Embarked_S'] == 1, 'Embarked'] = 2  # If 'Embarked_Y' is 1, set 'Embarked' to 2


# Drop the original 'Embarked_Q' and 'Embarked_Y' columns.
df = df.drop(['Embarked_Q', 'Embarked_S'], axis=1)

# Now you can check the updated dataframe
df.head(3)
```

	PassengerId	Survived_x	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	Sex	Embarked
0	1	0	3	108	22.0	1	0	523	7.2500	1	2
1	2	1	1	190	38.0	1	0	596	71.2833	0	0
2	3	1	3	353	26.0	0	0	669	7.9250	0	2

```
df.isnull().sum()
```

	0
PassengerId	0
Survived_x	0
Pclass	0
Name	0
Sex	0
Age	0
SibSp	0
Parch	0
Ticket	0
Fare	0
Embarked	0

dtype: int64

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived_x   891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          891 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  Embarked     891 non-null    object
dtypes: float64(2), int64(5), object(4)
memory usage: 76.7+ KB
```

```
df.describe()
```

	PassengerId	Survived_x	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.509894	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	13.529108	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	21.658443	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.994146	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	36.090382	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

```
# prompt: Normalize or scale numerical features like Age, Fare to avoid large differences in values affecting model performance.
# python
# Copy
# Edit

from sklearn.preprocessing import MinMaxScaler

# Select numerical features to scale
numerical_features = ['Age', 'Fare']

# Create a MinMaxScaler object
scaler = MinMaxScaler()

# Fit and transform the selected features
df[numerical_features] = scaler.fit_transform(df[numerical_features])

# Now you can check the updated dataframe
df.head(3)
df.describe()
```

