Data and Artificial Intelligence
Cyber Shujaa Program

Week 3 Assignment
Titanic Exploratory Data Analysis

Student Name: Shirleen Nanetia Simon

Student ID: CS-DA01-25077

Introduction

Exploratory Data Analysis (EDA) is the process of analyzing datasets to summarize their main characteristics, often using visual methods. It helps in understanding the structure of the data, detecting anomalies, identifying patterns, and checking assumptions before applying any machine learning models.

The steps involved in EDA are:

1. Initial exploration of the data
2. Handling missing values and outliers
3. Univariate analysis
4. Bivariate analysis
5. Multivariate analysis
6. Analyze and visualize the Survived target variable.

This project presents a comprehensive Exploratory Data Analysis (EDA) on the Titanic dataset. The aim is to understand how different features such as age, sex, class are related to the likelihood of survival.

The goals of this assignment were:

- Understand the structure and features of the Titanic dataset.
- Clean and preprocess the data for analysis.
- Perform univariate, bivariate, and multivariate analyses.
- Visualize key patterns related to survival.
- Derive actionable insights that could support predictive modeling.

Tasks Completed

Importing the essential Python libraries and loading the Dataset.

- import pandas as pd: Imports Pandas, used for data manipulation.
- import numpy as np: Imports NumPy, useful for numerical operations.
- import matplotlib.pyplot as plt and import seaborn as sns: Import libraries for data visualization.
- sns.set() and plt.style.use(): Set the style for plots (clean and muted).
- df = pd.read_csv(...): Loads the Titanic training dataset into a DataFrame called df.
- df.head(): Displays the first 5 rows of the dataset.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Setup
sns.set(style="whitegrid")
plt.style.use("seaborn-v0_8-muted")

# Load dataset
df = pd.read_csv('/kaggle/input/titanic/train.csv')

# Preview
df.head()
```

Step 1: **Data Exploration**

Helps one understand-How your dataset is structured, What types of values you're dealing with and where there might be missing data, duplicates, or potential problems.

Key Functions for Initial Exploration
Here are some essential pandas functions used for initial exploration:

- df.head(): Displays the first few rows of the dataset to give you a quick preview.
- df.shape: Returns the number of rows and columns in the dataset.
- df.info(): Provides details about the columns, their data types, and the number of non-null (non-missing) values.
- df.describe(): Provides summary statistics (mean, median, min, max, etc.) for numerical columns.
- df.columns: Lists the names of all columns in the dataset.
- df.nunique(): Returns the number of unique values in each column.
- df.duplicated(): Checks for duplicate rows.

In [2]:
```
# Shape of the dataset
print("Shape of the dataset:", df.shape)
```

```
Shape of the dataset: (891, 12)
```

```
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 [4]:
```
# Summary statistics for numerical features
print("\nSummary statistics:")
df.describe()
```

```
Summary statistics:
```

Out[4]:

	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

```
# Column names
print("\nColumn names:", df.columns.tolist())
```

```
Column names: ['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Far
e', 'Cabin', 'Embarked']
```

In [6]:
```
# Number of unique values per column
print("\nUnique values per column:")
print(df.nunique())
```

```
Unique values per column:
PassengerId    891
Survived         2
Pclass           3
Name           891
Sex              2
Age             88
SibSp            7
Parch            7
Ticket         681
Fare           248
Cabin          147
Embarked         3
dtype: int64
```

In [8]:
```
# Check for duplicates
print("\nNumber of duplicate rows:", df.duplicated().sum())
```

```
Number of duplicate rows: 0
```

Step 2: **Handling missing values and outliers**

Handling missing values and outliers is a key part of cleaning the dataset before deeper analysis.

In this step, I handled missing values for Age, Cabin, and Embarked, and checked for outliers in Fare and Age.

Checking missing values

```
# Checking missing values again
print("Missing values before handling:")
print(df.isnull().sum())
```

```
Missing values before handling:
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
```

Handling missing numeric data.

- **Action:** fills all missing values in the **`Age`** column with the median age of the dataset
- **Purpose:** to handle missing data by replacing it with a representative value (the median), ensuring the `Age` column remains complete and usable in analysis or modeling without introducing bias from extreme values

```
# Fill missing Age with median
df['Age'].fillna(df['Age'].median(), inplace=True)
```

Handling missing categorical data

- **Action**: Fills all missing values in the **`Embarked`** column with the most frequent value (mode) found in that column.
- **Purpose**: To address missing categorical data by using the most common embarkation point, ensuring the column is complete and consistent for analysis or modeling.

```
# Fill missing Embarked with the most frequent value (mode)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)
```

