In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df=pd.read_csv("/content/titanic.csv")
df
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

In [3]:
```
df.describe()
```

Out[3]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	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	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [4]:
```
#Missing values
df.isna().sum()
```

Out[4]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [5]:
```
#handling misssing value
#Numerical variables
#Distribution of the numerical variables
age=df['Age'].mean()
df['Age1']=df['Age'].fillna(age)
df=df.drop(['Age'],axis=1)
```

In [6]:
```
df.isna().sum()
```

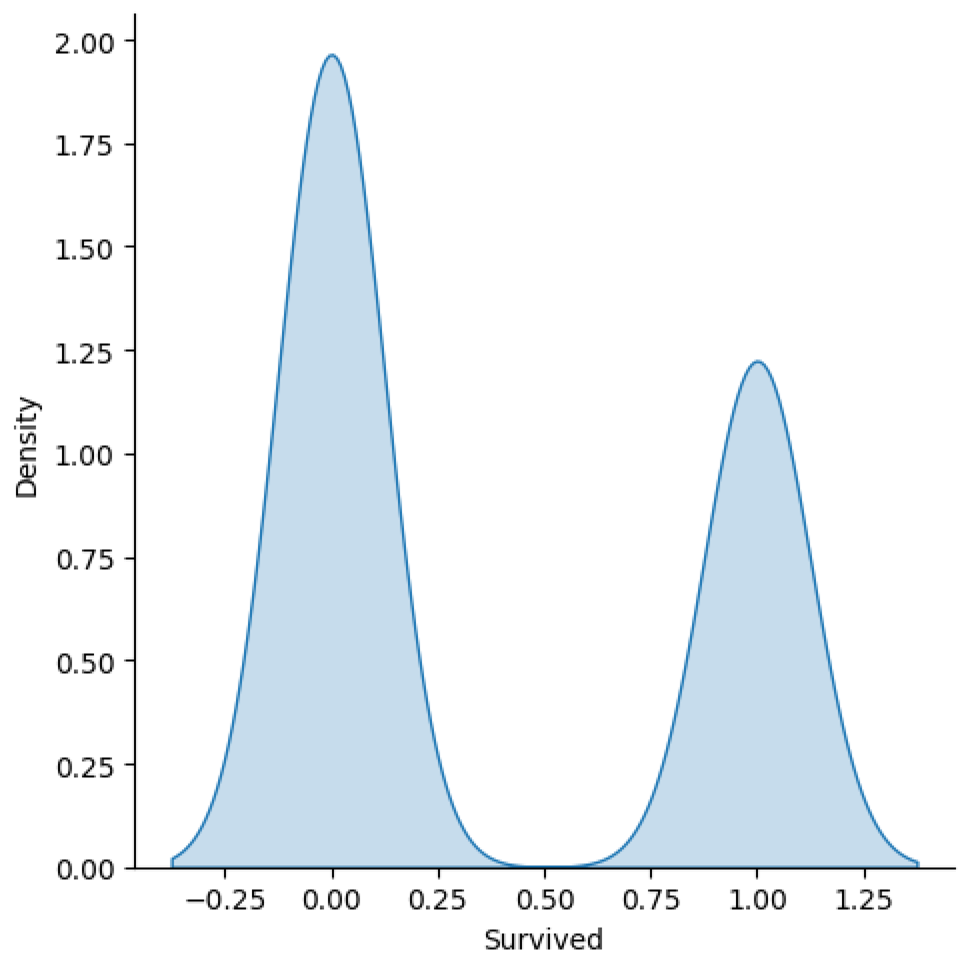
Out[6]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
Age1             0
dtype: int64
```

In [7]:
```
# Categorical variables
print(df['Embarked'].value_counts())
```

```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

In [8]:
```
#Categorical variables
sns.displot(df,x='Survived',kind="kde",fill=True)
```

Out[8]:
```
<seaborn.axisgrid.FacetGrid at 0x7f5f2cf7b8e0>
```



In [9]:
```
df.corr()
```

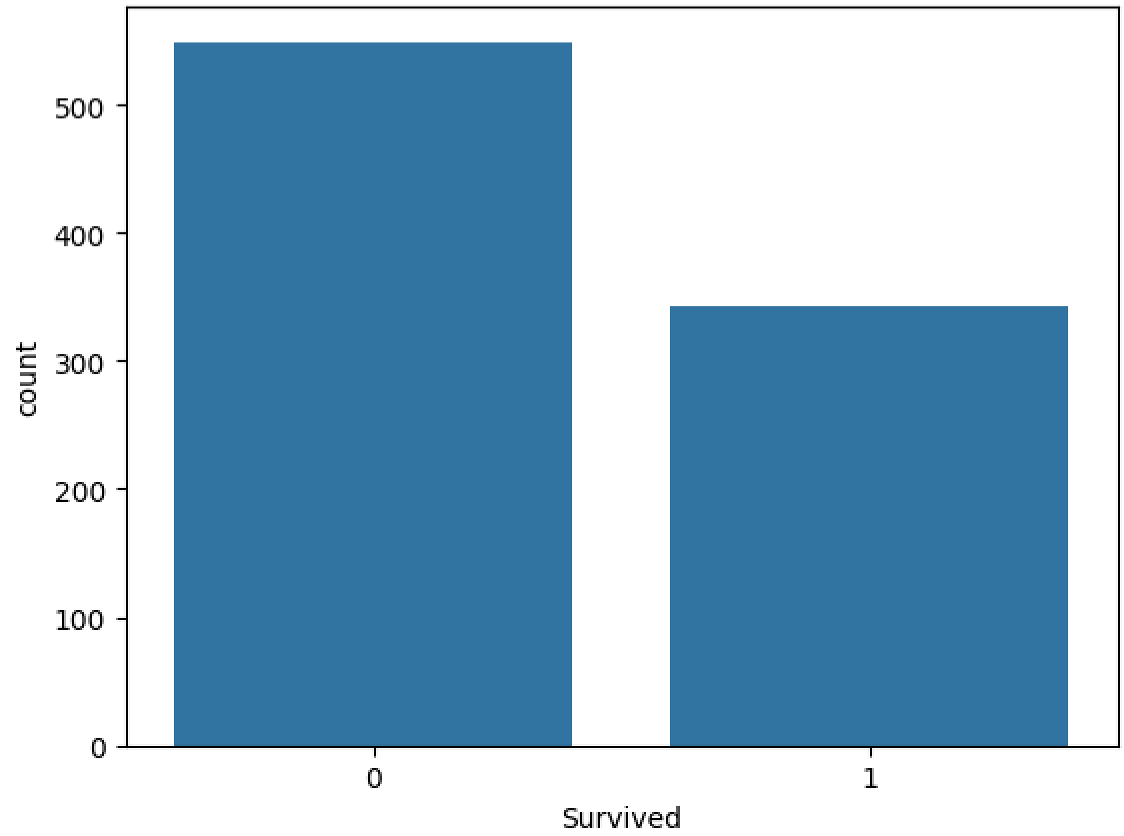
```
<ipython-input-9-2f6f6606aa2c>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In
a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence t
his warning.
  df.corr()
```

