1 - DEFINE THE PROBLEM

This notebook uses the titanic3 dataset to explore AI concepts through machine learning.
The goal is to predict whether a passenger survived the Titanic disaster.

2 - IMPORT REQUIRED LIBRARIES

2.1 - Base Libraries

In [1]:
```
# Import base libraries. We will use pandas for data handling (DataFrames),
# numpy for numerical operations, and matplotlib/seaborn for data visualization.
```

2.2 - ML/DL Libraries

In [2]:
```
# Import specific functions from the scikit-learn library that are required
# for preprocessing, model training, and evaluation.
```

3 - LOAD THE DATA

In [3]:
```
# Read the housing.csv file located in the datasets folder.
# Store it in a DataFrame called 'data'.

data =
```

4 - EDA (Exploratory Data Analysis)

In [4]:
```
# Display the first five rows of the dataset to understand the structure of the data
# and preview the features available for analysis and modeling.
```

Out[4]:

	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 [5]:
```
# Analyze Data Types
# Display the data types of each column in the dataset.
# This helps us identify which features are numerical and which are categorical.
```

Out[5]:
```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

In [6]:
```
# Display general information about the dataset
# Includes number of entries, non-null counts, and data types for each column
```

```
<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
None
```

In [7]:
```
# Show summary statistics for numerical columns
# Includes count, mean, std deviation, min, 25%, median (50%), 75%, and max
```

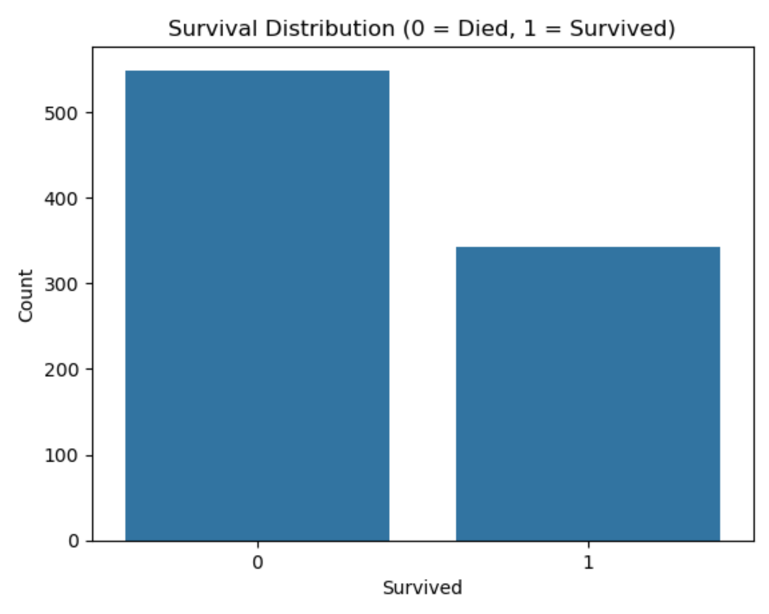
```
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000   
mean    446.000000    0.383838    2.308642   29.699113    0.523008   
std     257.353842    0.486592    0.836071   14.526507    1.102743   
min       1.000000    0.000000    1.000000    0.416700    0.000000   
25%     223.500000    0.000000    2.000000   20.125000    0.000000   
50%     446.000000    0.000000    3.000000   28.000000    0.000000   
75%     668.500000    1.000000    3.000000   38.000000    1.000000   
max     891.000000    1.000000    3.000000   80.000000    8.000000   