Eliminating a column with too many missing values

- **Action**: removes the Cabin column entirely from the dataset.
- **Purpose**: To eliminate a column with too many missing values, which would be difficult to impute reliably and might introduce noise or bias in the analysis or modeling.

```
# Drop Cabin column (too many missing values)
df.drop(columns=['Cabin'], inplace=True)
```

Checking missing values after handling

- **Action**: Prints out the number of missing values in each column after data cleaning steps have been applied.
- **Purpose**: To verify that the missing data has been properly handled and confirm that the dataset is now clean and ready for further analysis or modeling.

```
# Check again after handling
print("\nMissing values after handling:")
print(df.isnull().sum())
```

```
Missing values after handling:
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

Step 3: **Univariate Analysis**

Univariate analysis focuses on analyzing one variable at a time.
This helps us understand:

- The distribution of numerical features (e.g., Age, Fare)

- The frequency of categories in categorical features (e.g., Sex, Pclass, Embarked)
- Potential skewness, outliers, or unusual values
- Whether transformations (e.g., log, bins) are needed

In this step I Explored both **numerical** and **categorical** variables in the dataset using visualizations and descriptive statistics.

Categorical

- **Action**: It Imports **Seaborn** and **Matplotlib** for data visualization, it sets a clean, white grid style for Seaborn plots and also performs over a list of **categorical features** ('Sex', 'Pclass', 'Embarked', 'Survived') and creates **count plots** for each.
- **Purpose**: To visually explore the distribution of categorical features, helping to identify class imbalances, patterns, or relationships that may influence survival on the Titanic.

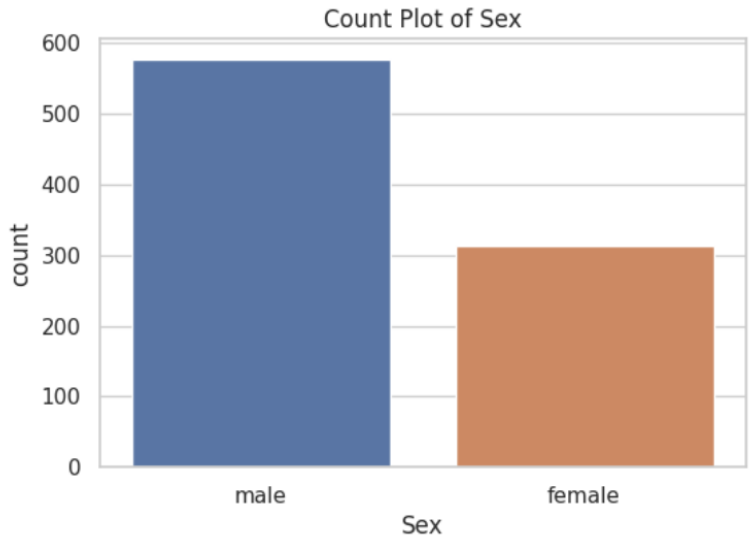
```
import seaborn as sns
import matplotlib.pyplot as plt