	PassengerId	Survived_x	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	Sex	Embarked
count	749.000000	749.000000	749.000000	749.000000	749.000000	749.000000	749.000000	749.000000	749.000000	749.000000	749.000000
mean	446.857143	0.343124	2.506008	444.730307	0.484055	0.448598	0.344459	340.224299	0.276499	0.679573	1.603471
std	260.437178	0.475070	0.717638	254.575839	0.207570	0.911360	0.791946	189.502326	0.210066	0.466953	0.734220
min	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	2.000000	0.000000	0.000000	0.000000
25%	214.000000	0.000000	2.000000	224.000000	0.363733	0.000000	0.000000	180.000000	0.124621	0.000000	2.000000
50%	448.000000	0.000000	3.000000	449.000000	0.497158	0.000000	0.000000	335.000000	0.197685	1.000000	2.000000
75%	672.000000	1.000000	3.000000	659.000000	0.593496	1.000000	0.000000	499.000000	0.410365	1.000000	2.000000
max	891.000000	1.000000	3.000000	890.000000	1.000000	5.000000	6.000000	679.000000	1.000000	1.000000	2.000000

```
# prompt: check outlier

# Identify outliers using IQR method for numerical features
def find_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = data[(data[column] < lower_bound) | (data[column] > upper_bound)]
    return outliers

# Example usage for 'Age' and 'Fare'
age_outliers = find_outliers_iqr(df, 'Age')
fare_outliers = find_outliers_iqr(df, 'Fare')
```

```
print("Age Outliers:\n", age_outliers)
print("\nFare Outliers:\n", fare_outliers)

# Visualization of outliers (optional)
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
sns.boxplot(y='Age', data=df)
plt.title('Boxplot of Age')

plt.subplot(1, 2, 2)
sns.boxplot(y='Fare', data=df)
plt.title('Boxplot of Fare')

plt.show()
```

```
Age Outliers:
      PassengerId  Survived_x  Pclass  Name       Age  SibSp  Parch  Ticket  \
6              7           0       1   515  0.946978      0      0      85
15            16           1       2   359  0.964652      0      0     153
78            79           1       2   127  0.007246      0      2     158
152          153           0       3   532  0.973489      0      0     516
164          165           0       3   628  0.010251      4      1     249
172          173           1       3   408  0.010251      1      1     344
174          175           0       1   769  0.982326      0      0      90
183          184           1       2    76  0.010251      2      1     114
249          250           0       2   147  0.946978      1      0     144
317          318           0       2   555  0.946978      0      0     232
381          382           1       3   572  0.010251      0      2     187
386          387           0       3   298  0.010251      5      2     566
467          468           0       1   766  0.982326      0      0      44
469          470           1       3    54  0.005832      2      1     194
492          493           0       1   547  0.964652      0      0      41
513          514           1       1   705  0.946978      1      0     599
582          583           0       2   224  0.946978      0      0     226
626          627           0       2   442  1.000000      0      0     104
644          645           1       3    53  0.005832      2      1     194
647          648           1       1   747  0.982326      0      0      69
755          756           1       2   321  0.004419      1      1     166
772          773           0       2   496  1.000000      0      0     619
774          775           1       2   367  0.946978      1      3     236
788          789           1       3   208  0.010251      1      2     548
803          804           1       3   807  0.000000      0      1     174
827          828           1       2   503  0.010251      0      2     618
831          832           1       2   686  0.007246      1      1     237

         Fare  Sex  Embarked
6    0.818559    1         2
15   0.252532    0         2
78   0.457714    1         2
152  0.127055    1         2
164  0.626398    1         2
172  0.175720    0         2
174  0.484480    1         0
183  0.615547    1         2
249  0.410365    1         2
317  0.220966    1         2
381  0.248455    0         0
386  0.740235    1         2
467  0.419045    1         2
469  0.303959    0         0
492  0.481389    1         2
513  0.937525    0         0
582  0.410365    1         2
626  0.194923    1         1
644  0.303959    0         0
647  0.560305    1         0
755  0.228857    1         2
772  0.165724    0         2
774  0.363015    0         2
788  0.324740    1         2
803  0.134421    1         0
827  0.584047    1         0
```

```
# prompt: there is some outlier in age column and fare column so reome the outlier

# Remove outliers based on IQR method
def remove_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
```

```
    upper_bound = Q3 + 1.5 * IQR
    data_no_outliers = data[(data[column] >= lower_bound) & (data[column] <= upper_bound)]
    return data_no_outliers

# Remove outliers from 'Age' and 'Fare'
df_no_age_outliers = remove_outliers_iqr(df, 'Age')
df_no_outliers = remove_outliers_iqr(df_no_age_outliers, 'Fare')

# Now df_no_outliers contains the data with outliers removed
df_no_outliers.shape
```

```
(701, 11)
692    693    1    3    452    0.505022    0    0    80
781    782    1    1    214    0.293036    1    0    89
```

```
# prompt: change in dataset

# Assuming the provided code is already executed and df is available.
# ... (previous code) ...

# Remove outliers based on IQR method
def remove_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    #Instead of returning a new DataFrame, modify the existing one in place
    data.drop(data[(data[column] < lower_bound) | (data[column] > upper_bound)].index, inplace=True)
    return data

# Remove outliers from 'Age' and 'Fare'
df = remove_outliers_iqr(df, 'Age')
df = remove_outliers_iqr(df, 'Fare')

# Now df contains the data with outliers removed
df.shape
```

```
(701, 11)
```

Boxplot of Age

Boxplot of Fare

```
from sklearn.model_selection import train_test_split

# Define features and target
X = df.drop(['Survived_x', 'Name', 'Ticket'], axis=1)  # Dropping columns that are not useful for modeling
y = df['Survived_x']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
model = LogisticRegression()
```

```
model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
▾ LogisticRegression
LogisticRegression()
```

Start coding or generate with AI.

