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

In [2]:
```python
data= pd.read_csv(r"C:\Users\MEW\Downloads\train.csv")
```

In [3]:
```python
data.head()  #for first 5 rows
```

Out[3]:

	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

In [4]:
```python
data.info()  #for checking any missing value or not
```

```
<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
```

here we can see in age, cabin and embarked column have missing value.

In [5]:
```python
data.isna().sum()   # for sum of missing values
```

Out[5]:
```
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
```

Missing Value treatment

In [6]:
```python
 # Use mean to fill for numerical columns
data['Age'].fillna(data['Age'].mean(), inplace=True)
```

```
C:\Users\MEW\AppData\Local\Temp\ipykernel_1456\827470355.py:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  data['Age'].fillna(data['Age'].mean(), inplace=True)
```

In [7]:
```python
 # Use mean to fill for categorical columns
data['Cabin'].fillna(data['Cabin'].mode()[0], inplace=True)
```

```
C:\Users\MEW\AppData\Local\Temp\ipykernel_1456\101752579.py:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  data['Cabin'].fillna(data['Cabin'].mode()[0], inplace=True)
```

In [8]:
```python
 # Use mean to fill for categorical columns
data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)
```

```
C:\Users\MEW\AppData\Local\Temp\ipykernel_1456\4169062297.py:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)
```

In [9]:
```python
data.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          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  Cabin        891 non-null    object 
 11  Embarked     891 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [10]:
```python
data.isna().sum()
```

Out[10]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
dtype: int64
```