Out[9]:

	PassengerId	Survived	Pclass	SibSp	Parch	Fare	Age1
PassengerId	1.000000	-0.005007	-0.035144	-0.057527	-0.001652	0.012658	0.033207
Survived	-0.005007	1.000000	-0.338481	-0.035322	0.081629	0.257307	-0.069809
Pclass	-0.035144	-0.338481	1.000000	0.083081	0.018443	-0.549500	-0.331339
SibSp	-0.057527	-0.035322	0.083081	1.000000	0.414838	0.159651	-0.232625
Parch	-0.001652	0.081629	0.018443	0.414838	1.000000	0.216225	-0.179191
Fare	0.012658	0.257307	-0.549500	0.159651	0.216225	1.000000	0.091566
Age1	0.033207	-0.069809	-0.331339	-0.232625	-0.179191	0.091566	1.000000

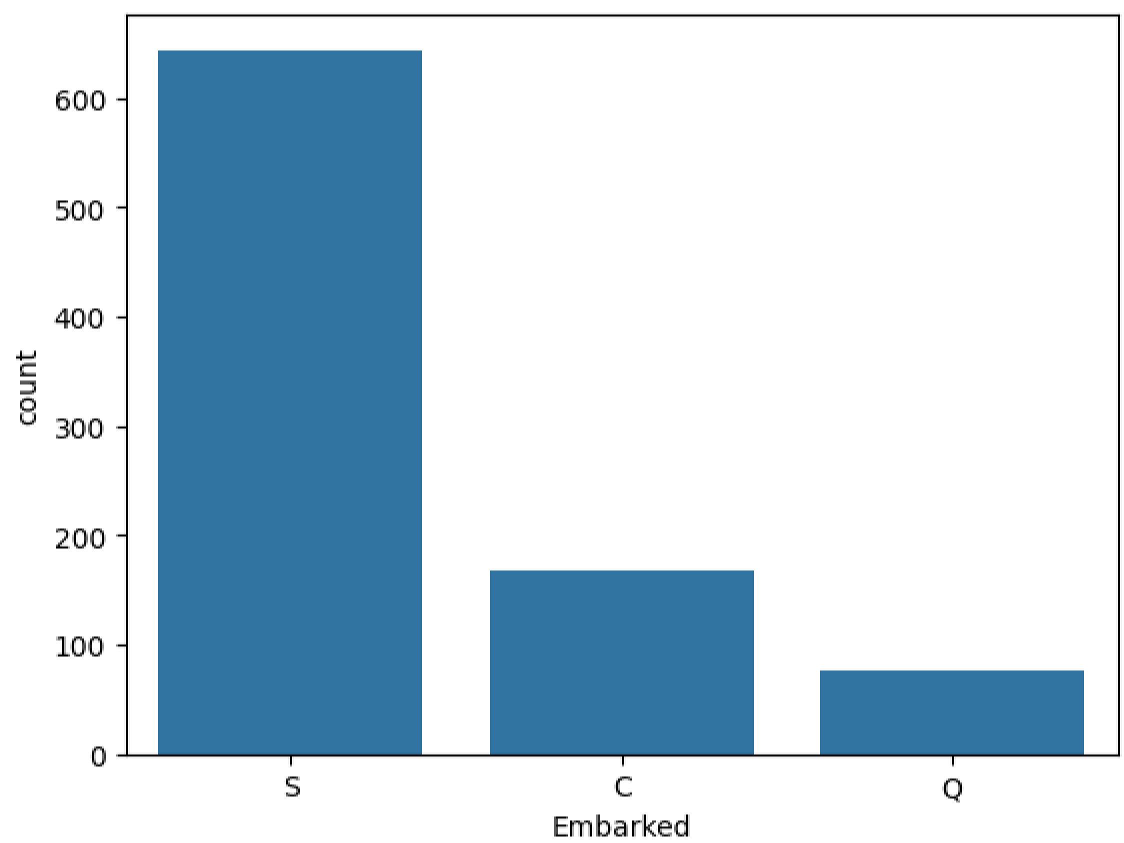
In [10]:
```
#Categorical variables
sns.countplot(x='Survived',data=df)
```

Out[10]:
```
<Axes: xlabel='Survived', ylabel='count'>
```



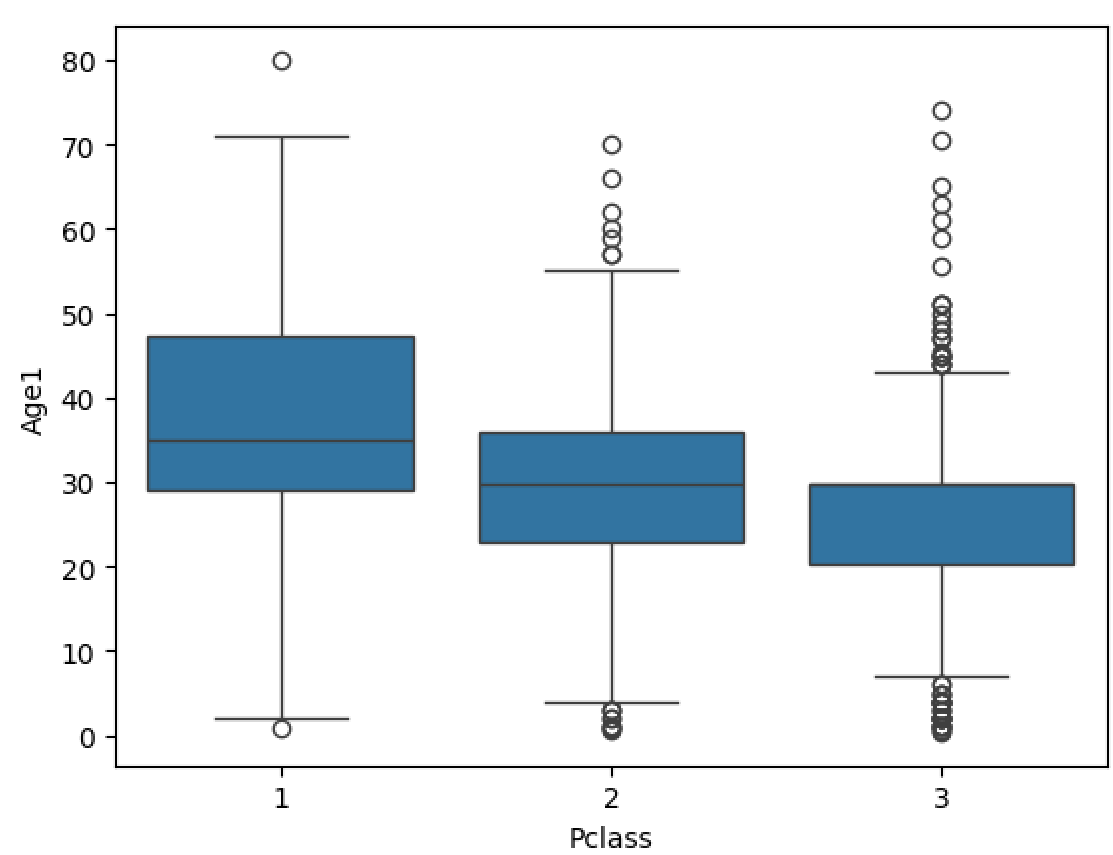
In [11]:
```
sns.countplot(x='Embarked',data=df)
```

