In [1]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error, accuracy_score, classification_report, confusion_matrix

from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier


import streamlit as st
from sklearn.preprocessing import StandardScaler
import warnings
```

In [4]:
```python
df = pd.read_csv(r"C:\Users\91797\Downloads\archive\Titanic-Dataset.csv")
```

In [5]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 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  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [6]:
```python
df.head
```

Out[6]:
```
<bound method NDFrame.head of      PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                  Name     Sex   Age  SibSp  \
0                              Braund, Mr. Owen Harris    male  22.0      1
1    Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                               Heikkinen, Miss. Laina  female  26.0      0
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                             Allen, Mr. William Henry    male  35.0      0
..                                                 ...     ...   ...    ...
886                              Montvila, Rev. Juozas    male  27.0      0
887                       Graham, Miss. Margaret Edith  female  19.0      0
888           Johnston, Miss. Catherine Helen "Carrie"  female   NaN      1
889                              Behr, Mr. Karl Howell    male  26.0      0
890                                Dooley, Mr. Patrick    male  32.0      0

     Parch            Ticket     Fare Cabin Embarked
0        0         A/5 21171   7.2500   NaN        S
1        0          PC 17599  71.2833   C85        C
2        0  STON/O2. 3101282   7.9250   NaN        S
3        0            113803  53.1000  C123        S
4        0            373450   8.0500   NaN        S
..     ...               ...      ...   ...      ...
886      0            211536  13.0000   NaN        S
887      0            112053  30.0000   B42        S
888      2        W./C. 6607  23.4500   NaN        S
889      0            111369  30.0000  C148        C
890      0            370376   7.7500   NaN        Q

[891 rows x 12 columns]>
```

In [7]:
```python
duplicate_rows = df[df.duplicated()]
print(f"Number of duplicate rows: {duplicate_rows.shape[0]}")
print(duplicate_rows)
```

```
Number of duplicate rows: 0
Empty DataFrame
Columns: [PassengerId, Survived, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, Embarked]
Index: []
```

In [8]:
```python
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)
```

In [9]:
```python
df = df.drop(columns=['PassengerId', 'Name', 'Ticket', 'Cabin'])
print(df.head())
```