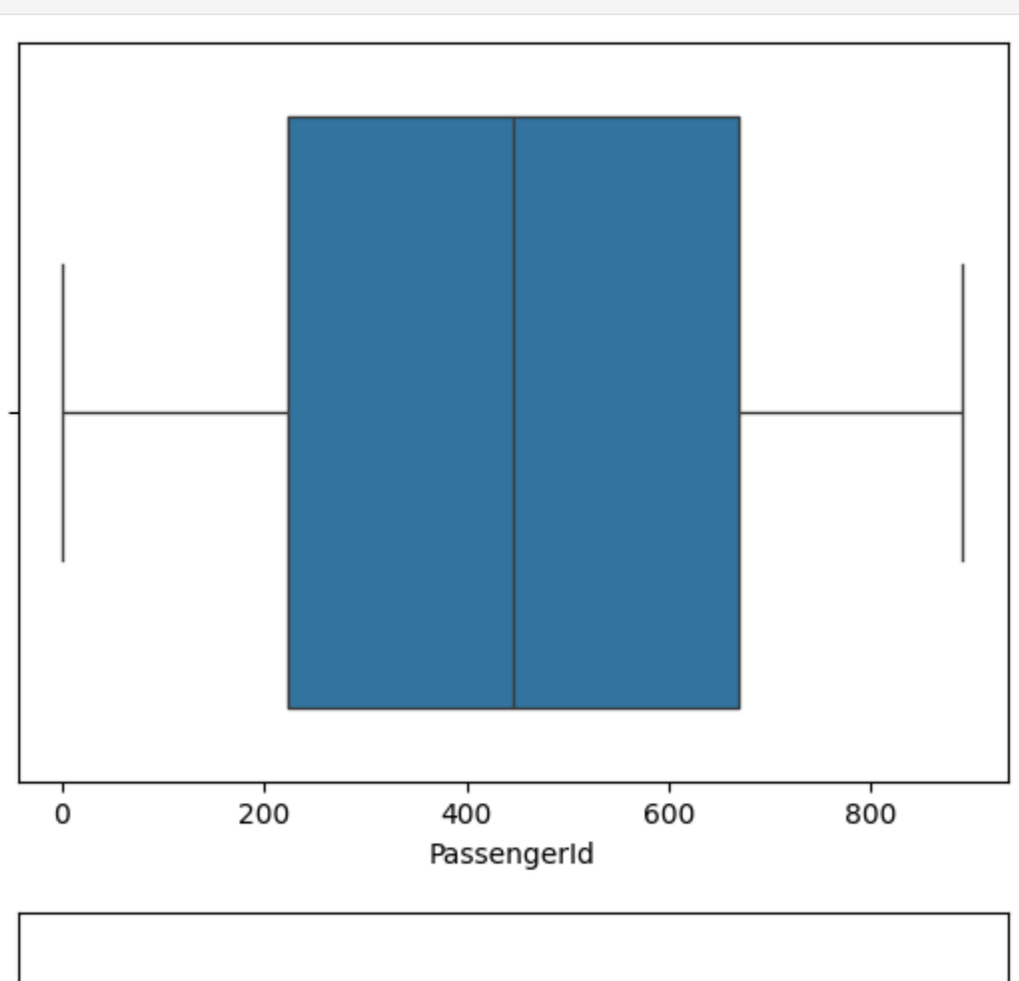
There is no missing values

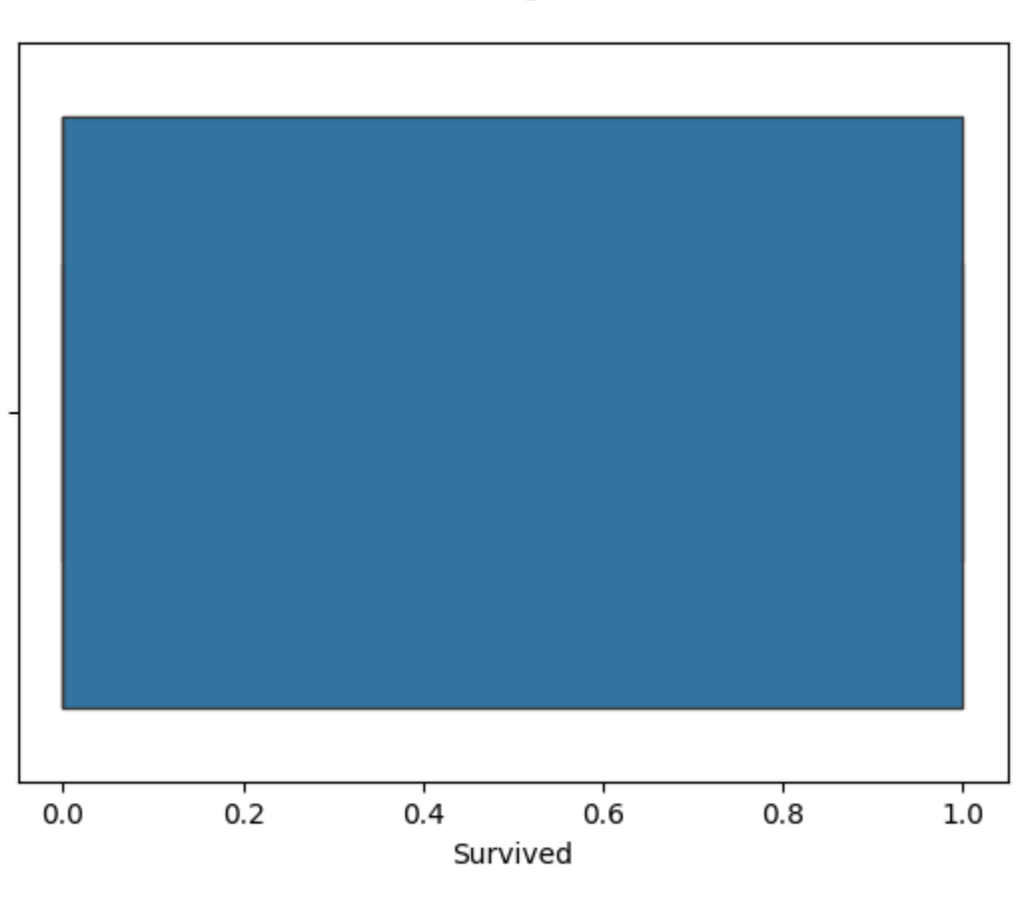
In [11]:
```python
data.describe()
```

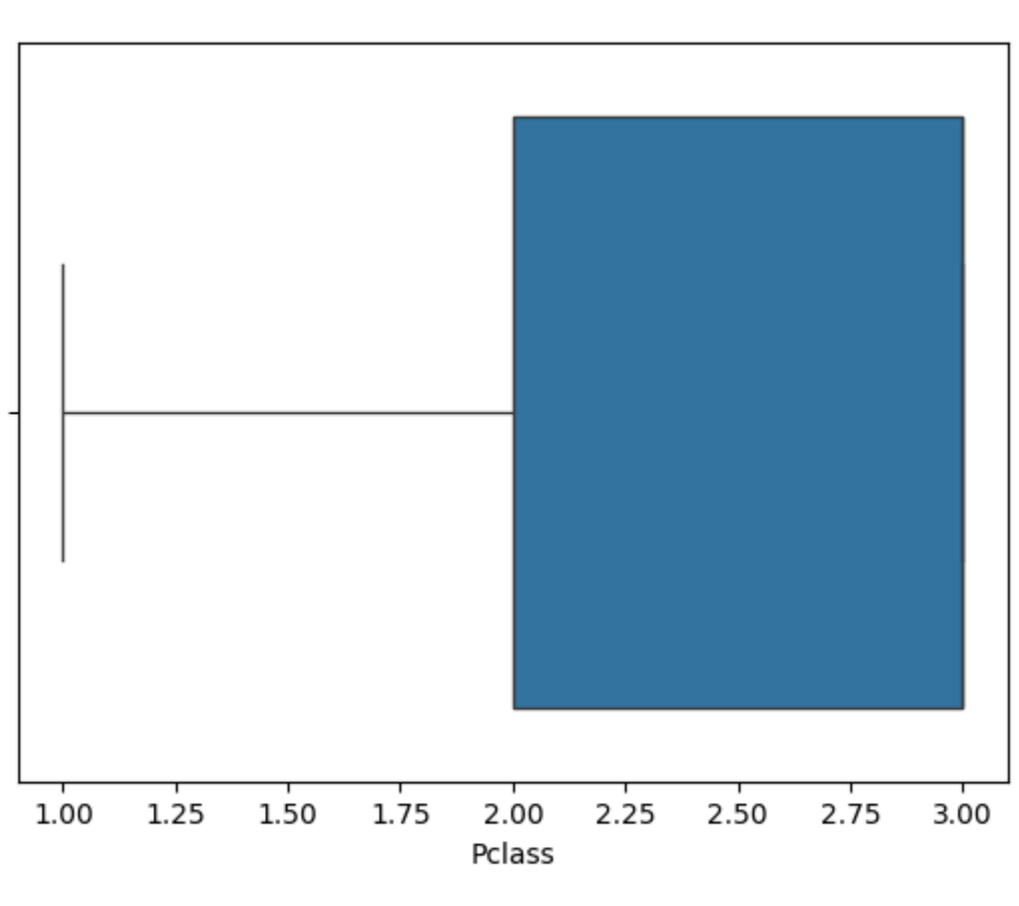
Out[11]:

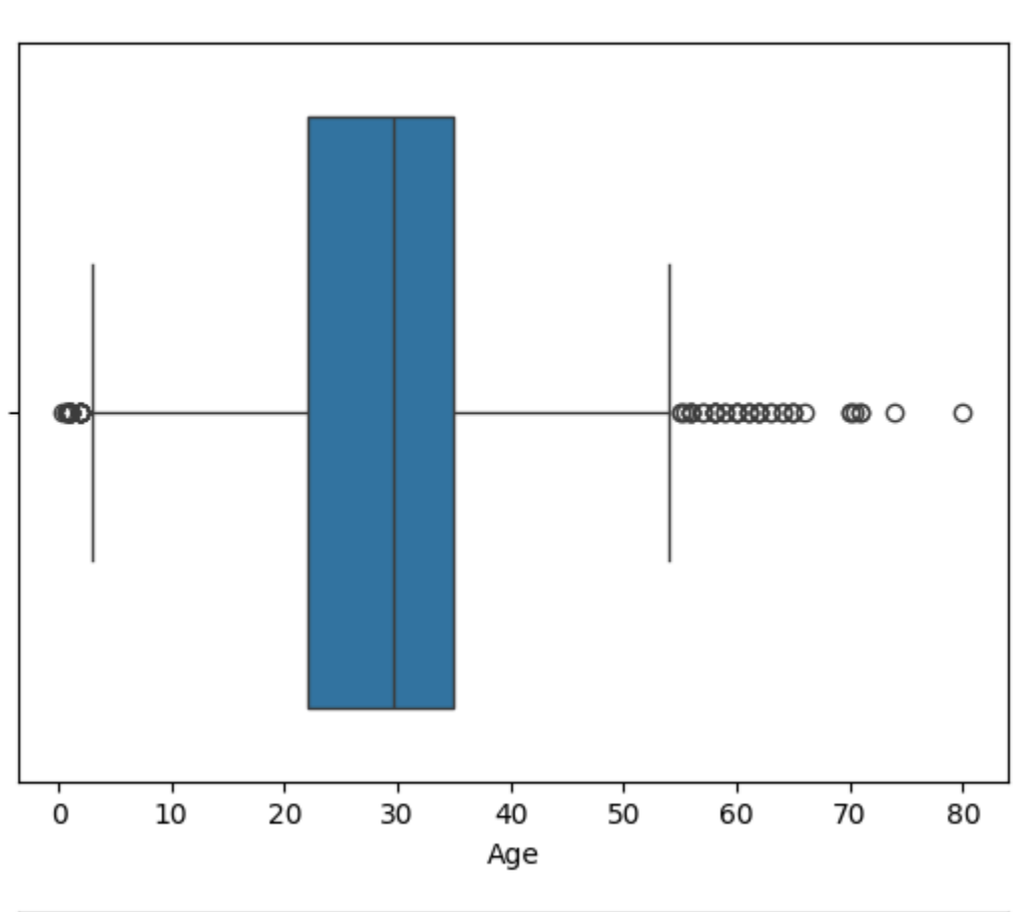
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	13.002015	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	22.000000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	29.699118	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	35.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

