Chapter*04* Practical Lab Session: Data Exploration and Treatment with Python

Learning Objectives

By completing this lab, students will be able to:

1. Perform initial data exploration and profiling
2. Handle missing values and detect outliers
3. Apply attribute transformations and normalization
4. Visualize distributions and relationships between features
5. Understand the importance of data preparation for machine learning

Required Libraries

In [4]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import missingno as msno
```

Part 1: Data Exploration

Step 1: Load the Dataset

In [5]:
```
import pandas as pd

df = pd.read_csv("Titanics.csv")
df.info()
df.describe(include='all')
```

```
<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
```

Out[5]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.000000	891	891.000000	204	889
unique	NaN	NaN	NaN	891	2	NaN	NaN	NaN	681	NaN	147	3
top	NaN	NaN	NaN	Braund, Mr. Owen Harris	male	NaN	NaN	NaN	347082	NaN	B96 B98	S
freq	NaN	NaN	NaN	1	577	NaN	NaN	NaN	7	NaN	4	644
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.381594	NaN	32.204208	NaN	NaN
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.806057	NaN	49.693429	NaN	NaN
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.000000	NaN	0.000000	NaN	NaN
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.000000	NaN	7.910400	NaN	NaN
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000	NaN	14.454200	NaN	NaN
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.000000	NaN	31.000000	NaN	NaN
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.000000	NaN	512.329200	NaN	NaN

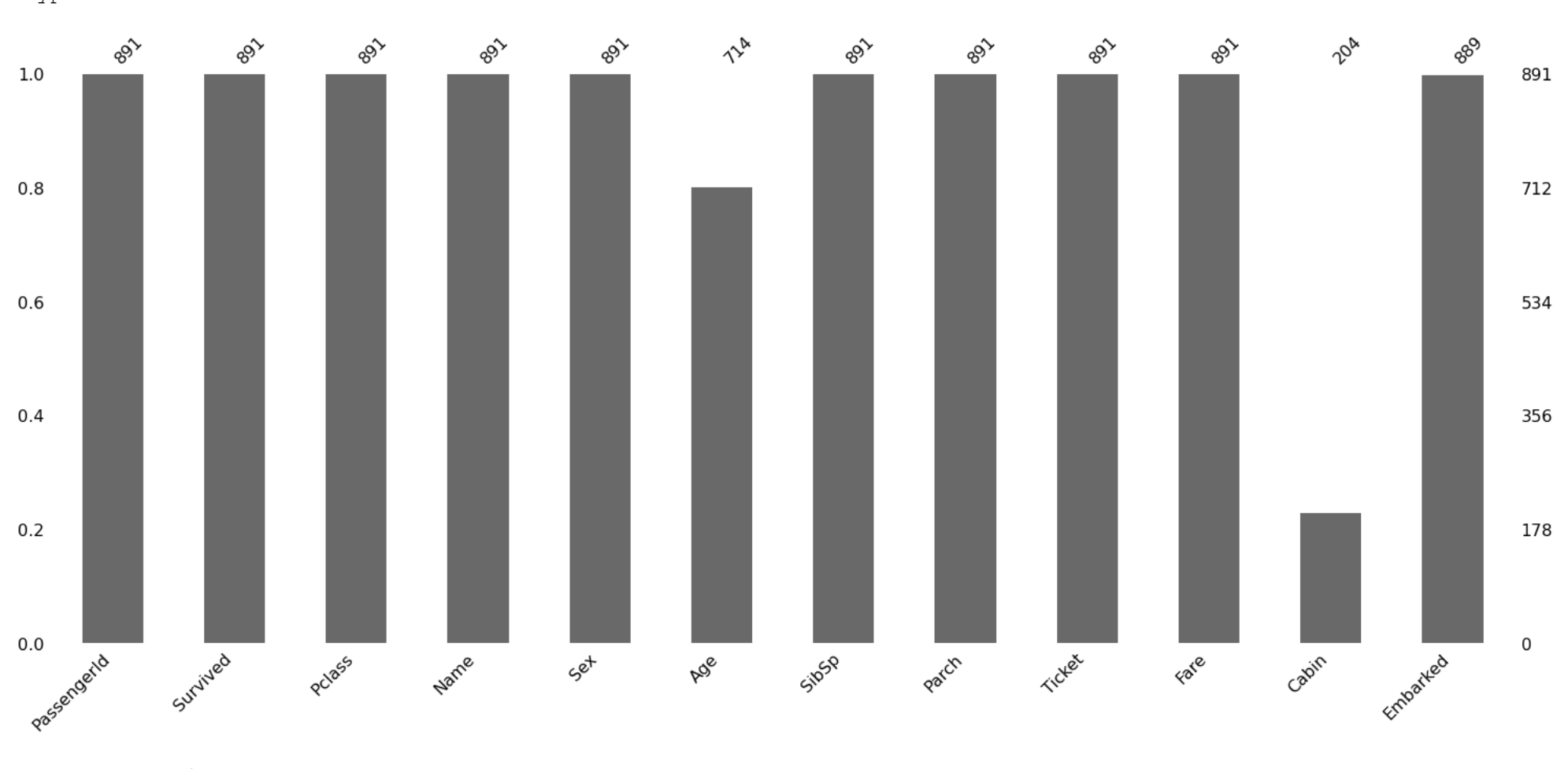
Step 2: Identify Target and Predictors

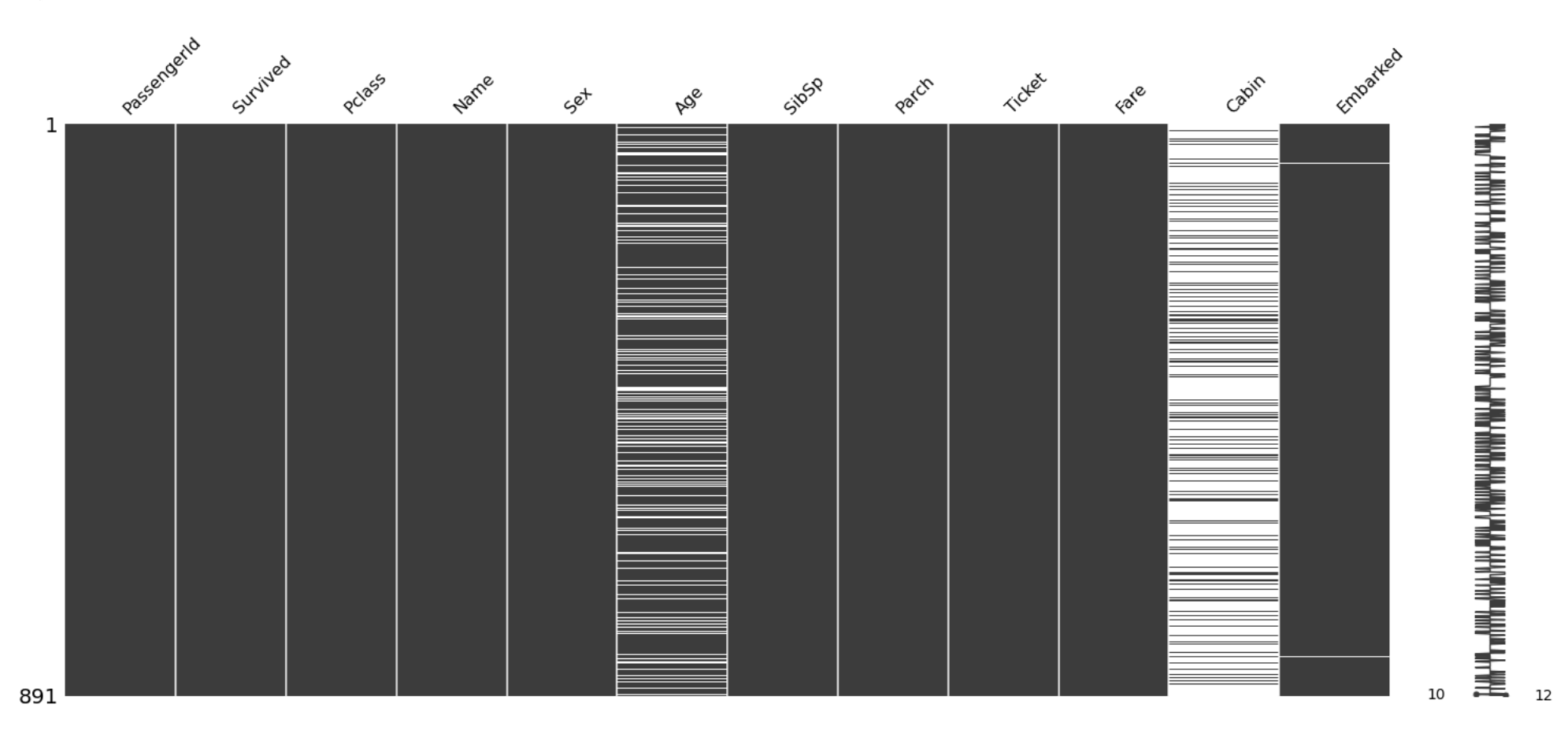
Target: Survived Predictors: Pclass, Sex, Age, Fare, Embarked, etc.

Step 3: Check for Missing Values

In [4]:
```
print(df.isnull().sum())        # Show the number of missing (NaN) values in each column
msno.bar(df)                    # Create a bar chart showing total and missing values per column
msno.matrix(df)                 # Visualize missing data patterns across the dataset as a matrix
plt.show()                      # Display the above visualizations
```