Out[11]: <Axes: xlabel='Embarked', ylabel='count'>



In [12]:
```
sns.boxplot(x='Pclass', y='Age1', data =df)
```

Out[12]: <Axes: xlabel='Pclass', ylabel='Age1'>



In [13]:
```
df=df.drop(["Cabin"],axis=1)
```

In [14]:
```
df.head()
```

Out[14]:

	PassengerId	Survived	Pclass	Name	Sex	SibSp	Parch	Ticket	Fare	Embarked	Age1
0	1	0	3	Braund, Mr. Owen Harris	male	1	0	A/5 21171	7.2500	S	22.0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	1	0	PC 17599	71.2833	C	38.0
2	3	1	3	Heikkinen, Miss. Laina	female	0	0	STON/O2. 3101282	7.9250	S	26.0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	1	0	113803	53.1000	S	35.0
4	5	0	3	Allen, Mr. William Henry	male	0	0	373450	8.0500	S	35.0

In [15]:

```
df=df.rename(columns={'Age1':'Age'})
df.head()
```

Out[15]:

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

In [16]:

```
#drop null value
df.dropna(inplace=True)
```

In [17]:

```
df=df.drop(['PassengerId','Name','Ticket'],axis=1)
df
```

Out[17]:

	Survived	Pclass	Sex	SibSp	Parch	Fare	Embarked	Age
0	0	3	male	1	0	7.2500	S	22.000000
1	1	1	female	1	0	71.2833	C	38.000000
2	1	3	female	0	0	7.9250	S	26.000000
3	1	1	female	1	0	53.1000	S	35.000000
4	0	3	male	0	0	8.0500	S	35.000000
...	...	...	...	...	...	...	...	...
886	0	2	male	0	0	13.0000	S	27.000000
887	1	1	female	0	0	30.0000	S	19.000000
888	0	3	female	1	2	23.4500	S	29.699118
889	1	1	male	0	0	30.0000	C	26.000000
890	0	3	male	0	0	7.7500	Q	32.000000

889 rows × 8 columns

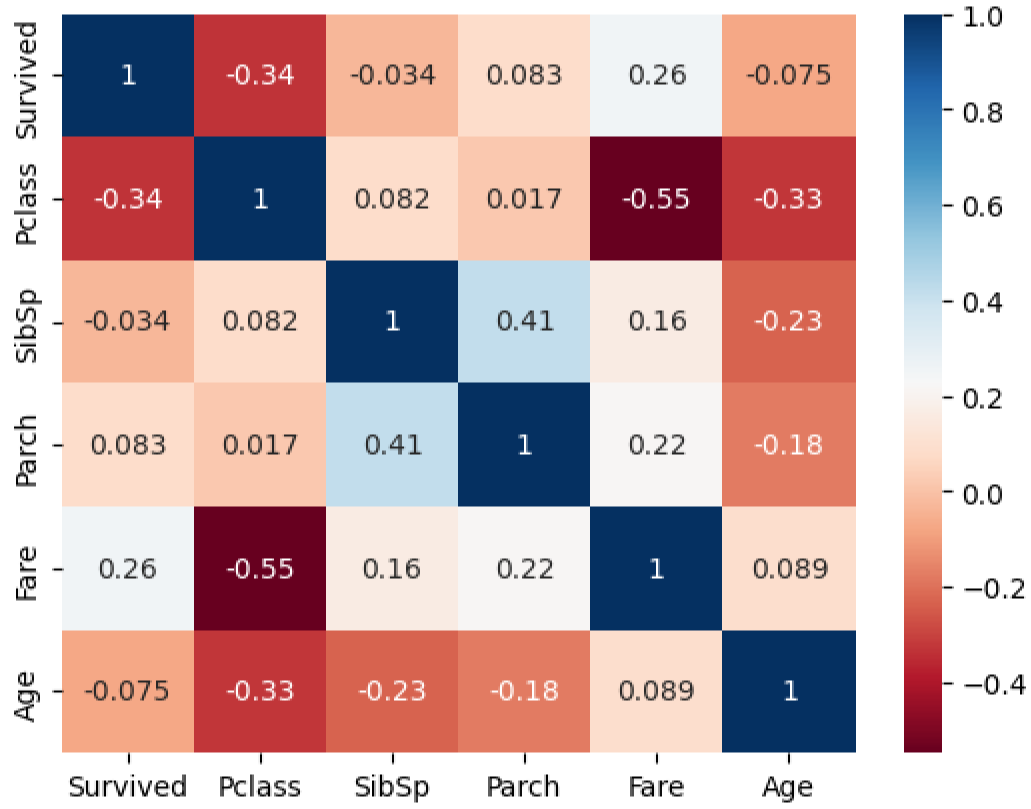
In [18]:

```
sns.heatmap(df.corr(),annot=True,cmap='RdBu')
```

```
<ipython-input-18-7c8bda1b552c>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence t
his warning.
  sns.heatmap(df.corr(),annot=True,cmap='RdBu')
```

Out[18]:

```
<Axes: >
```



In [19]:
```
from sklearn.preprocessing import LabelEncoder
df1 = df.copy()
e1 = LabelEncoder()
e2 = LabelEncoder()
df1.Sex = e1.fit_transform(df1.Sex)
df1.Embarked = e2.fit_transform(df1.Embarked)
df1
```