```
   Survived  Pclass     Sex   Age  SibSp  Parch     Fare Embarked
0         0       3    male  22.0      1      0   7.2500        S
1         1       1  female  38.0      1      0  71.2833        C
2         1       3  female  26.0      0      0   7.9250        S
3         1       1  female  35.0      1      0  53.1000        S
4         0       3    male  35.0      0      0   8.0500        S
```

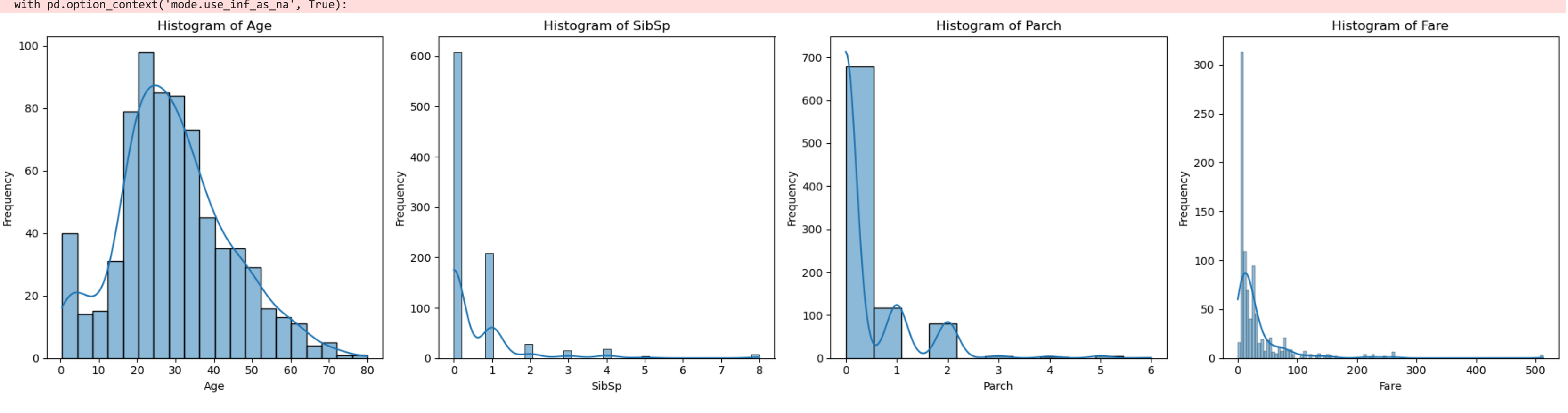
In [10]:
```python
df.describe()
```

Out[10]:

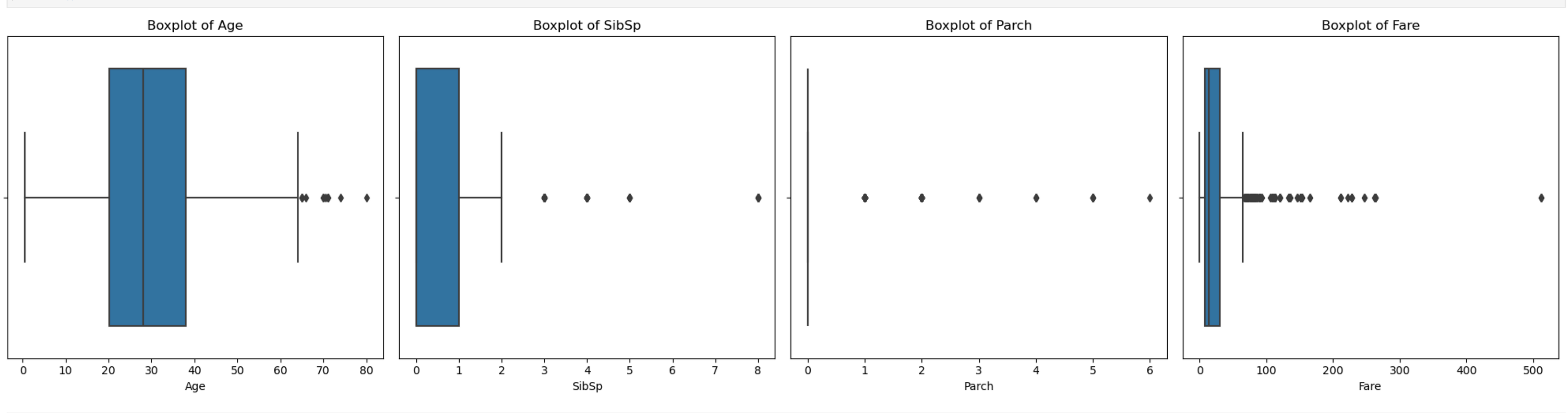
	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [11]:
```python
numerical_columns = ['Age', 'SibSp', 'Parch', 'Fare']

plt.figure(figsize=(20, 5))

for i, column in enumerate(numerical_columns, 1):
    plt.subplot(1, 4, i)
    sns.histplot(x=df[column], kde=True)
    plt.title(f'Histogram of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```

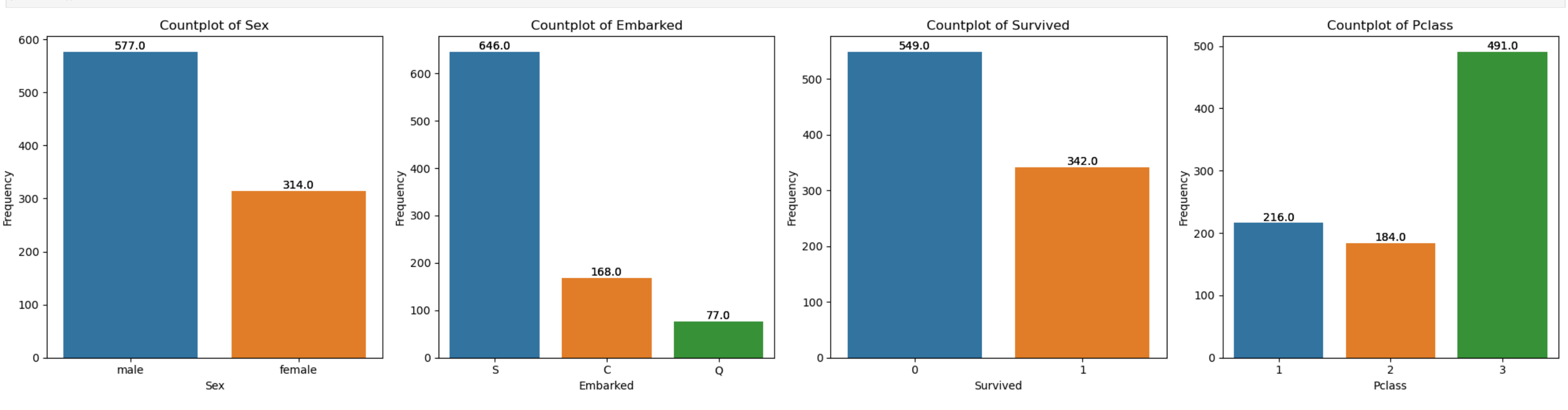
```
C:\Users\91797\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\91797\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\91797\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\91797\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [13]:
```python
numerical_columns = ['Age', 'SibSp', 'Parch', 'Fare']