```
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
```





Exercise One

How many unique values are there in the Embarked column, and how many passengers belong to each category?

Part 2: Data Treatment

Step 1: Handle Missing Values

Option 1: Drop missing data

In [11]:
```
#Dropping Rows
df_drop_rows = df.dropna()          # Drop rows with missing data
print(df_drop_rows.isnull().sum()) # This will display the number of missing values per column in the new DataFrame.
# If all values are 0, then all rows with missing data were successfully removed."
```

```
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
```

In [9]:
```
#Dropping Columns
df_drop_columns = df.dropna(axis=1)  # Drop columns with missing data
print(df.shape[0], "→", df_drop_rows.shape[0]) # Shows how many rows were removed by comparing original and cleaned ro
w counts.
```

```
891 → 183
```

Option 2: Impute missing data

Sample Image

In [19]:
```
df['Age'].fillna(df['Age'].mean(), inplace=True) # Fills missing values in the 'Age' column with the mean age.
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True) # Fills missing values in the 'Embarked' column with the
most frequent (mode) value.
print(df.isnull().sum()) #Check if Missing Values Are Filled

# Show rows that had missing values in 'Age' or 'Embarked' originally
#print(df[df['Age'].isnull() | df['Embarked'].isnull()])
# After filling, this should return an empty DataFrame if imputation worked
```

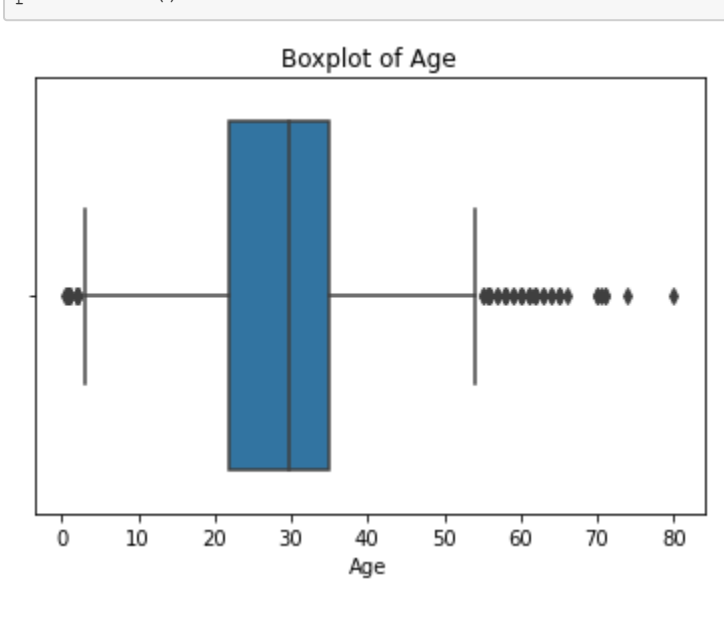
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         0
dtype: int64
```

Step 2: Outlier Detection

- ***Boxplot***

In [20]:
```
import seaborn as sns

sns.boxplot(x=df['Age'])
plt.title("Boxplot of Age")
plt.show()
```



- ***Z-Score Method***

In [21]:
```
mean_age = df['Age'].mean() # Calculate the mean (average) value of the 'Age' column.
std_age = df['Age'].std()  # Calculate the standard deviation of the 'Age' column.
z_scores = (df['Age'] - mean_age) / std_age # Compute the Z-score for each value in 'Age' — this tells how many standar
d deviations
# each age value is from the mean.

outliers = df[z_scores.abs() > 3] # Filter the dataset to include only rows where the absolute Z-score is greater than
3.
# These are considered statistical outliers.

print(outliers[['Age']]) # Display the 'Age' values identified as outliers.
```

```
      Age
96   71.0
116  70.5
493  71.0
630  80.0
672  70.0
745  70.0
851  74.0
```