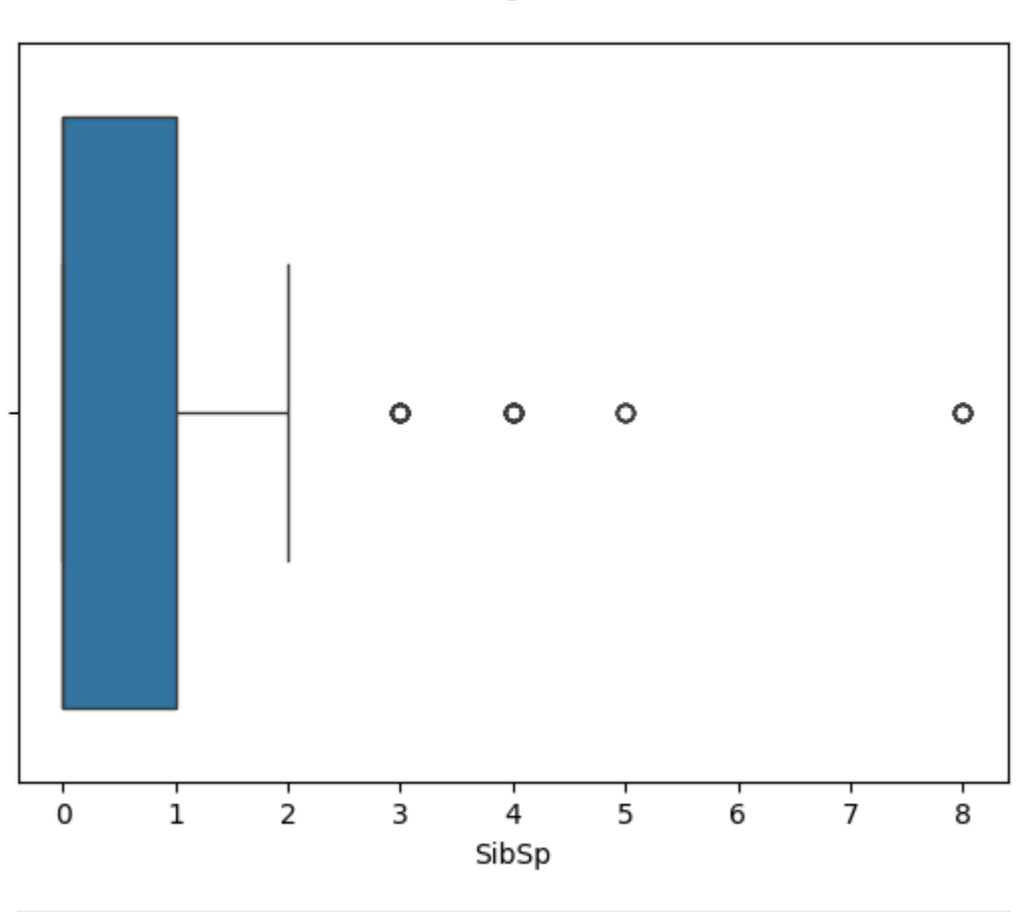
In [12]:
```python
# Create box plots for each numerical feature
for col in data.select_dtypes(include=np.number).columns:
    sns.boxplot(x=data[col])
    plt.show()
```