plt.figure(figsize=(20, 5))

for i, column in enumerate(numerical_columns, 1):
    plt.subplot(1, 4, i)
    sns.boxplot(x=df[column])
    plt.title(f'Boxplot of {column}')

plt.tight_layout()
plt.show()
```



In [14]:
```python
categorical_columns = ['Sex', 'Embarked', 'Survived', 'Pclass']

plt.figure(figsize=(20, 5))  # Adjust figure size as needed

for i, column in enumerate(categorical_columns, 1):
    plt.subplot(1, 4, i)  # 1 row, 4 columns, i-th plot
    sns.countplot(x=df[column], ax=plt.gca()) # Use countplot
    ax = sns.countplot(x=df[column], ax=plt.gca())  # Store the axes object

    # Add labels to the bars
    for p in ax.patches:
        ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                    ha='center', va='center', fontsize=10, color='black', xytext=(0, 5),
                    textcoords='offset points')

    plt.title(f'Countplot of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```



In [15]:
```python
bins_age = [0, 18, 60, np.inf]
labels = ['Child', 'Adult', 'Elderly']

# Create a new column 'AgeGroup' with binned age values
df['AgeGroup'] = pd.cut(df['Age'], bins=bins_age, labels=labels)

print(df[['Age', 'AgeGroup']].head())
```

```
    Age AgeGroup
0  22.0    Adult
1  38.0    Adult
2  26.0    Adult
3  35.0    Adult
4  35.0    Adult
```

In [16]:
```python
df['SibSp'] = df['SibSp'].clip(upper=4)

#cap Parch at 4
df['Parch'] = df['Parch'].clip(upper=4)

#cap Fare at 200
df['Fare'] = df['Fare'].clip(upper=200)

print(df[['SibSp', 'Parch', 'Fare']].head())
```

```
   SibSp  Parch     Fare
0      1      0   7.2500
1      1      0  71.2833
2      0      0   7.9250
3      1      0  53.1000
4      0      0   8.0500
```

In [19]:
```python
df['FamilySize'] = df['SibSp'] + df['Parch'] + 1

# Create FareGroup column by binning the 'Fare' column into 4 equal-sized bins
df['FareGroup'] = pd.qcut(df['Fare'], q=4, labels=['Low', 'Mid', 'High', 'Very High'])

print(df[['SibSp', 'Parch', 'FamilySize', 'FareGroup']].head())
```

```
   SibSp  Parch  FamilySize  FareGroup
0      1      0           2        Low
1      1      0           2  Very High
2      0      0           1        Mid
3      1      0           2  Very High
4      0      0           1        Mid
```

In [18]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Survived    891 non-null    int64
 1   Pclass      891 non-null    int64
 2   Sex         891 non-null    object
 3   Age         714 non-null    float64
 4   SibSp       891 non-null    int64
 5   Parch       891 non-null    int64
 6   Fare        891 non-null    float64
 7   Embarked    891 non-null    object
 8   AgeGroup    714 non-null    category
 9   FamilySize  891 non-null    int64
 10  FareGroup   891 non-null    category
dtypes: category(2), float64(2), int64(5), object(2)
memory usage: 64.8+ KB
```

In [20]:
```python
df.describe()
```

Out[20]:

	Survived	Pclass	Age	SibSp	Parch	Fare	FamilySize
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.485971	0.373737	30.423994	1.859708
std	0.486592	0.836071	14.526497	0.908496	0.763306	38.770782	1.421924
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000	1.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400	1.000000
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200	1.000000
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000	2.000000
max	1.000000	3.000000	80.000000	4.000000	4.000000	200.000000	7.000000

In [22]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Survived    891 non-null    int64
 1   Pclass      891 non-null    int64
 2   Sex         891 non-null    object
 3   Age         714 non-null    float64
 4   SibSp       891 non-null    int64
 5   Parch       891 non-null    int64
 6   Fare        891 non-null    float64
 7   Embarked    891 non-null    object
 8   AgeGroup    714 non-null    category
 9   FamilySize  891 non-null    int64
 10  FareGroup   891 non-null    category
dtypes: category(2), float64(2), int64(5), object(2)
memory usage: 64.8+ KB
```