Step 3: Attribute Transformation

- ***Normalization (Min-Max Scaling)***

In [23]:
```
from sklearn.preprocessing import MinMaxScaler # Import the MinMaxScaler from scikit-learn, which is used to scale feat
ures to a given range (default is 0 to 1).

scaler = MinMaxScaler() # Create an instance of the MinMaxScaler.
df[['Age', 'Fare']] = scaler.fit_transform(df[['Age', 'Fare']]) # Apply the scaler to the 'Age' and 'Fare' columns.
# This transforms the values so that they fall between 0 and 1.
# Useful for models that are sensitive to feature scales (e.g., k-NN, SVM, neural networks).
print(df[['Age', 'Fare']].head(10))
```

```
        Age      Fare
0  0.271174  0.014151
1  0.472229  0.139136
2  0.321438  0.015469
3  0.434531  0.103644
4  0.434531  0.015713
5  0.367921  0.016510
6  0.673285  0.101229
7  0.019854  0.041136
8  0.334004  0.021731
9  0.170646  0.058694
```

Compare Before and After Normalization

In [25]:
```
# Save a copy of the original columns before scaling
original_df = pd.read_csv("Titanics.csv")  # reloading to get original values

# Display side-by-side comparison of original vs normalized
comparison = pd.DataFrame({
    'Original_Age': original_df['Age'],
    'Normalized_Age': df['Age'],
    'Original_Fare': original_df['Fare'],
    'Normalized_Fare': df['Fare']
})

print(comparison.head(10))
```

```
   Original_Age  Normalized_Age  Original_Fare  Normalized_Fare
0          22.0        0.271174         7.2500         0.014151
1          38.0        0.472229        71.2833         0.139136
2          26.0        0.321438         7.9250         0.015469
3          35.0        0.434531        53.1000         0.103644
4          35.0        0.434531         8.0500         0.015713
5           NaN        0.367921         8.4583         0.016510
6          54.0        0.673285        51.8625         0.101229
7           2.0        0.019854        21.0750         0.041136
8          27.0        0.334004        11.1333         0.021731
9          14.0        0.170646        30.0708         0.058694
```

In [27]:
```
df['AgeGroup'] = pd.cut(df['Age'], bins=3, labels=['Young', 'Adult', 'Senior']) # Divide the 'Age' column into 3 equal-
width bins and assign labels to each group:
# 'Young', 'Adult', and 'Senior'.
# This transforms continuous numerical data into categorical groups for easier analysis.
# Show the new 'AgeGroup' column alongside the original 'Age' column
print(df[['Age', 'AgeGroup']].head(10))
```

```
        Age AgeGroup
0  0.271174    Young
1  0.472229    Adult
2  0.321438    Young
3  0.434531    Adult
4  0.434531    Adult
5  0.367921    Adult
6  0.673285   Senior
7  0.019854    Young
8  0.334004    Adult
9  0.170646    Young
```

In [28]:
```
# Count how many records fall into each age group
print(df['AgeGroup'].value_counts())
```

```
Adult     522
Young     319
Senior     50
Name: AgeGroup, dtype: int64
```

Exercise Two

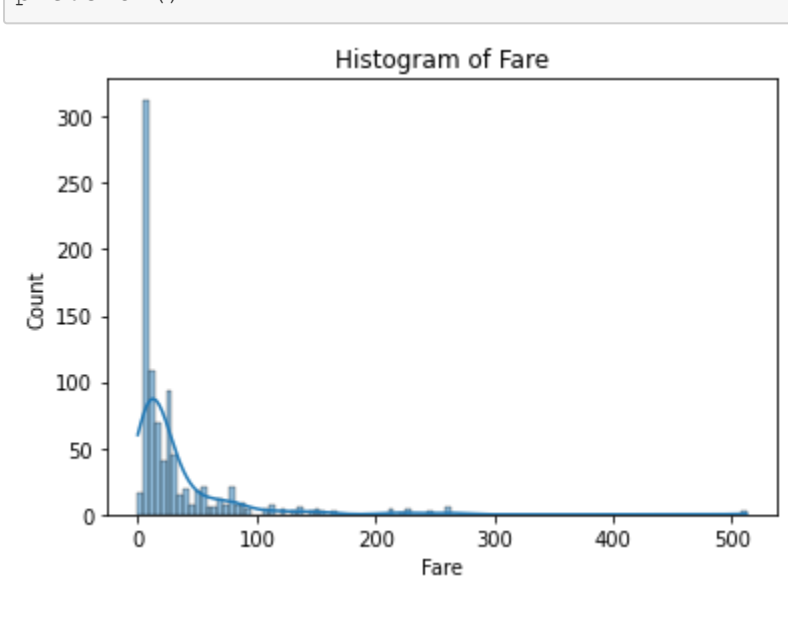
After filling missing values in the Age column, calculate the average age of passengers in each Pclass.