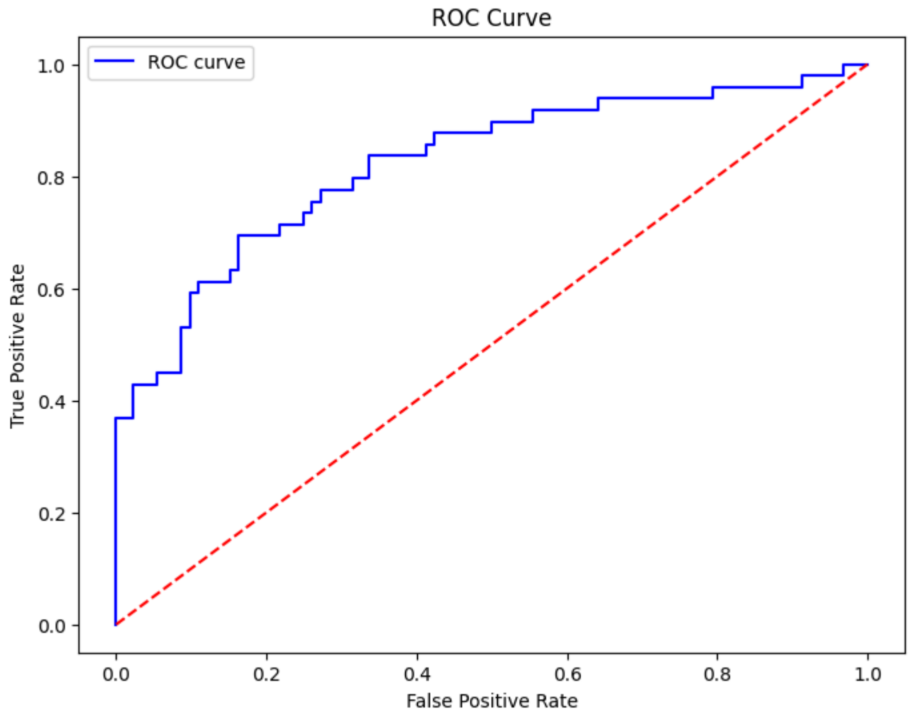
```
# Make predictions
y_pred = model.predict(X_train,)

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

```
Accuracy: 0.8071428571428572
```

```
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-72-e0cda1608a41> in <cell line: 0>()
----> 1 print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
      2 print("Classification Report:\n", classification_report(y_test, y_pred))

3 frames
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py in check_consistent_length(*arrays)
    473     uniques = np.unique(lengths)
    474     if len(uniques) > 1:
--> 475         raise ValueError(
    476             "Found input variables with inconsistent numbers of samples: %r"
    477             % [int(l) for l in lengths]

ValueError: Found input variables with inconsistent numbers of samples: [141, 560]
```

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(X_test)[:, 1])

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label='ROC curve')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')  # Diagonal line (random model)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()

# AUC Score
auc_score = roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])
print("AUC Score:", auc_score)
```




```
AUC Score: 0.8223158828748891
```

```
model.score(X_test,y_test)
```