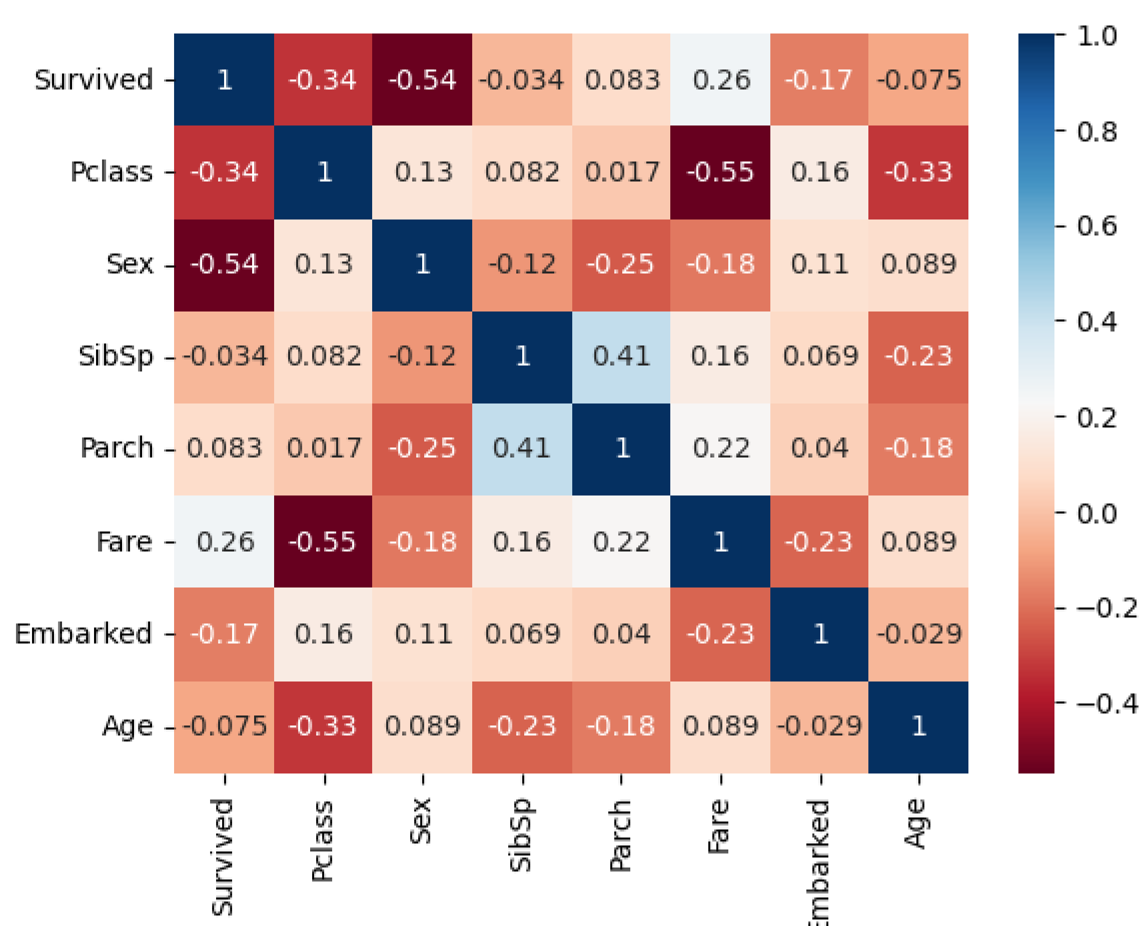
Out[19]:

	Survived	Pclass	Sex	SibSp	Parch	Fare	Embarked	Age
0	0	3	1	1	0	7.2500	2	22.000000
1	1	1	0	1	0	71.2833	0	38.000000
2	1	3	0	0	0	7.9250	2	26.000000
3	1	1	0	1	0	53.1000	2	35.000000
4	0	3	1	0	0	8.0500	2	35.000000
...	...	...	...	...	...	...	...	...
886	0	2	1	0	0	13.0000	2	27.000000
887	1	1	0	0	0	30.0000	2	19.000000
888	0	3	0	1	2	23.4500	2	29.699118
889	1	1	1	0	0	30.0000	0	26.000000
890	0	3	1	0	0	7.7500	1	32.000000

889 rows × 8 columns

In [20]:
```
sns.heatmap(df1.corr(),annot=True,cmap='RdBu')
```

Out[20]: <Axes: >



In [21]:
```
df1.head()
```

Out[21]:

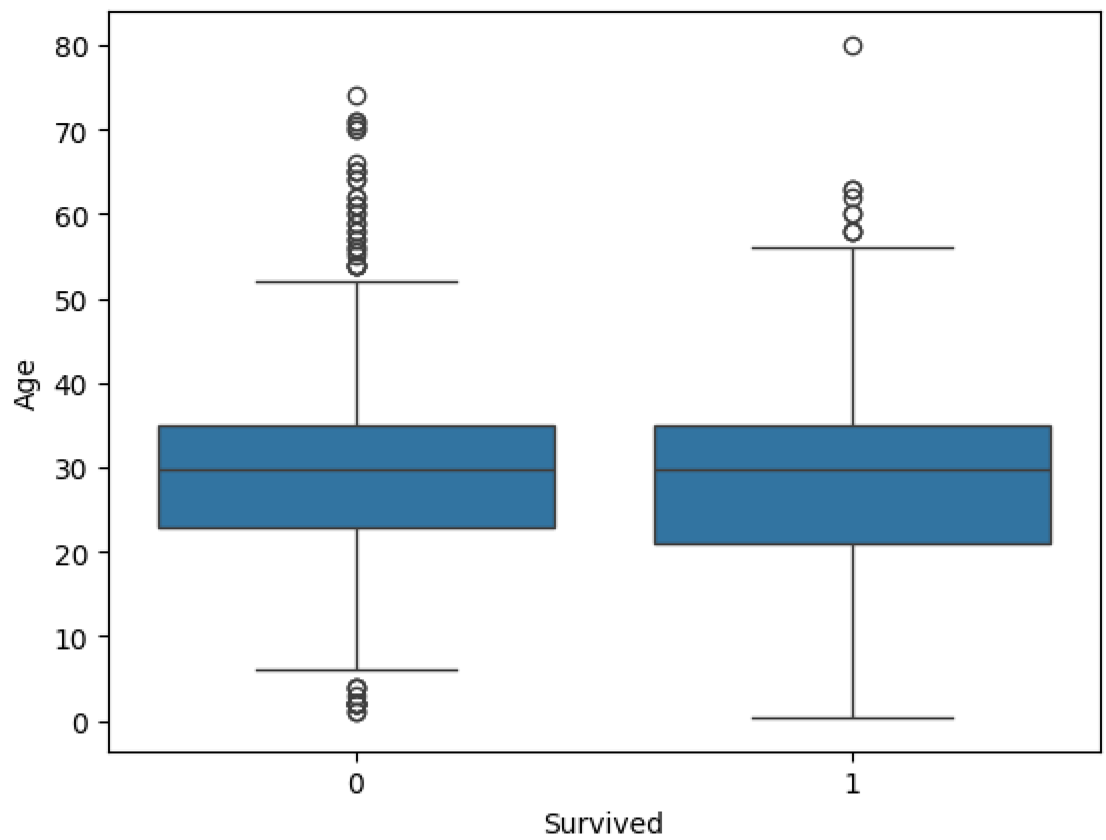
	Survived	Pclass	Sex	SibSp	Parch	Fare	Embarked	Age
0	0	3	1	1	0	7.2500	2	22.0
1	1	1	0	1	0	71.2833	0	38.0
2	1	3	0	0	0	7.9250	2	26.0
3	1	1	0	1	0	53.1000	2	35.0
4	0	3	1	0	0	8.0500	2	35.0