# Set style
sns.set(style="whitegrid")

# Plot categorical features
categorical_features = ['Sex', 'Pclass', 'Embarked', 'Survived']

for feature in categorical_features:
    plt.figure(figsize=(6, 4))
    sns.countplot(x=feature, data=df)
    plt.title(f'Count Plot of {feature}')
    plt.show()
```

Each count plot shows the **number of occurrences** for each category within a feature:

- **Sex**: Displays how many male and female passengers were on board.
- **Pclass**: Shows how many passengers were in each passenger class (1st, 2nd, 3rd).
- **Embarked**: Indicates how many passengers boarded from each port (C = Cherbourg, Q = Queenstown, S = Southampton).
- **Survived**: Displays the number of passengers who survived (1) versus those who did not (0).

These visualizations help identify class distribution and potential imbalances.

An example of the white grid style and Sex as a categorical feature.



Numerical

Action:

- Iterates through the numerical features Age and Fare.
- For each, it creates a histogram with a KDE curve using Seaborn's histplot, showing the distribution of values.
- Sets figure size, title, axis labels, and displays the plot.

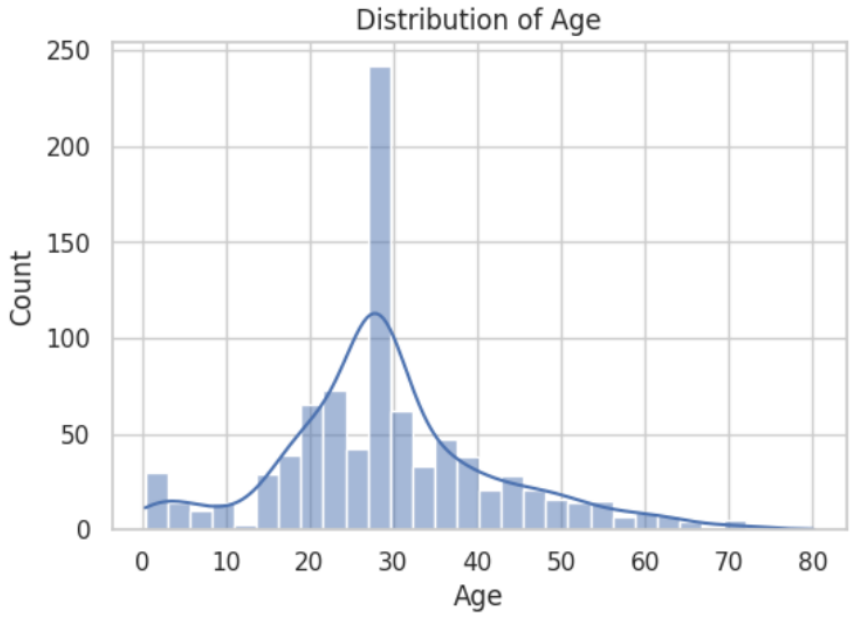
Purpose: To visualize the distribution of key numerical variables. This helps identify patterns such as skewness, peaks, and spread, and provides insight into the underlying structure of the data — useful for detecting outliers or deciding on transformations.

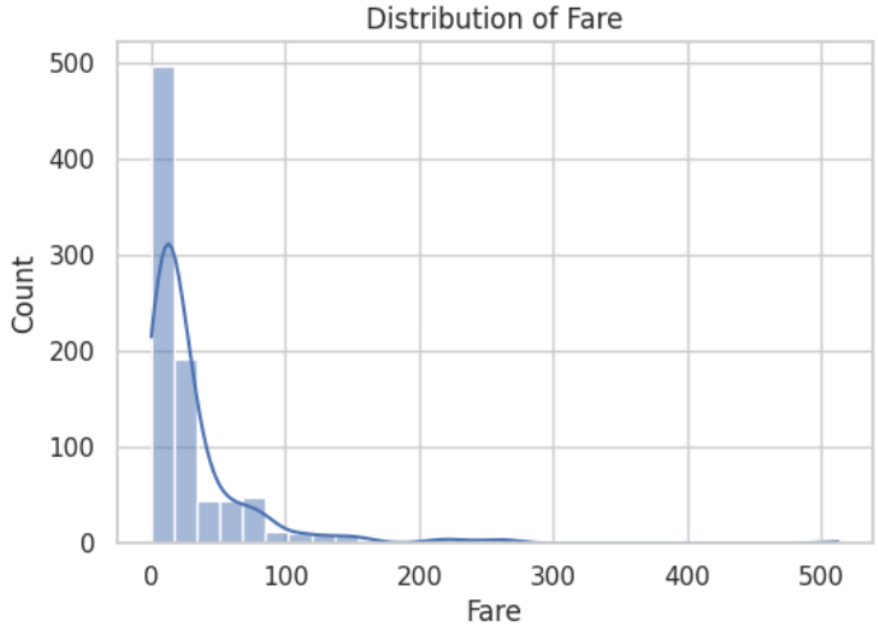
```
# Plot numerical features
numerical_features = ['Age', 'Fare']