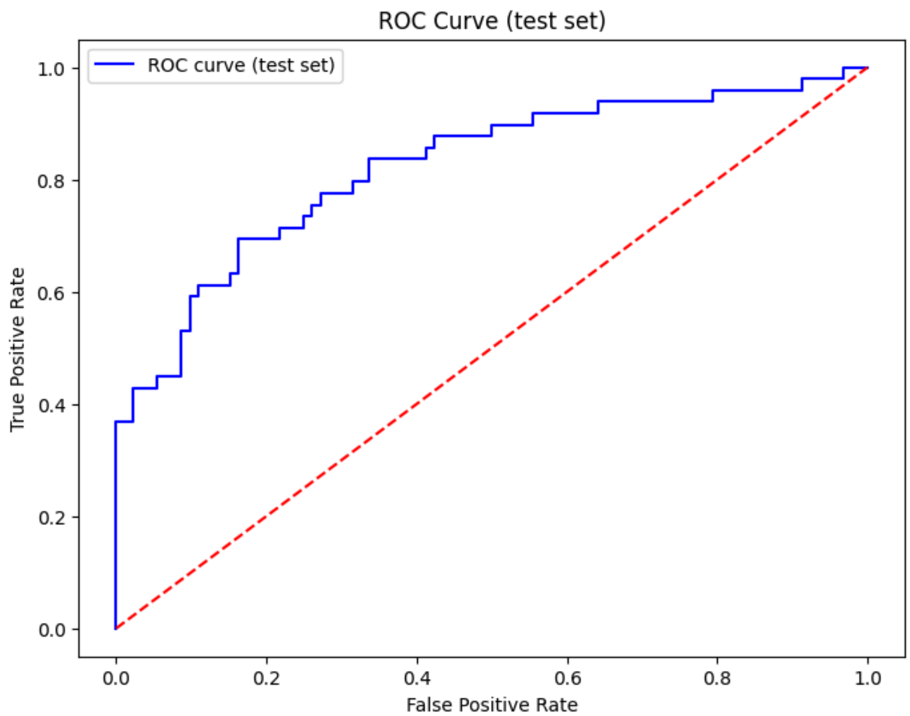
```
0.7730496453900709
```

```
model.score(X_train,y_train)
```

```
0.8071428571428572
```

```
# prompt: what next

# ... (Your existing code)

# Assuming the provided code is already executed and df, X_train, X_test, y_train, y_test are available.

# Make predictions on the test set
y_pred_test = model.predict(X_test)

# Evaluate the model on the test set
print("Accuracy on test set:", accuracy_score(y_test, y_pred_test))
print("Confusion Matrix (test set):\n", confusion_matrix(y_test, y_pred_test))
print("Classification Report (test set):\n", classification_report(y_test, y_pred_test))


# Calculate ROC curve for the test set
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, model.predict_proba(X_test)[:, 1])

# Plot ROC curve for the test set
plt.figure(figsize=(8, 6))
plt.plot(fpr_test, tpr_test, color='blue', label='ROC curve (test set)')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')  # Diagonal line (random model)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (test set)')
plt.legend()
plt.show()

# AUC Score for the test set
auc_score_test = roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])
print("AUC Score (test set):", auc_score_test)
```

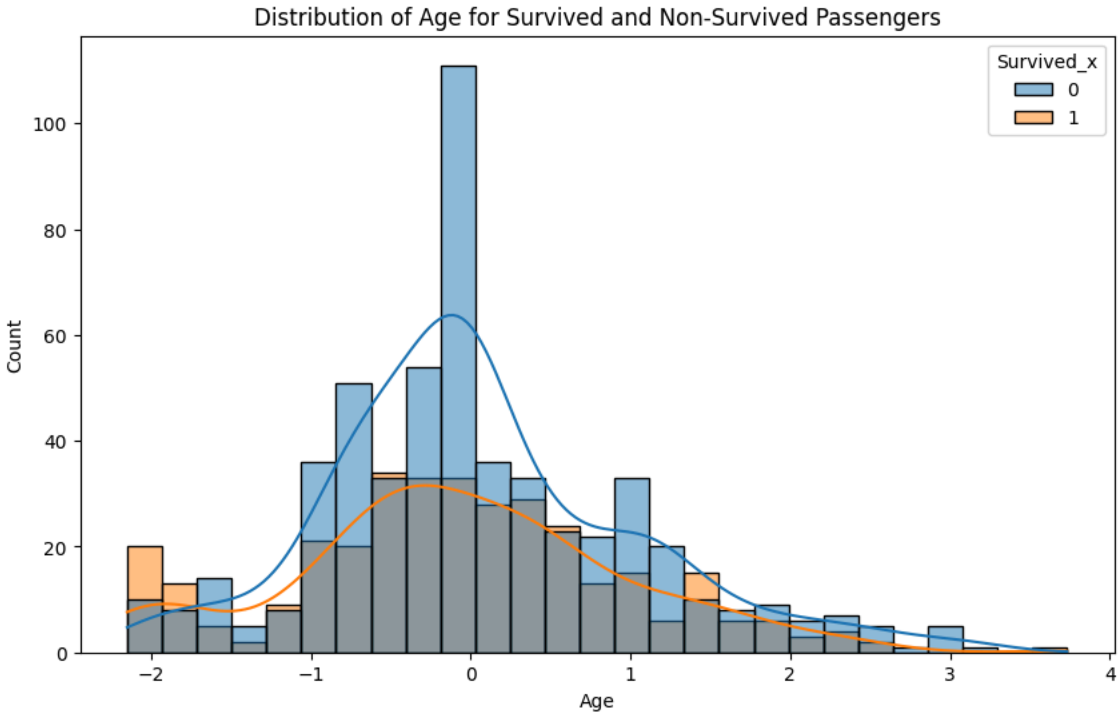
```
Accuracy on test set: 0.7730496453900709
Confusion Matrix (test set):
 [[83  9]
 [23 26]]
Classification Report (test set):
               precision    recall  f1-score   support

           0       0.78      0.90      0.84        92
           1       0.74      0.53      0.62        49

    accuracy                           0.77       141
   macro avg       0.76      0.72      0.73       141
weighted avg       0.77      0.77      0.76       141
```