In [22]:

```
sns.boxplot(x='Survived', y='Age', data =df)
```

Out[22]:

```
<Axes: xlabel='Survived', ylabel='Age'>
```

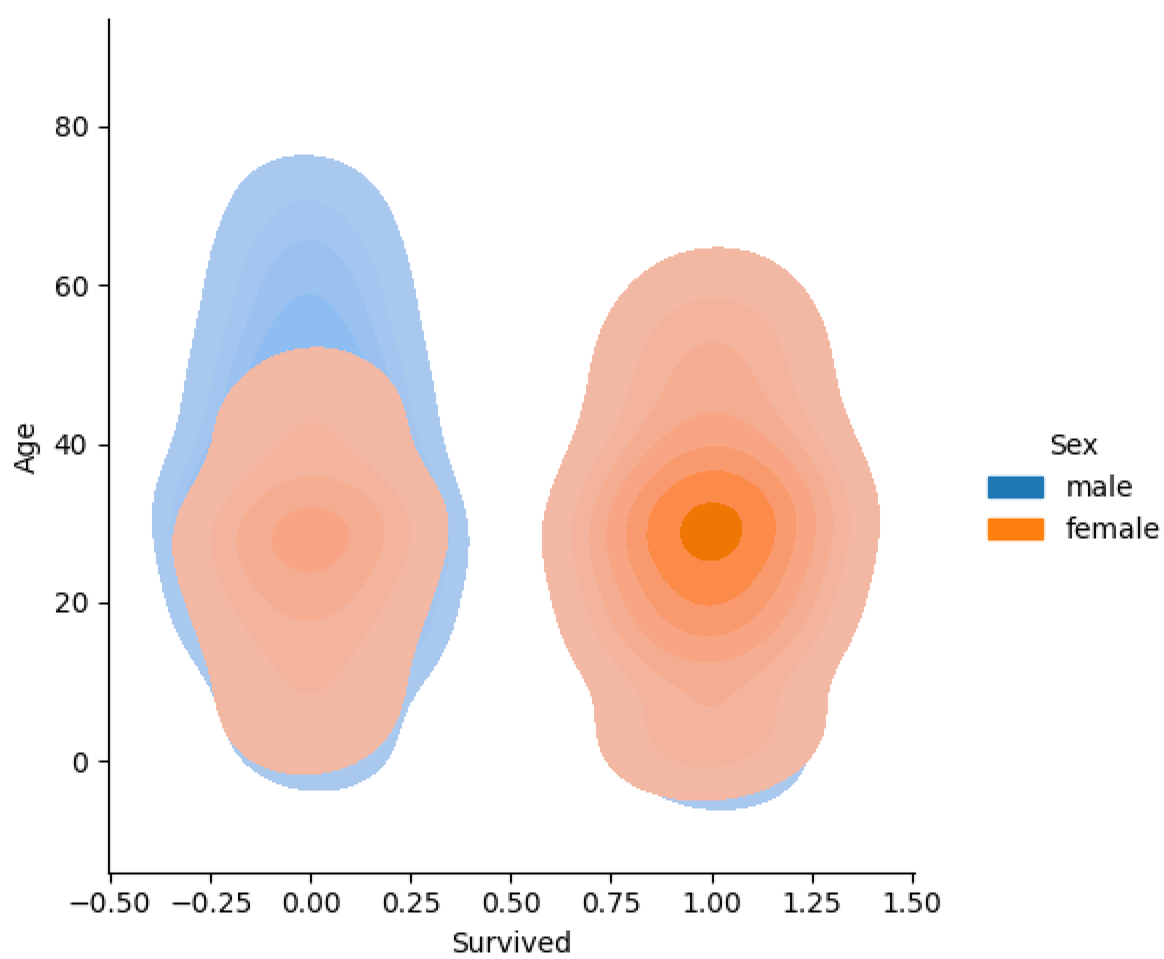


In [23]:

```
# Distribution of the numerical variables,Categorical variables
sns.displot(df,x='Survived',y='Age',hue='Sex',kind="kde",fill=True)
```

Out[23]:

```
<seaborn.axisgrid.FacetGrid at 0x7f5f28c7fbe0>
```

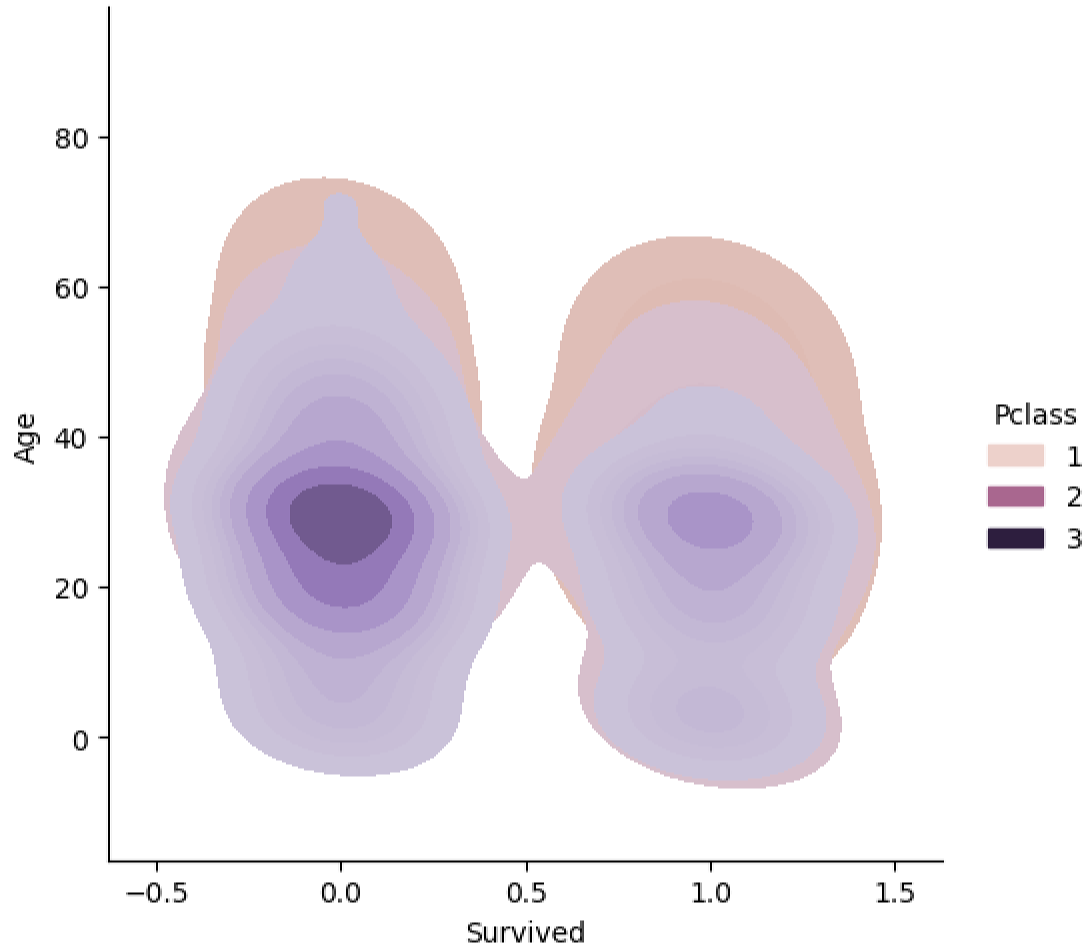


In [24]:

```
# Distribution of the numerical variables,Categorical variables
sns.displot(df,x='Survived',y='Age',hue='Pclass',kind="kde",fill=True)
```