Part 3: Data Visualization (Analysis)

Step 1: Histogram and KDE

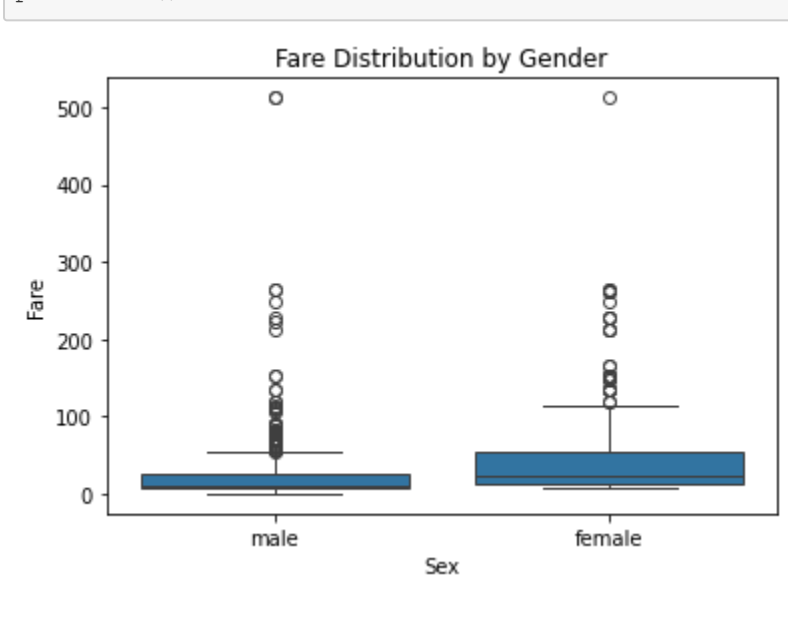
In [6]:
```
!pip install --upgrade seaborn --user

sns.histplot(df['Fare'], kde=True)
plt.title("Histogram of Fare")
plt.show()
```



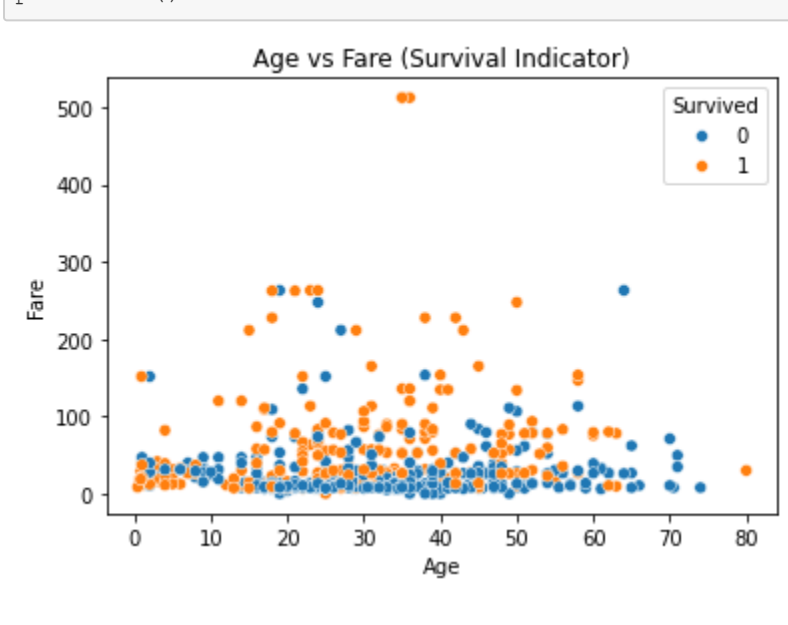
Step 2: Boxplots by Category

In [7]:
```
sns.boxplot(x='Sex', y='Fare', data=df)
plt.title("Fare Distribution by Gender")
plt.show()
```