In [23]:
```python
numerical_features = ['Fare', 'FamilySize']
categorical_features = ['Pclass', 'Sex', 'Embarked', 'AgeGroup', 'FareGroup']

print("Numerical Features:", numerical_features)
print("Categorical Features:", categorical_features)
```

```
Numerical Features: ['Fare', 'FamilySize']
Categorical Features: ['Pclass', 'Sex', 'Embarked', 'AgeGroup', 'FareGroup']
```

In [26]:
```python
numerical_features = ['Fare', 'FamilySize']
categorical_features = ['Pclass', 'Sex', 'Embarked', 'AgeGroup', 'FareGroup']
target = 'Survived'

# Split the data into training and testing sets
X = df[numerical_features + categorical_features]
y = df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Preprocess the data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline([('scaler', StandardScaler()), ('passthrough', 'passthrough')]), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ])

pipeline_smote = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('smote', SMOTE(random_state=42)),
    ('classifier', LogisticRegression(solver='liblinear', random_state=42))
])

pipeline_smote.fit(X_train, y_train)
y_pred = pipeline_smote.predict(X_test)

# Predict (probabilities and classes)
y_pred_prob = pipeline_smote.predict_proba(X_test)[:, 1]
y_pred_class = pipeline_smote.predict(X_test)

# Evaluate (Classification Metrics)
print("\nLogistic Regression:")
accuracy = accuracy_score(y_test, y_pred_class)
print(f'Accuracy: {accuracy}')

print(classification_report(y_test, y_pred_class))

cm = confusion_matrix(y_test, y_pred_class)
print("Confusion Matrix:\n", cm)
```

```
Logistic Regression:
Accuracy: 0.7877094972067039
              precision    recall  f1-score   support

           0       0.87      0.75      0.81       105
           1       0.70      0.84      0.77        74

    accuracy                           0.79       179
   macro avg       0.79      0.80      0.79       179
weighted avg       0.80      0.79      0.79       179

Confusion Matrix:
 [[79 26]
 [12 62]]
```

In [27]:
```python
X = df[numerical_features + categorical_features]
y = df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Preprocess the data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline([('scaler', StandardScaler()), ('passthrough', 'passthrough')]), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ])

# --- Pipeline with SMOTE and Random Forest ---
pipeline_rf_smote = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('smote', SMOTE(random_state=42)),  # Apply SMOTE for oversampling
    ('classifier', RandomForestClassifier(random_state=42))  # Random Forest classifier
])

# --- Train the model ---
pipeline_rf_smote.fit(X_train, y_train)

# --- Make Predictions ---
y_pred = pipeline_rf_smote.predict(X_test)

# --- Evaluate the Model ---
print("\nRandom Forest Classifier:")
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')

print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
```

```
Random Forest Classifier:
Accuracy: 0.8212290502793296
              precision    recall  f1-score   support

           0       0.84      0.86      0.85       105
           1       0.79      0.77      0.78        74

    accuracy                           0.82       179
   macro avg       0.82      0.81      0.81       179
weighted avg       0.82      0.82      0.82       179

Confusion Matrix:
 [[90 15]
 [17 57]]
```

In [28]:
```python
pipeline_svm_smote = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('smote', SMOTE(random_state=42)),  # Apply SMOTE
    ('classifier', SVC(random_state=42))  # SVM classifier
])

# --- Train the model ---
pipeline_svm_smote.fit(X_train, y_train)

# --- Make Predictions ---
y_pred = pipeline_svm_smote.predict(X_test)

# --- Evaluate the Model ---
print("\nSVM:")
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')

print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
```

```
SVM:
Accuracy: 0.8156424581005587
              precision    recall  f1-score   support

           0       0.85      0.84      0.84       105
           1       0.77      0.78      0.78        74

    accuracy                           0.82       179
   macro avg       0.81      0.81      0.81       179
weighted avg       0.82      0.82      0.82       179

Confusion Matrix:
 [[88 17]
 [16 58]]
```