Out[24]:

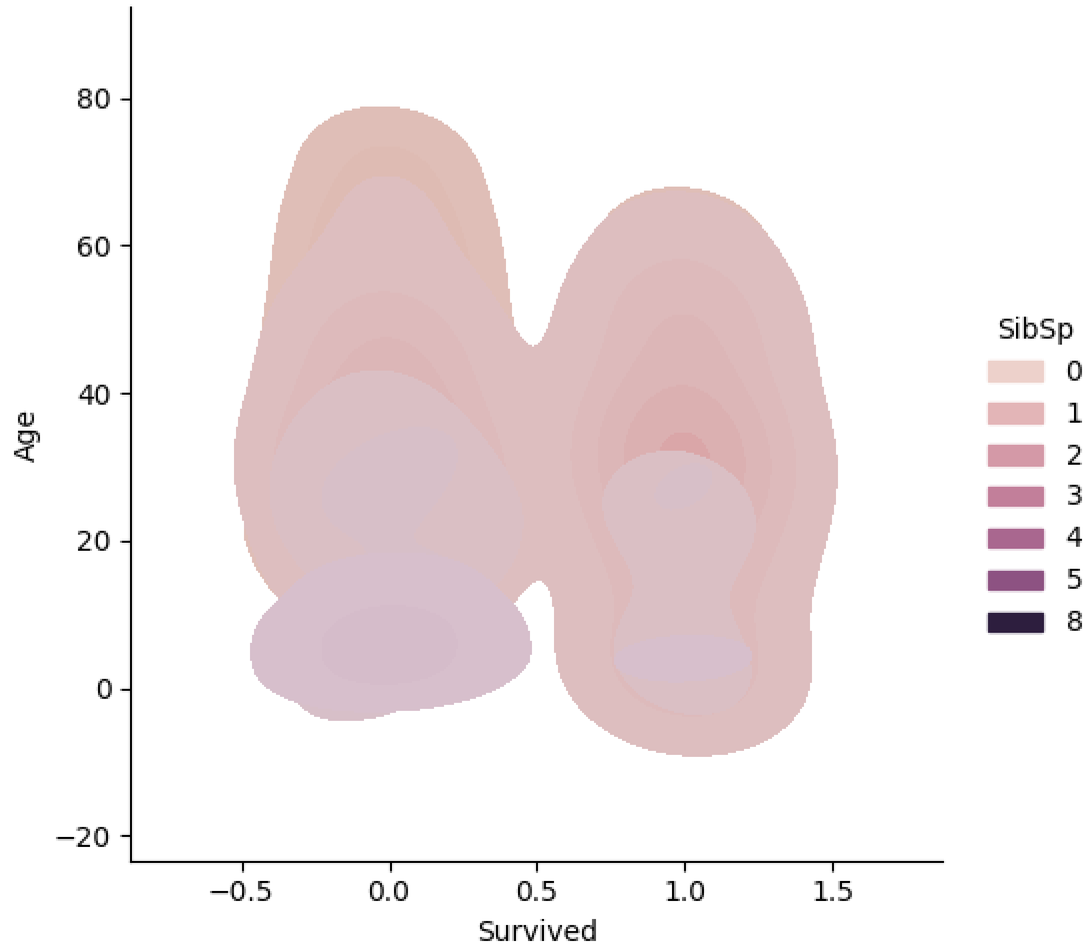
```
<seaborn.axisgrid.FacetGrid at 0x7f5f26a50220>
```



In [25]:
```
# Distribution of the numerical variables,Categorical variables
sns.displot(df,x='Survived',y='Age',hue='SibSp',kind="kde",fill=True)
```

```
<ipython-input-25-93673a6db70d>:2: UserWarning: KDE cannot be estimated (0 variance or perfect covariance). Pass `warn_
singular=False` to disable this warning.
  sns.displot(df,x='Survived',y='Age',hue='SibSp',kind="kde",fill=True)
<ipython-input-25-93673a6db70d>:2: UserWarning: KDE cannot be estimated (0 variance or perfect covariance). Pass `warn_
singular=False` to disable this warning.
  sns.displot(df,x='Survived',y='Age',hue='SibSp',kind="kde",fill=True)
```

Out[25]: <seaborn.axisgrid.FacetGrid at 0x7f5f28de9bd0>

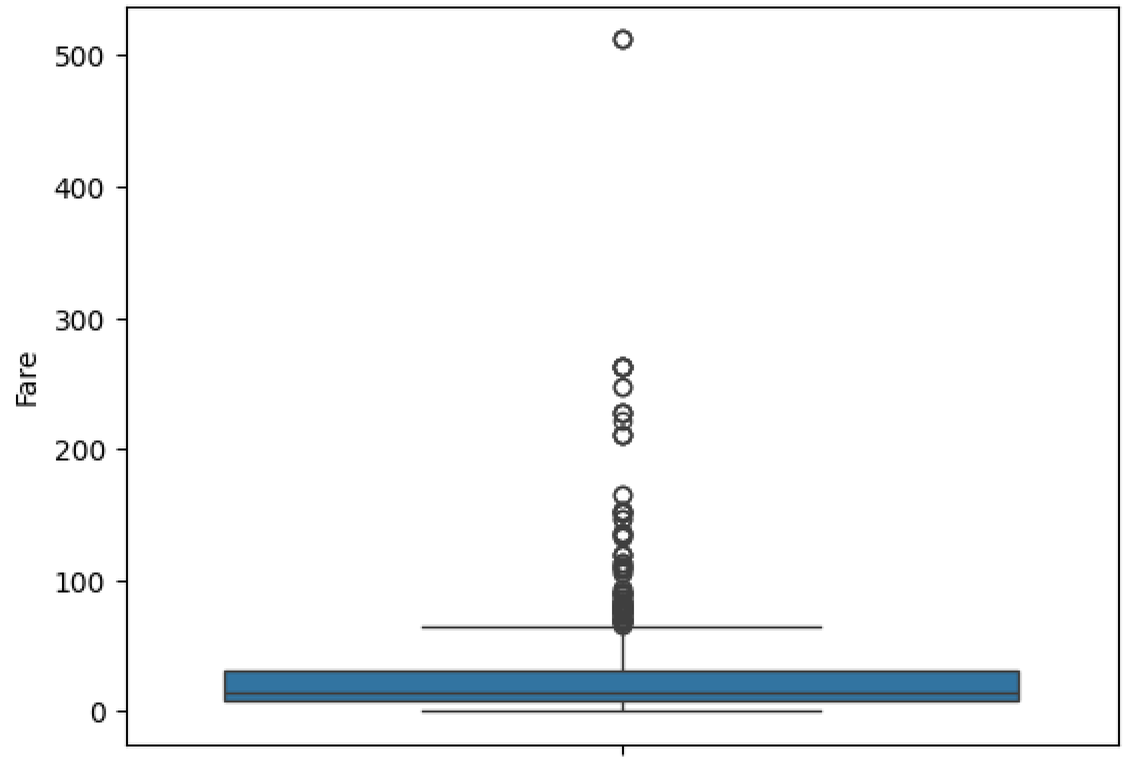


In [26]:
```
import plotly.express as px


# Create the 3D displot
fig = px.violin(df,x='Survived',y='Age', color='Sex')

# Show the plot
fig.show()
```