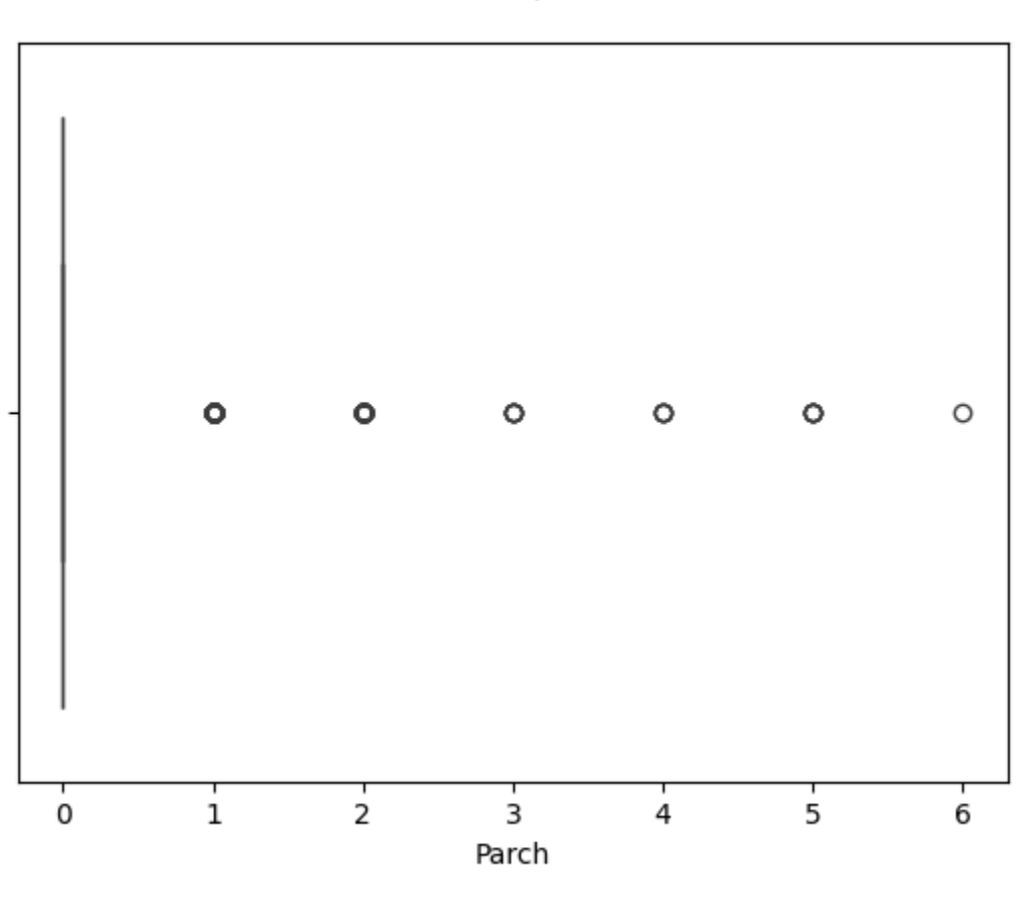


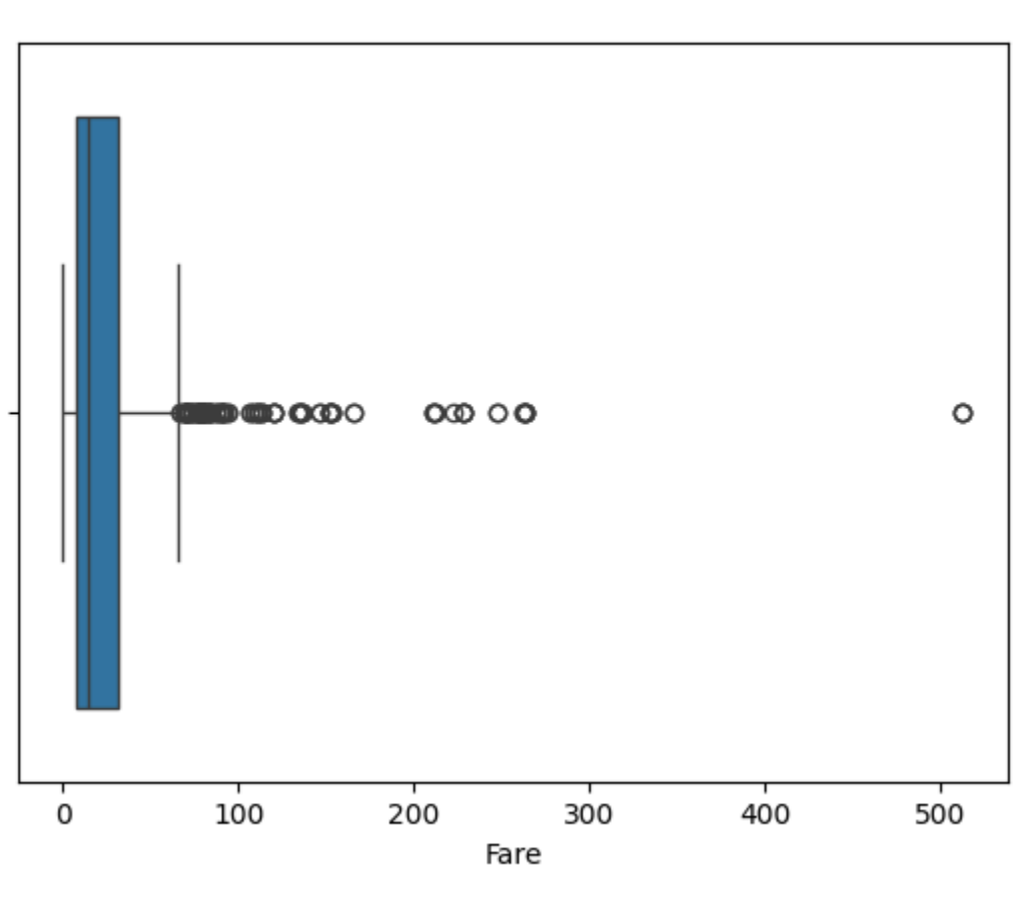












So we can see some outliers in Age,Sibsp,parch,and fare so we have to remove outliers to get a cleaner dataset and potentially improve the performance of our analysis or machine learning models.