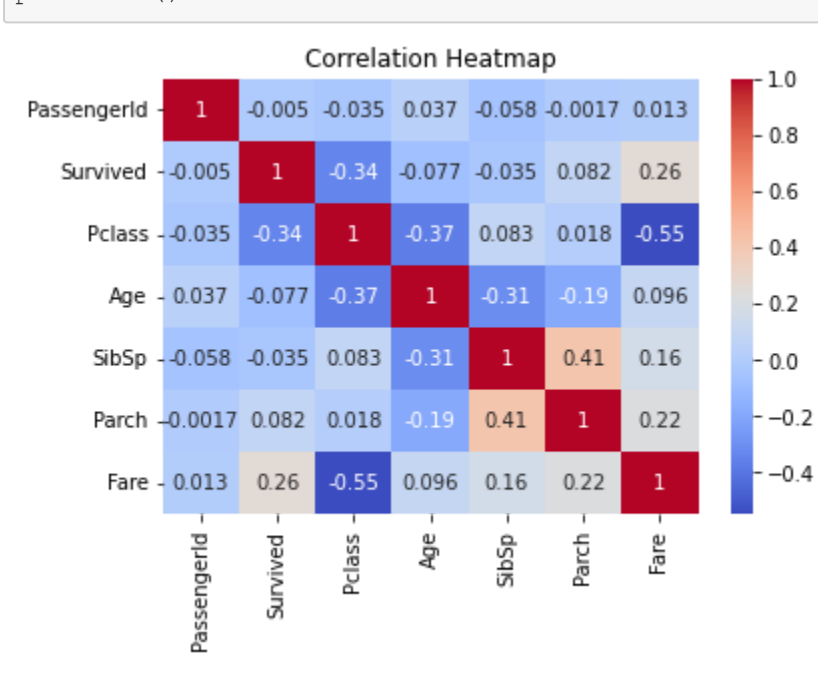
Step 3: Scatter Plot for Relationships

In [8]:
```
sns.scatterplot(x='Age', y='Fare', hue='Survived', data=df)
plt.title("Age vs Fare (Survival Indicator)")
plt.show()
```



Step 4: Correlation Heatmap (Bonus)

In [9]:
```
corr = df.corr(numeric_only=True)
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```



Exercise Three

Create a bar chart showing the number of survivors (Survived) for each combination of Sex and Pclass.

Part 4: Descriptive Statistics

Step 1: Univariate Analysis – Numerical

In [10]:
```
# Calculate the mean (average) of the 'Age' column
print("Mean Age:", df['Age'].mean())

# Calculate the median (middle value) of the 'Age' column
print("Median Age:", df['Age'].median())

# Calculate the mode (most frequent value) of the 'Age' column
print("Mode of Age:", df['Age'].mode()[0])
```

```
Mean Age: 29.69911764705882
Median Age: 28.0
Mode of Age: 24.0
```

In [11]:
```
# Calculate the standard deviation - shows how spread out the values are
print("Standard Deviation of Age:", df['Age'].std())

# Calculate the variance - average squared deviation from the mean
print("Variance of Age:", df['Age'].var())

# Calculate the range - difference between max and min values
age_range = df['Age'].max() - df['Age'].min()
print("Range of Age:", age_range)

# Calculate the Interquartile Range (IQR) - spread of the middle 50% of values
Q1 = df['Age'].quantile(0.25)   # First quartile (25th percentile)
Q3 = df['Age'].quantile(0.75)   # Third quartile (75th percentile)
IQR = Q3 - Q1                   # IQR = Q3 - Q1
print("Interquartile Range (IQR) of Age:", IQR)