In [29]:
```python
pipeline_gb_smote = Pipeline(steps=[
    ('preprocessor', preprocessor),  # Use the same preprocessor
    ('smote', SMOTE(random_state=42)),
    ('classifier', GradientBoostingClassifier(random_state=42))
])

pipeline_gb_smote.fit(X_train, y_train)
y_pred_gb = pipeline_gb_smote.predict(X_test)

accuracy_gb = accuracy_score(y_test, y_pred_gb)
print("\nGradient Boosting:")
print(f'Accuracy: {accuracy_gb}')
print(classification_report(y_test, y_pred_gb))
cm_gb = confusion_matrix(y_test, y_pred_gb)
print("Confusion Matrix:\n", cm_gb)
```

```
Gradient Boosting:
Accuracy: 0.8491620111731844
              precision    recall  f1-score   support

           0       0.88      0.86      0.87       105
           1       0.81      0.84      0.82        74

    accuracy                           0.85       179
   macro avg       0.84      0.85      0.85       179
weighted avg       0.85      0.85      0.85       179

Confusion Matrix:
 [[90 15]
 [12 62]]
```

In [30]:
```python
pipeline_nn_smote = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('smote', SMOTE(random_state=42)),
    ('classifier', MLPClassifier(random_state=42, max_iter=500, early_stopping=True))  # Increased max_iter, added early_stopping
])

pipeline_nn_smote.fit(X_train, y_train)
y_pred_nn = pipeline_nn_smote.predict(X_test)

accuracy_nn = accuracy_score(y_test, y_pred_nn)
print("\nNeural Network:")
print(f'Accuracy: {accuracy_nn}')
print(classification_report(y_test, y_pred_nn))
cm_nn = confusion_matrix(y_test, y_pred_nn)
print("Confusion Matrix:\n", cm_nn)
```

```
Neural Network:
Accuracy: 0.7597765363128491
              precision    recall  f1-score   support

           0       0.85      0.71      0.78       105
           1       0.67      0.82      0.74        74

    accuracy                           0.76       179
   macro avg       0.76      0.77      0.76       179
weighted avg       0.78      0.76      0.76       179

Confusion Matrix:
 [[75 30]
 [13 61]]
```

In [31]:
```python
param_grid = {
    'classifier__n_estimators': [50, 100],  # Number of trees
    'classifier__max_depth': [None, 10],      # Maximum depth of trees
    'classifier__min_samples_split': [2, 5],   # Minimum samples to split a node
    'classifier__min_samples_leaf': [1, 2],     # minimum samples in a leaf node
    'classifier__criterion': ['gini', 'entropy'] # Splitting criteria
}
grid_search = GridSearchCV(pipeline_rf_smote, param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=1)  # 5-fold CV, use all cores

grid_search.fit(X_train, y_train)

# --- Best Model and Best Parameters ---
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# --- Evaluate the Best Model on the Test Set ---
y_pred = best_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy (Best Model): {accuracy}')

print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

# --- Cross-Validation Score (Optional) ---
cv_scores = cross_val_score(best_model, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)
print(f"Cross-Validation Scores (Best Model): {cv_scores}")
print(f"Mean CV Accuracy (Best Model): {cv_scores.mean()}")
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
Best Hyperparameters: {'classifier__criterion': 'gini', 'classifier__max_depth': 10, 'classifier__min_samples_leaf': 1, 'classifier__min_samples_split': 5, 'classifier__n_estimators': 50}
Accuracy (Best Model): 0.8268156424581006
              precision    recall  f1-score   support

           0       0.83      0.89      0.86       105
           1       0.82      0.74      0.78        74

    accuracy                           0.83       179
   macro avg       0.83      0.81      0.82       179
weighted avg       0.83      0.83      0.83       179

Confusion Matrix:
 [[93 12]
 [19 55]]
Cross-Validation Scores (Best Model): [0.8041958  0.8041958  0.81690141 0.78873239 0.85211268]
Mean CV Accuracy (Best Model): 0.8132276174529695
```

In [32]:
```python
df.head()
```

Out[32]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	AgeGroup	FamilySize	FareGroup
0	0	3	male	22.0	1	0	7.2500	S	Adult	2	Low
1	1	1	female	38.0	1	0	71.2833	C	Adult	2	Very High
2	1	3	female	26.0	0	0	7.9250	S	Adult	1	Mid
3	1	1	female	35.0	1	0	53.1000	S	Adult	2	Very High
4	0	3	male	35.0	0	0	8.0500	S	Adult	1	Mid