In [27]:
```
import plotly.express as px


# Create the 3D displot
fig = px.violin(df,x='Survived',y='Age', color='Pclass')

# Show the plot
fig.show()
```

In [28]:
```
sns.boxplot(df['Fare'])
```

Out[28]:
```
<Axes: ylabel='Fare'>
```



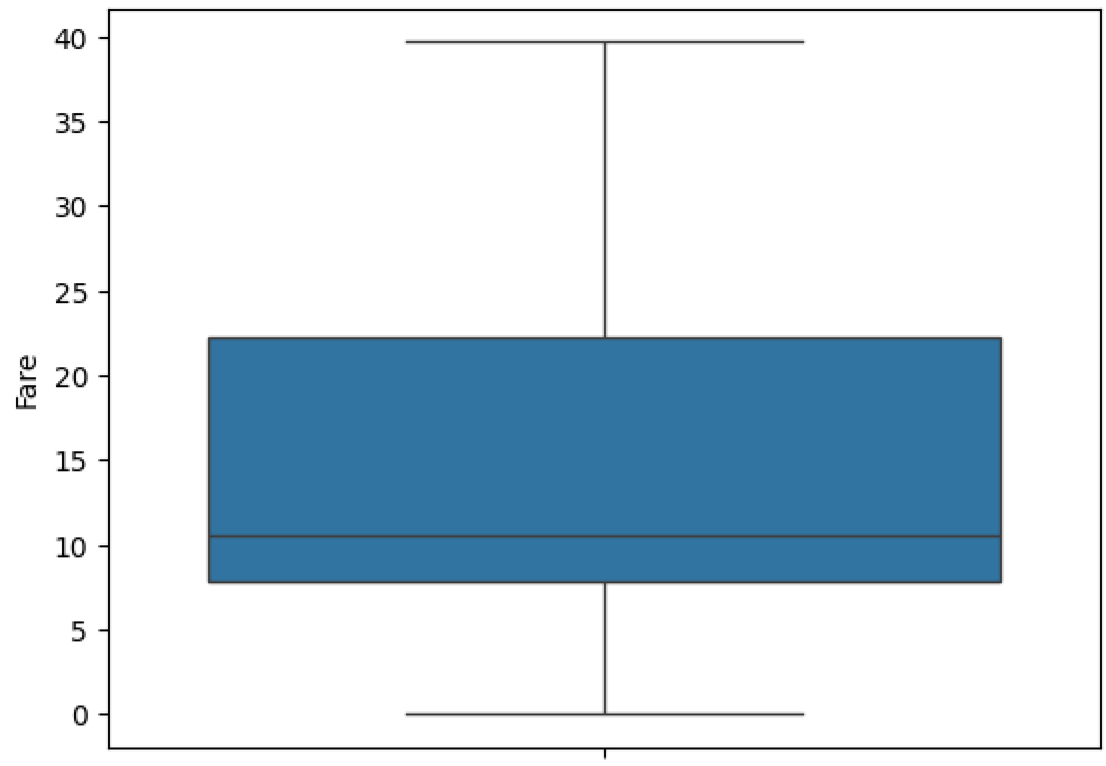
In [29]:
```
# Outliers
import numpy as np
outlier = []
def detect_z(data):
    thres = 3
    mean = np.mean(data)
    std = np.std(data)

    for i in data:
        z = (i-mean)/std
        if (np.abs(z) > thres):
            outlier.append(i)
    print(outlier)

detect_z(df['Fare'])
```

```
[263.0, 263.0, 247.5208, 512.3292, 247.5208, 262.375, 263.0, 211.5, 227.525, 263.0, 221.7792, 227.525, 512.3292, 211.33
75, 227.525, 227.525, 211.3375, 512.3292, 262.375, 211.3375]
```

In [30]:
```
for i in df.index:
    if df.loc[i,'Fare']>40:
        df.drop(i, inplace=True)
```

In [31]:
```
sns.boxplot(df['Fare'])
print(df.shape)
```

```
(715, 8)
```



In [32]:
```
#IQR technique for outlier

import numpy as np
def remove_outliers_iqr(data):
    # Calculate the first and third quartiles
    q1 = np.percentile(data, 25)
    q3 = np.percentile(data, 75)

    # Calculate the IQR (Interquartile Range)
    iqr = q3 - q1

    # Define the lower and upper bounds for outliers
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr

    # Identify indices of outliers
    outliers = np.where((data < lower_bound) | (data > upper_bound))
```

```
    # Remove outliers from the original data
    data_no_outliers = data[(data >= lower_bound) & (data <= upper_bound)]

    return data_no_outliers, outliers

# Example usage:
data = df1['Fare']
cleaned_data, outlier_indices = remove_outliers_iqr(data)

print("Original data:", data)
print("Cleaned data:", cleaned_data)
print("Outlier indices:", outlier_indices)
```

```
Original data: 0       7.2500
1      71.2833
2       7.9250
3      53.1000
4       8.0500
        ...
886    13.0000
887    30.0000
888    23.4500
889    30.0000
890     7.7500
Name: Fare, Length: 889, dtype: float64
Cleaned data: 0       7.2500
2       7.9250
3      53.1000
4       8.0500
5       8.4583
        ...
886    13.0000
887    30.0000
888    23.4500
889    30.0000
890     7.7500
Name: Fare, Length: 775, dtype: float64
Outlier indices: (array([  1,  27,  31,  34,  52,  61,  71,  87, 101, 117, 119, 123, 138,
       150, 158, 179, 194, 200, 214, 217, 223, 229, 244, 255, 256, 257,
       261, 267, 268, 274, 289, 290, 296, 298, 304, 305, 306, 309, 310,
       317, 318, 323, 324, 331, 333, 335, 336, 340, 365, 368, 372, 374,
       376, 379, 384, 389, 392, 411, 434, 437, 444, 452, 483, 485, 495,
       497, 503, 504, 519, 526, 536, 539, 543, 549, 556, 557, 580, 584,
       586, 590, 608, 626, 640, 644, 654, 658, 659, 664, 678, 680, 688,
       697, 699, 707, 715, 729, 736, 740, 741, 744, 758, 762, 764, 778,
       788, 791, 801, 819, 833, 844, 847, 854, 861, 877]),)
```