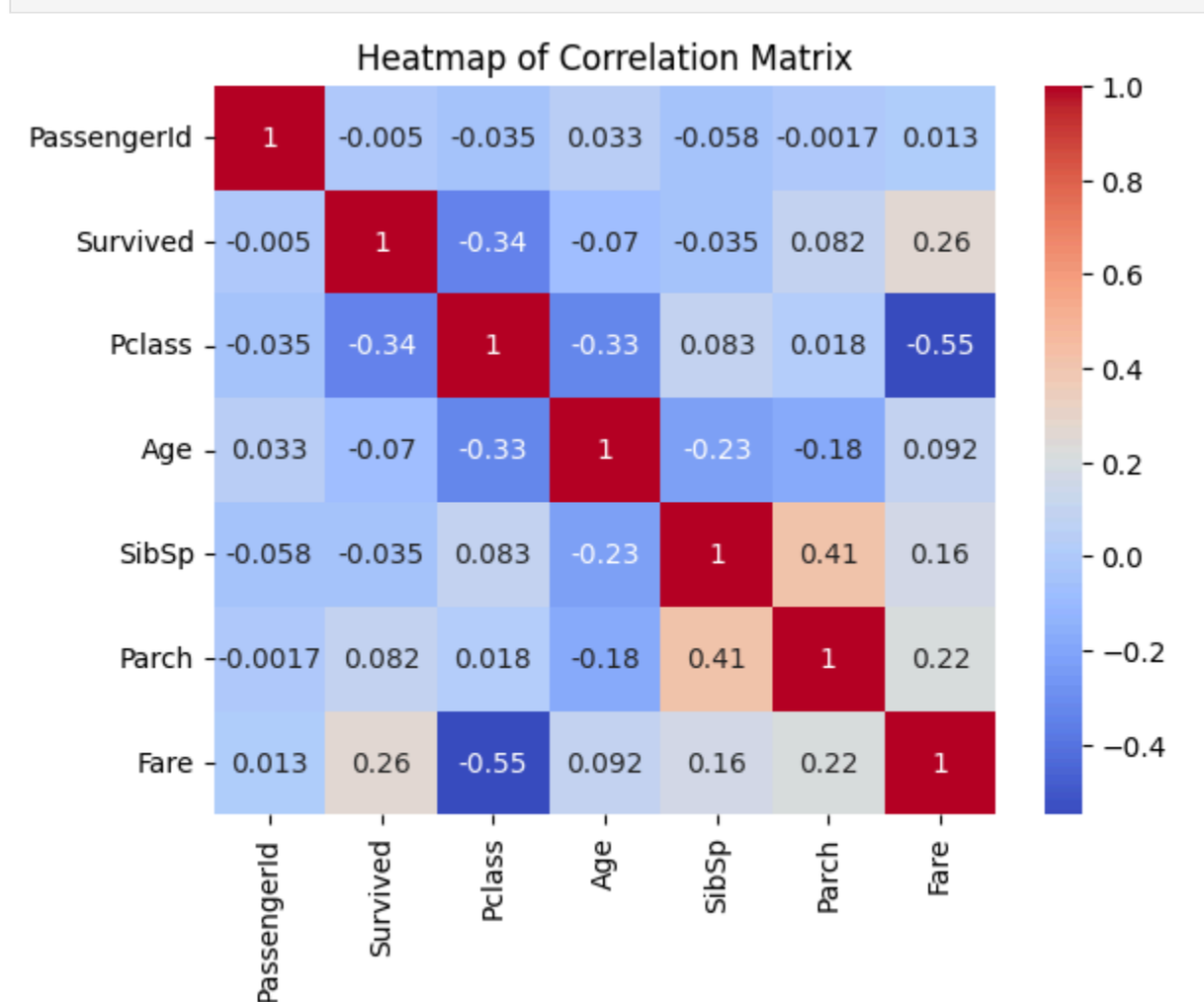
In [13]:
```python
print("value_counts()\n")
print("Survived:\n", data['Survived'].value_counts())
print("\nPclass:\n", data['Pclass'].value_counts())
print("\nSex:\n", data['Sex'].value_counts())
print("\nEmbarked:\n", data['Embarked'].value_counts())
print("\n")
```

```
value_counts()

Survived:
 Survived
0    549
1    342
Name: count, dtype: int64

Pclass:
 Pclass
3    491
1    216
2    184
Name: count, dtype: int64

Sex:
 Sex
male      577
female    314
Name: count, dtype: int64

Embarked:
 Embarked
S    646
C    168
Q     77
Name: count, dtype: int64
```

Survival Rate: Approximately 38.4% of passengers survived. **Passenger Class:** The most frequent passenger class was 3rd class. **Sex:** The majority of passengers were male. **Embarkation Port:** Most passengers embarked from Southampton (S).