# Calculate skewness - shows if the distribution is skewed left/right
print("Skewness of Age:", df['Age'].skew())
```

```
Standard Deviation of Age: 14.526497332334044
Variance of Age: 211.0191247463081
Range of Age: 79.58
Interquartile Range (IQR) of Age: 17.875
Skewness of Age: 0.38910778230082704
```

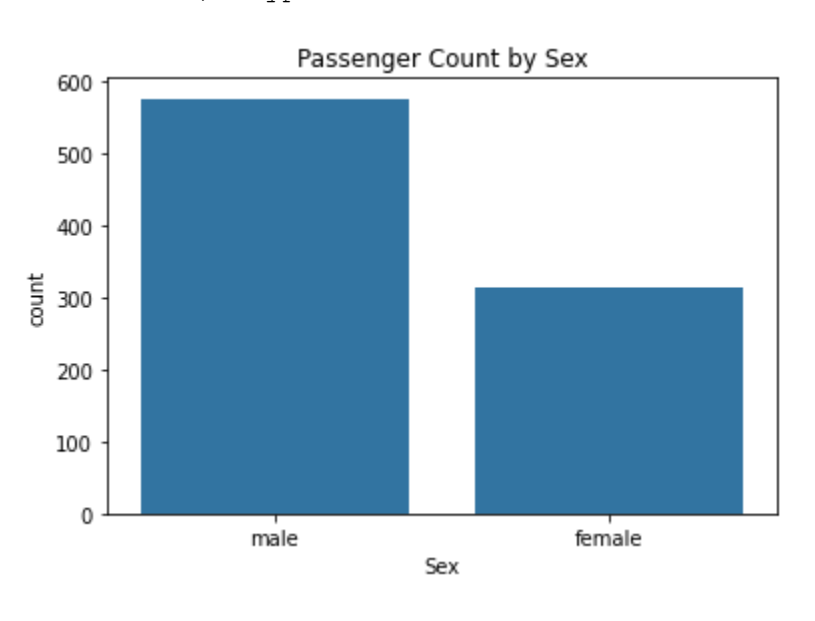
Step 2: Univariate Analysis – Categorical

In [12]:
```
# Count the number of passengers by sex
print(df['Sex'].value_counts())

# Visualize passenger counts by sex using a bar chart
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(x='Sex', data=df)      # Bar chart of count per category in 'Sex'
plt.title('Passenger Count by Sex') # Title for the plot
plt.show()                          # Display the chart
```

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

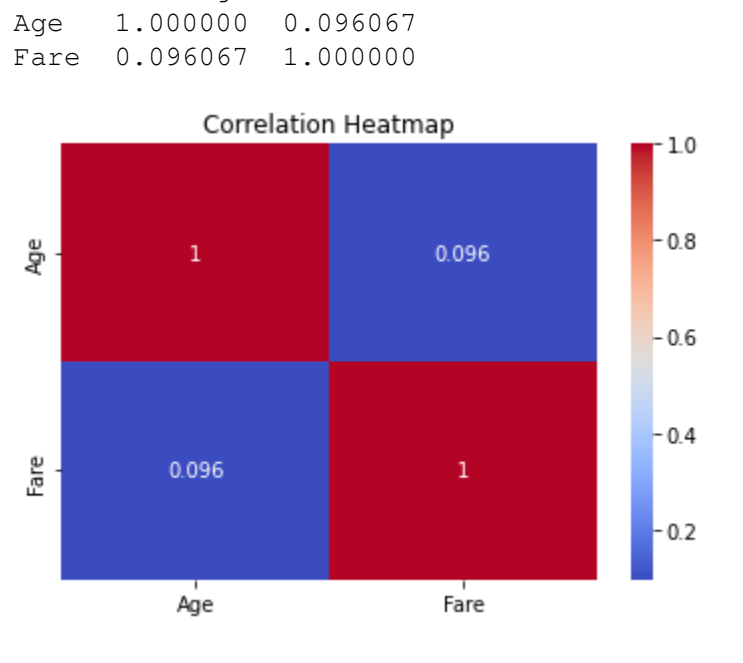


Step 3: Bivariate Analysis – Numerical vs Numerical

In [13]:
```
# Calculate correlation between 'Age' and 'Fare'
correlation = df[['Age', 'Fare']].corr()  # Compute correlation matrix
print(correlation)

# Display the correlation matrix as a heatmap
sns.heatmap(correlation, annot=True, cmap='coolwarm')  # Annotate values with color
plt.title("Correlation Heatmap")  # Add plot title
plt.show()
```