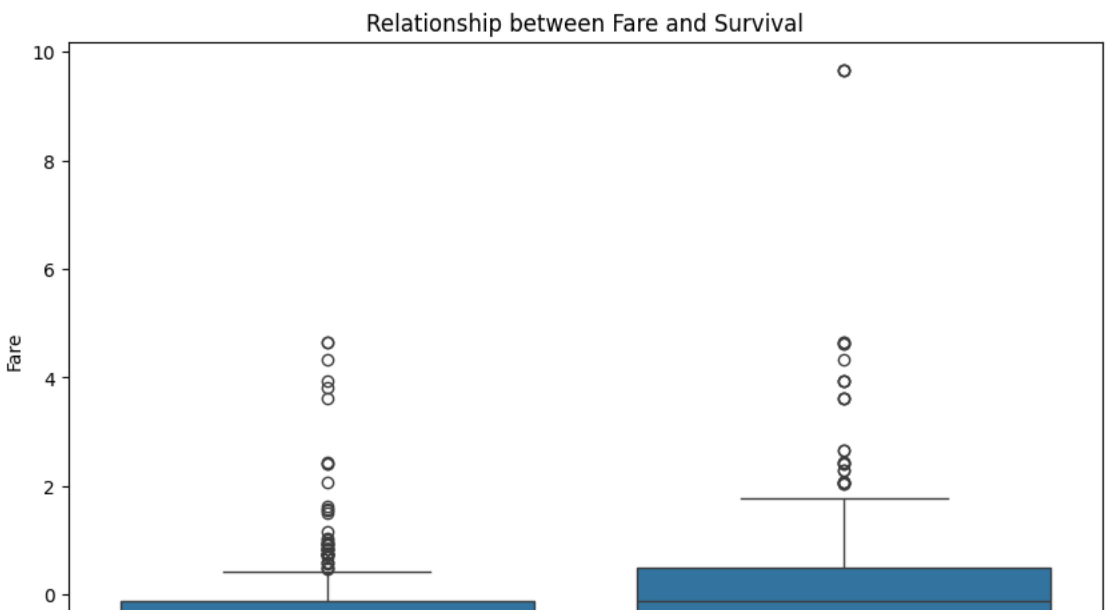



```
AUC Score (test set): 0.8223158828748891
```

```
# prompt: almost done with this project so go with some visualization for this

# Assuming the provided code is already executed and df is available.

# Visualize the distribution of ages for survived and non-survived passengers
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='Age', hue='Survived_x', kde=True)
plt.title('Distribution of Age for Survived and Non-Survived Passengers')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()

# Visualize the relationship between fare and survival
plt.figure(figsize=(10, 6))
sns.boxplot(x='Survived_x', y='Fare', data=df)
plt.title('Relationship between Fare and Survival')
plt.xlabel('Survived')
plt.ylabel('Fare')
plt.show()

# Visualize the survival rate based on passenger class (Pclass)
plt.figure(figsize=(8, 6))
sns.countplot(x='Pclass', hue='Survived_x', data=df)
plt.title('Survival Rate by Passenger Class')
plt.xlabel('Passenger Class')
plt.ylabel('Count')
plt.show()

# Visualize the survival rate based on sex
plt.figure(figsize=(8, 6))
sns.countplot(x='Sex', hue='Survived_x', data=df)
plt.title('Survival Rate by Sex')
```

```
plt.xlabel('Sex')
plt.ylabel('Count')
plt.show()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(12, 8))
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```






```
# prompt: save the clean data set
```