In [15]:
```python
# Select only numerical columns for the pairplot
numerical_df = data.select_dtypes(include=['number'])
```

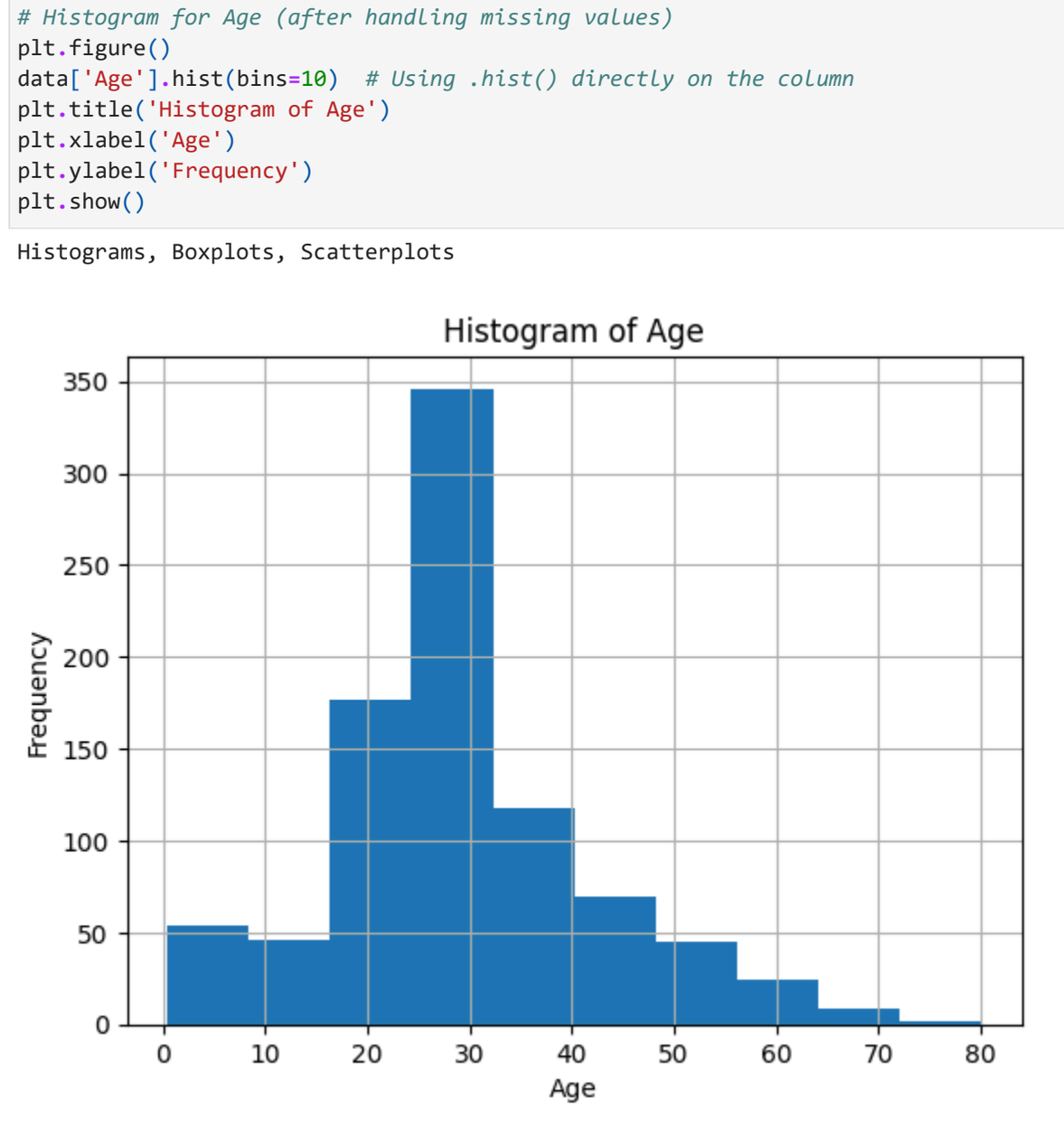
In [16]:
```python
# Create the heatmap.  Calculate correlation only for numerical columns.
correlation_matrix = numerical_df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Heatmap of Correlation Matrix')
plt.show()
```



you can see a strong negative correlation (-0.55) between "Pclass" (passenger class) and "Fare," suggesting that passengers in higher classes tended to pay higher fares. On the other hand, the correlation between "PassengerId" and "Survived" (-0.005) is very close to zero, indicating almost no linear relationship between a passenger's ID and their survival.And the highest correlation between different features is 0.41 between "SibSp" and "Parch".