In [34]:

```
# prompt: generate classification algortihm

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
# Logistic Regression
log_reg = LogisticRegression()
# K-Nearest Neighbor
knn = KNeighborsClassifier()
# Random Forest
random_forest = RandomForestClassifier()
# Naive Bayes
naive_bayes = GaussianNB()
# Support Vector Machine
svm = SVC()
# Decision Tree
decision_tree = DecisionTreeClassifier()
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df1.drop('Survived', axis=1), df1['Survived'], test_size=0.33, rand
# Train the models
models = [log_reg, knn, random_forest, naive_bayes, svm, decision_tree]
for model in models:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{model.__class__.__name__}: Accuracy - {accuracy:.2f}")
```

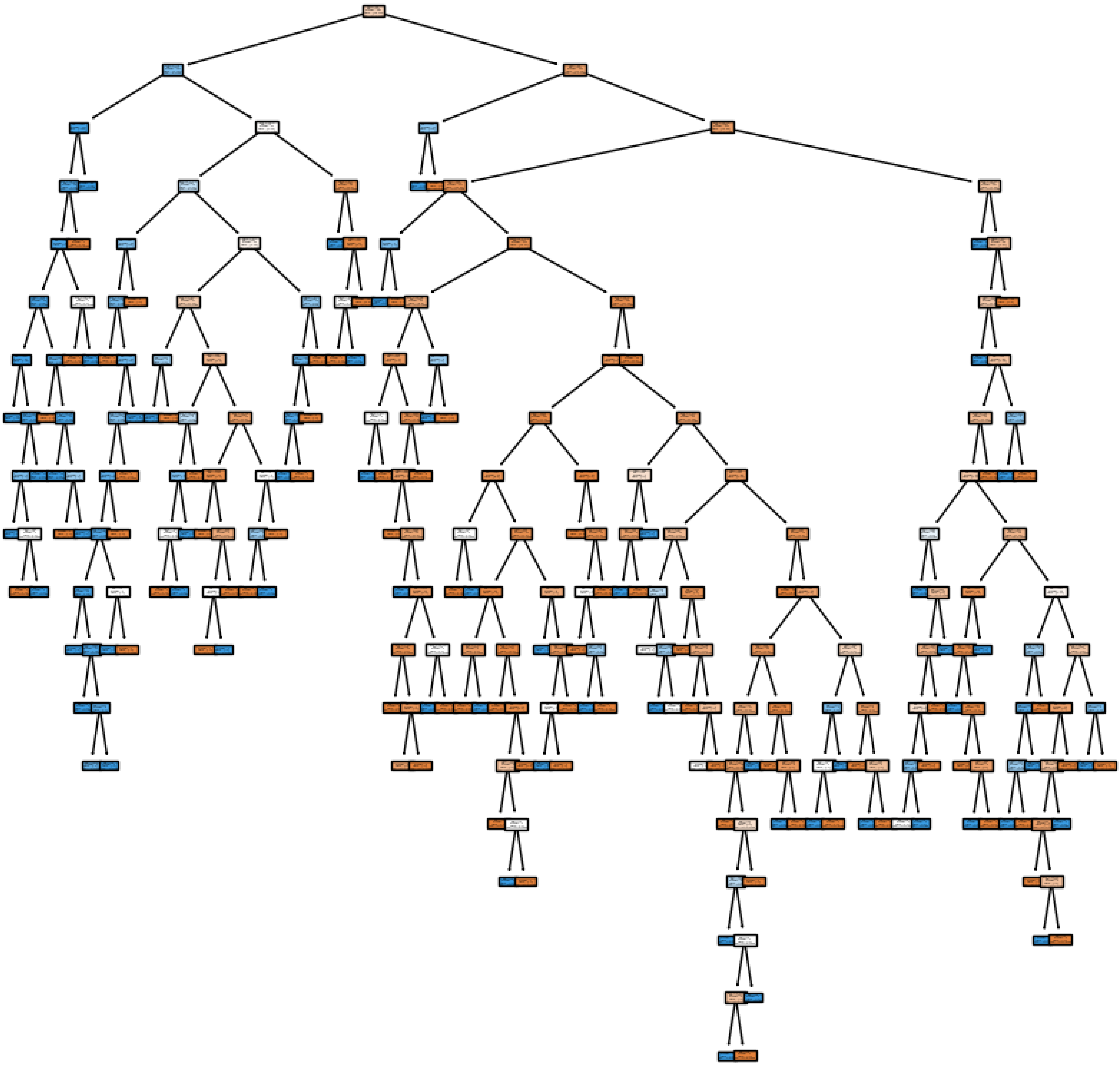
```
LogisticRegression: Accuracy - 0.81
KNeighborsClassifier: Accuracy - 0.70
RandomForestClassifier: Accuracy - 0.77
GaussianNB: Accuracy - 0.78
SVC: Accuracy - 0.67
DecisionTreeClassifier: Accuracy - 0.75
```

In [35]:

```
# prompt: decision tree code generate

decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
y_pred_dt = decision_tree.predict(X_test)
print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))

# Visualize the decision tree
import matplotlib.pyplot as plt
from sklearn import tree
```

```
plt.figure(figsize=(10, 10))
tree.plot_tree(decision_tree, feature_names=df1.drop('Survived', axis=1).columns, class_names=['Not Survived', 'Survive
plt.show()
```

Decision Tree Accuracy: 0.7482993197278912



In [37]:
```
# prompt: generate all algorithm classification report and accuracy

from sklearn.metrics import classification_report, accuracy_score

# Train the models
models = [log_reg, knn, random_forest, naive_bayes, svm, decision_tree]

for model in models:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{model.__class__.__name__}:")
    print(f"  Accuracy: {accuracy:.2f}")
    print(f"  Classification Report:")
    print(classification_report(y_test, y_pred))
    print()
```

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

           0       0.86      0.84      0.85       184
           1       0.74      0.76      0.75       110

    accuracy                           0.81       294
   macro avg       0.80      0.80      0.80       294
weighted avg       0.81      0.81      0.81       294


KNeighborsClassifier:
  Accuracy: 0.70
  Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.80      0.77       184
           1       0.61      0.53      0.57       110

    accuracy                           0.70       294
   macro avg       0.67      0.66      0.67       294
weighted avg       0.69      0.70      0.69       294


RandomForestClassifier:
  Accuracy: 0.78
  Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.83      0.82       184
           1       0.70      0.69      0.70       110

    accuracy                           0.78       294
   macro avg       0.76      0.76      0.76       294
weighted avg       0.77      0.78      0.78       294


GaussianNB:
  Accuracy: 0.78
  Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.80      0.82       184
           1       0.69      0.75      0.72       110

    accuracy                           0.78       294
   macro avg       0.77      0.78      0.77       294
weighted avg       0.79      0.78      0.78       294


SVC:
  Accuracy: 0.67
  Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.92      0.78       184
           1       0.66      0.26      0.38       110

    accuracy                           0.67       294
   macro avg       0.67      0.59      0.58       294
weighted avg       0.67      0.67      0.63       294


DecisionTreeClassifier:
  Accuracy: 0.74
  Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.78      0.79       184
           1       0.65      0.69      0.67       110

    accuracy                           0.74       294
   macro avg       0.73      0.73      0.73       294
weighted avg       0.75      0.74      0.75       294
```