```
           Age      Fare
Age   1.000000  0.096067
Fare  0.096067  1.000000
```

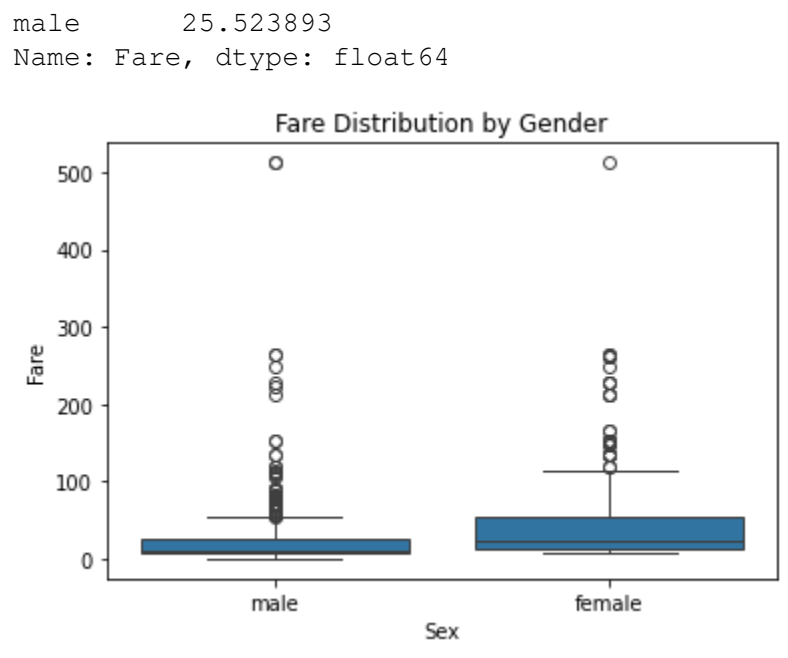


Step 4: Bivariate Analysis – Categorical vs Numerical

In [14]:
```
# Calculate average Fare for each gender
print(df.groupby('Sex')['Fare'].mean())

# Visualize the distribution of Fare across different genders
sns.boxplot(x='Sex', y='Fare', data=df)  # Boxplot shows distribution and outliers
plt.title("Fare Distribution by Gender")
plt.show()
```

```
Sex
female    44.479818
male      25.523893
Name: Fare, dtype: float64
```

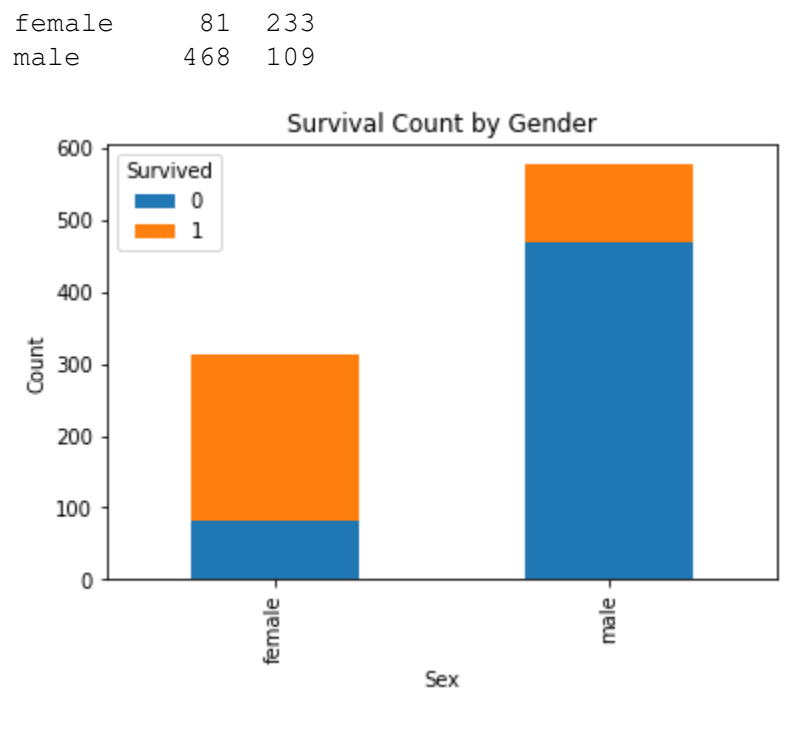


Step 5: Bivariate Analysis – Categorical vs Categorical

In [16]:
```
# Create a cross-tabulation table between 'Sex' and 'Survived'
ct = pd.crosstab(df['Sex'], df['Survived'])  # Shows count of survivors per gender
print(ct)

# Create a stacked bar chart to visualize survival by gender
ct.plot(kind='bar', stacked=True)        # Stacked bar to compare survival distribution
plt.title("Survival Count by Gender")    # Chart title
plt.xlabel("Sex")                        # X-axis label
plt.ylabel("Count")                      # Y-axis label
plt.show()
```

```
Survived    0    1
Sex
female     81  233
male      468  109
```



Exercise 4

Create a bar chart showing the number of survivors (Survived) for each combination of Sex and Pclass.

End