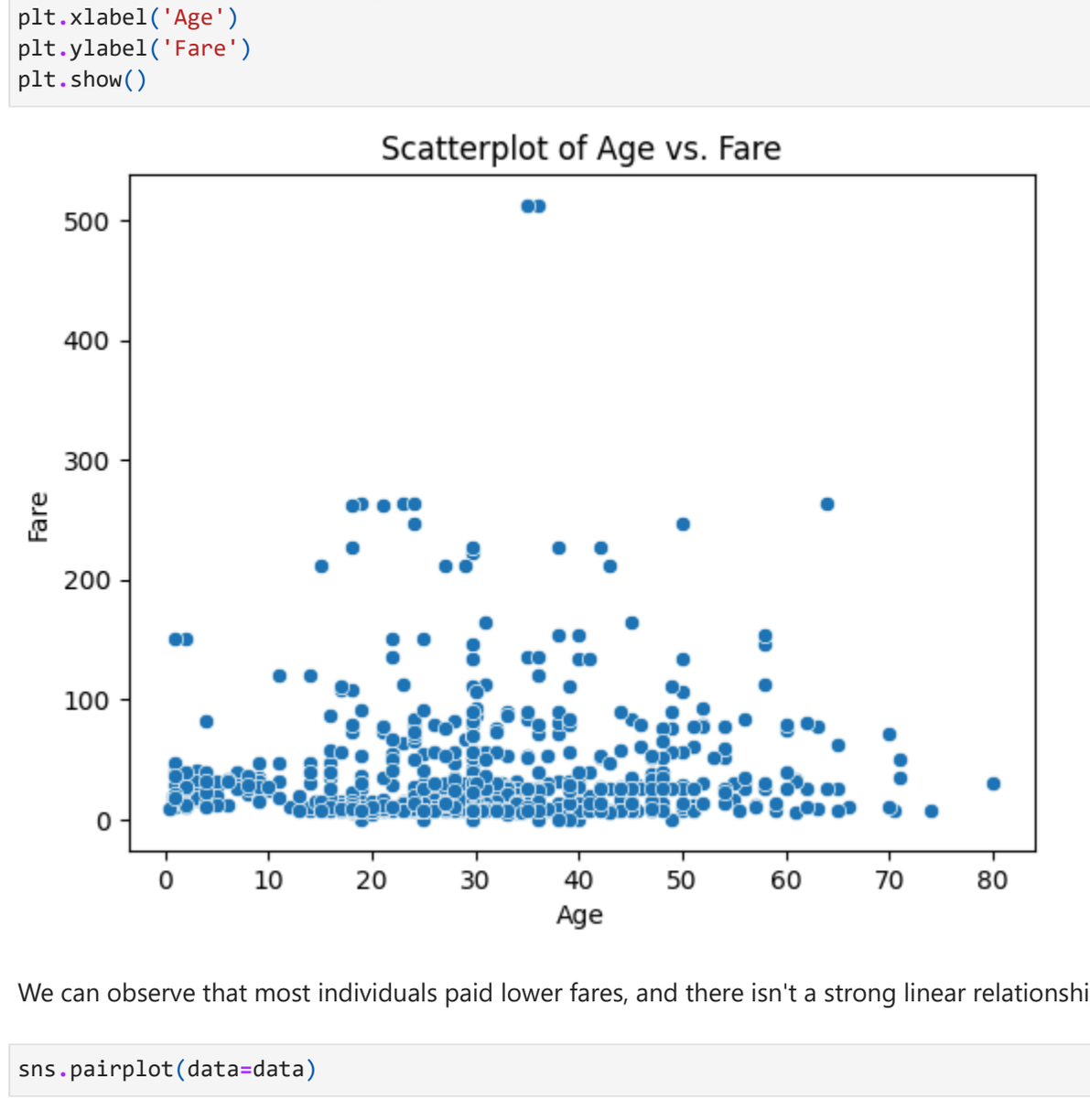
In [19]:
```python
print("Histograms, Boxplots, Scatterplots\n")
# Histogram for Age (after handling missing values)
plt.figure()
data['Age'].hist(bins=10)  # Using .hist() directly on the column
plt.title('Histogram of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```

```
Histograms, Boxplots, Scatterplots
```



The tallest bar is around the 20-30 age range, indicating that this is the most frequent age group in the dataset.

In [23]:
```python
# Scatterplot of Age vs. Fare
plt.figure()
sns.scatterplot(x='Age', y='Fare', data=data)
plt.title('Scatterplot of Age vs. Fare')
plt.xlabel('Age')
plt.ylabel('Fare')
plt.show()
```



We can observe that most individuals paid lower fares, and there isn't a strong linear relationship between age and fare. However, there are a few individuals, across different age groups, who paid significantly higher fares and outliers at the top.

In [31]:
```python
sns.pairplot(data=data)
```

Out[31]:
```
<seaborn.axisgrid.PairGrid at 0x17d76049a50>
```