            Parch        Fare  
count  891.000000  891.000000  
mean     0.381594   32.204208  
std      0.806057   49.693429  
min      0.000000    0.000000  
25%      0.000000    7.910400  
50%      0.000000   14.454200  
75%      0.000000   31.000000  
max      6.000000  512.329200  
```

In [8]:
```
# Check how many missing values exist in each column
# Useful for planning imputations or data cleaning strategies
```

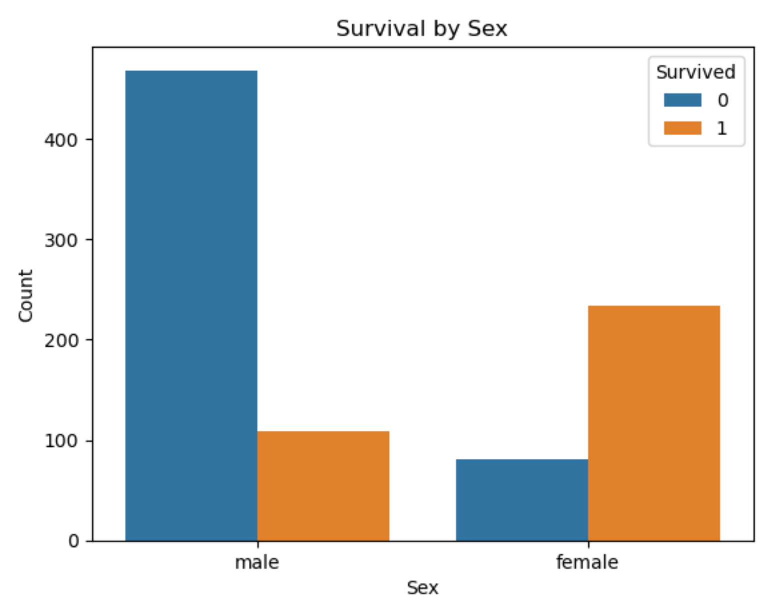
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

5 - VISUALIZE THE DATA

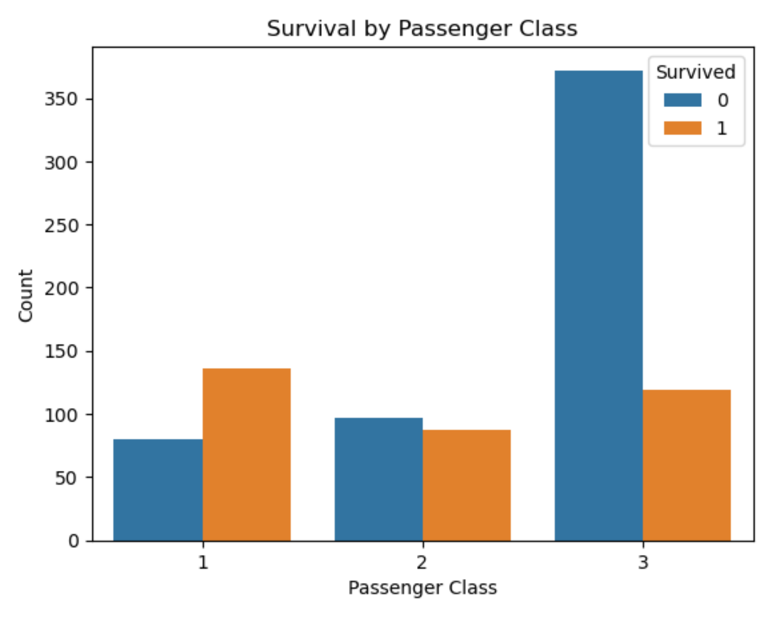
In [9]:
```
# Survival Distribution
sns.countplot
plt.title
plt.xlabel
plt.ylabel
```



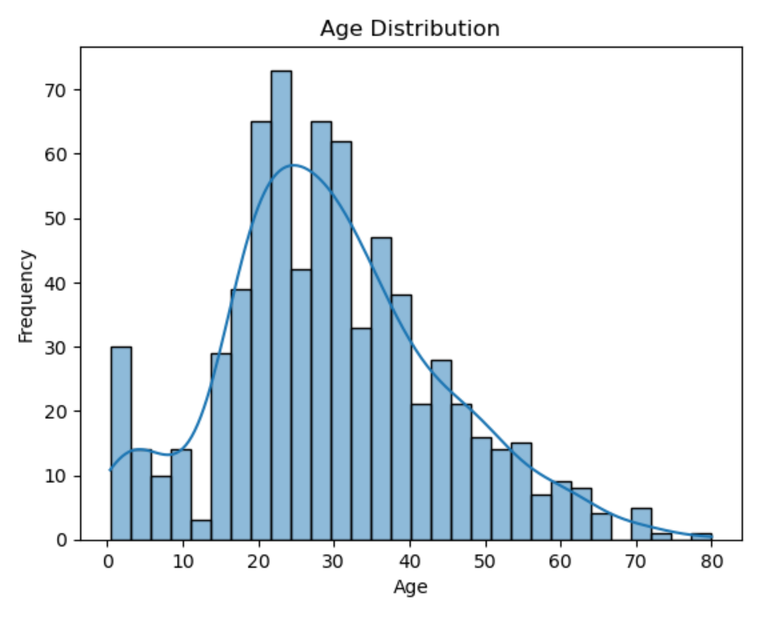
In [10]:
```
# Survival by Sex
sns.countplot
plt.title
plt.xlabel
plt.ylabel
```



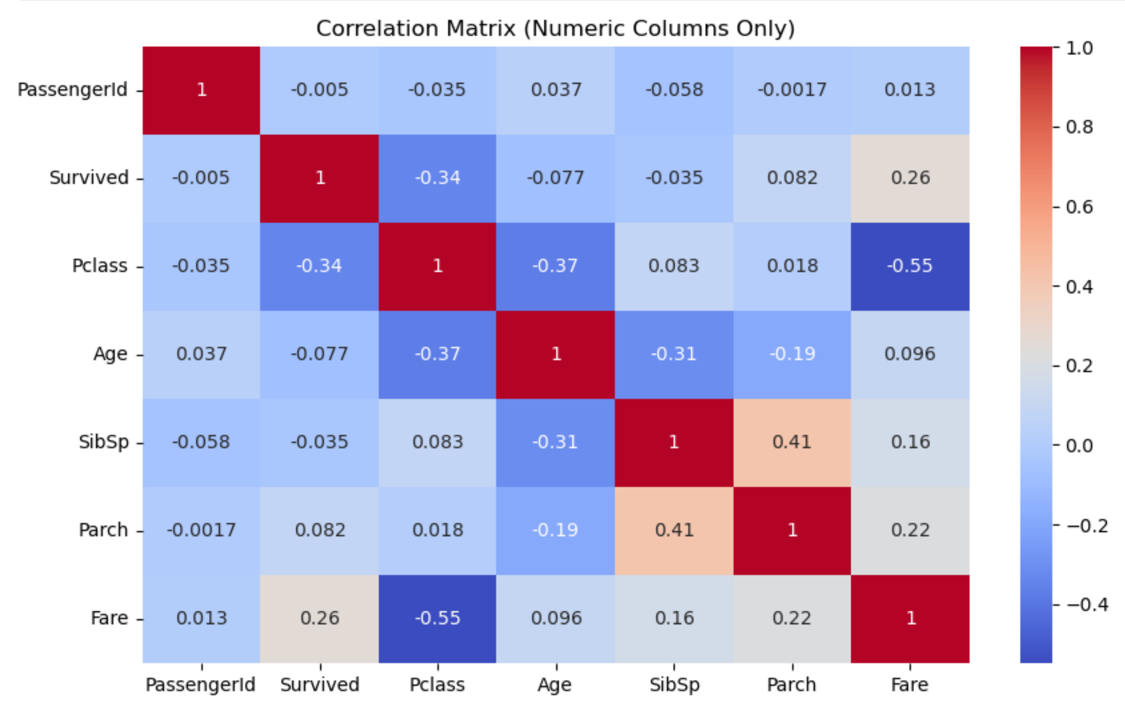
In [11]:
```
# Survival by Passenger Class
sns.countplot(x='Pclass', hue='Survived', data=data)
sns.countplot
plt.title
plt.xlabel
plt.ylabel
```



In [12]:
```
# Age distribution
sns.histplot
plt.title
plt.xlabel
plt.ylabel
```



In [13]:
```
# Correlation Heatmap (numeric only)

# Step 1: Create a figure with an appropriate size (e.g., width=10, height=6).
plt.figure

# Step 2: Generate a correlation matrix using the .corr() method.
# Make sure to include only numeric columns using numeric_only=True.
# Step 3: Use seaborn's heatmap to visualize the correlation matrix.
# Enable annotations (annot=True) and apply a diverging color map like 'coolwarm'.
sns.heatmap(data.corr

# Step 4: Add a descriptive title to the plot.
plt.title

# Step 5: Display the plot.
```



6 - PREPROCESS THE DATA

In [14]:
```
print
```

```
['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked']
```

In [15]:
```
# Step 1: Create X and y
# Select the appropriate features for X and define the target variable y.
# Make sure to exclude columns like 'PassengerId', 'Name', 'Ticket', and 'Cabin'.

X =
y =
```

In [16]:
```
# Step 2: Manually define the list of numerical and categorical features.
# For example: numerical → ['Age', 'Fare']; categorical → ['Pclass', 'Sex', 'Embarked']
numeric_features =
categorical_features =
```

In [16]:
```
# Step 3: Create the pipeline for numerical features:
# - Use SimpleImputer to fill missing values with the median.
# - Apply StandardScaler to normalize the features.
numeric_transformer =
```

In [16]:
```
# Step 4: Create the pipeline for categorical features:
# - Use SimpleImputer with 'most_frequent' strategy for missing values.
# - Apply OneHotEncoder (handle_unknown='ignore', and sparse_output=False).
categorical_transformer =
```

In [16]:
```
# Step 5: Combine the numeric and categorical pipelines using ColumnTransformer.
# Assign each transformer to the corresponding column list.
preprocessor =
```

In [16]:
```
# Step 6: Fit and transform the full dataset X using the combined preprocessor.
# Store the result in a variable named X_preprocessed.
X_preprocessed = preprocessor.fit_transform(X)
```

7 - SPLIT THE DATA

In [ ]:
```
# Step 1: Use train_test_split to divide the dataset into training and test sets.
# Use the processed features (X_preprocessed) and the target (y).
# Set the test size to 20% and use random_state=42 for reproducibility.
# Store the outputs as X_train, X_test, y_train, y_test.

X_train, X_test, y_train, y_test =
```

In [17]:
```
# Step 2: Print the shape of X_train and X_test to verify the split.
print
```

```
Train shape: (712, 10), Test shape: (179, 10)
```

In [ ]:
```
# Step 3: Calculate the total number of samples, as well as the number of training and test samples.
total =
train_rows, test_rows =
```

In [ ]:
```
# Step 4: Create percentage labels showing both the count and percentage for training and test sets.
# Example: "Train (712 - 79.9%)"
labels =
```

In [ ]:
```
# Step 5: Define colors and pie chart settings.
plt.figure
plt.pie
```

In [18]:
```
# Step 6: Create the pie chart showing the distribution of training and test samples.
# Customize the chart with title, colors, edge styling, and ensure it is displayed as a circle.
plt.title
plt.axis
```