for feature in numerical_features:
    plt.figure(figsize=(6, 4))
    sns.histplot(df[feature], kde=True, bins=30)
    plt.title(f'Distribution of {feature}')
    plt.xlabel(feature)
    plt.ylabel("Count")
    plt.show()
```

Each plot shows the **distribution shape** of the feature:

- **Age**: The histogram with KDE reveals how passenger ages are spread—often showing a peak around young adults and a long tail toward older ages.
- **Fare**: The plot typically shows a **right-skewed** distribution—most passengers paid lower fares, while a few paid very high amounts.

Examples:





Step 4: **Bivariate Analysis**

Bivariate analysis explores the relationship between two variables. This step helps us understand patterns or associations between features — for example, how gender affects survival, or how passenger class relates to fare price.

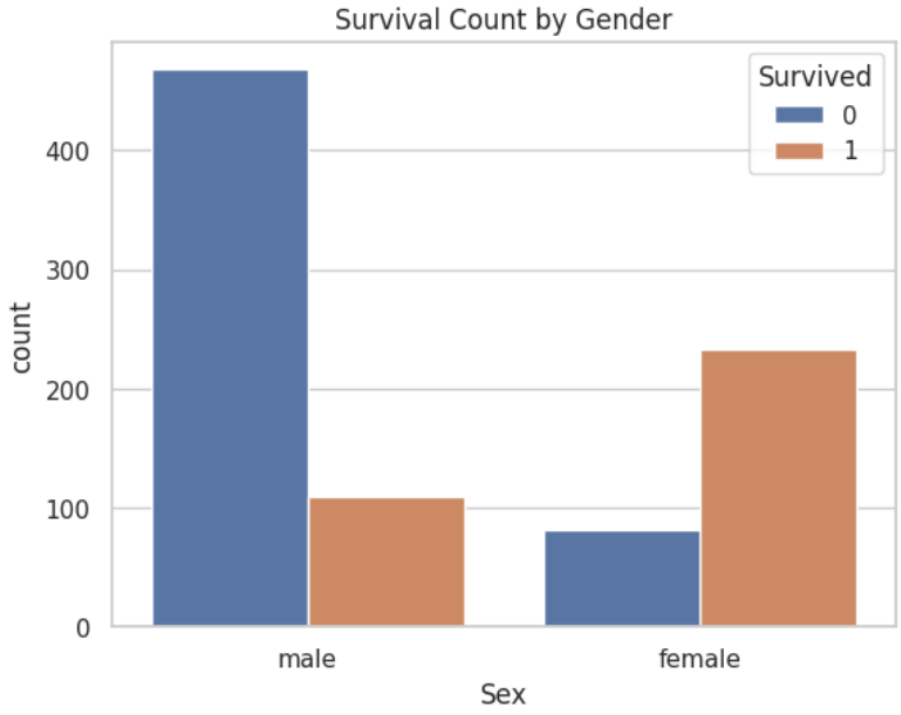
Key Plots Used:

- Bar plots
- Box plots
- Count plots (for categorical comparisons)

In this step, I explored the below:

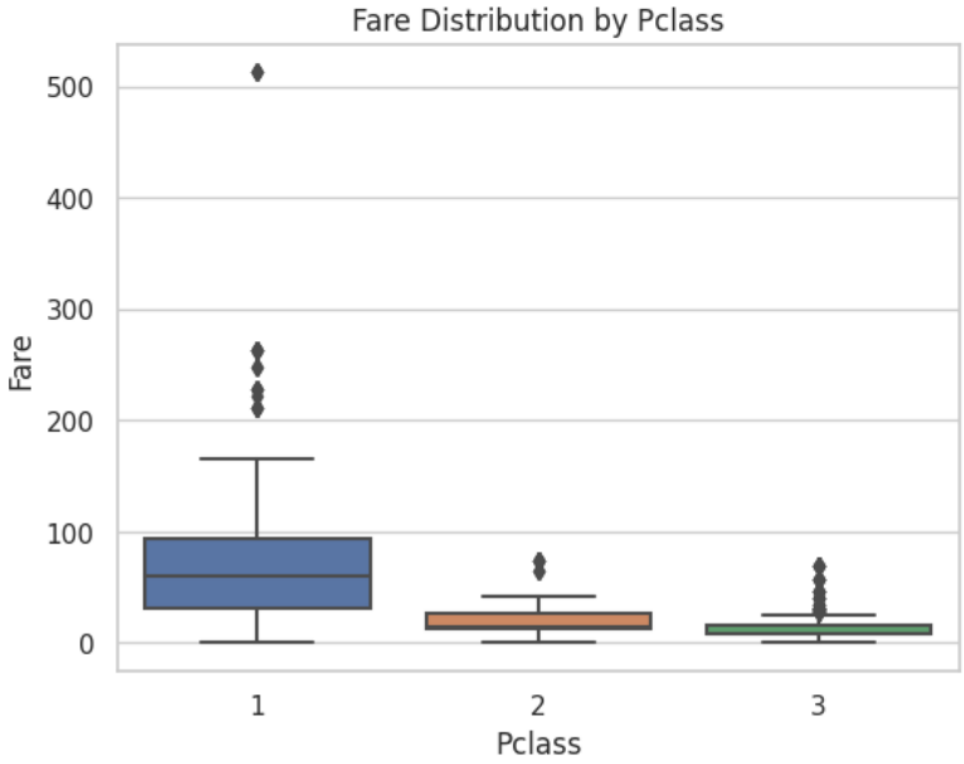
```
# Gender vs Survival
sns.countplot(x='Sex', hue='Survived', data=df)
plt.title('Survival Count by Gender')
plt.show()
```

- **Action**: Creates a count plot of the `Sex` column, with bars grouped by the `Survived` column using the `hue` parameter.
- **Purpose**: To compare survival rates by gender, visually showing how many males and females survived versus did not survive. This helps reveal patterns in survival across genders, which is important in the titanic dataset.



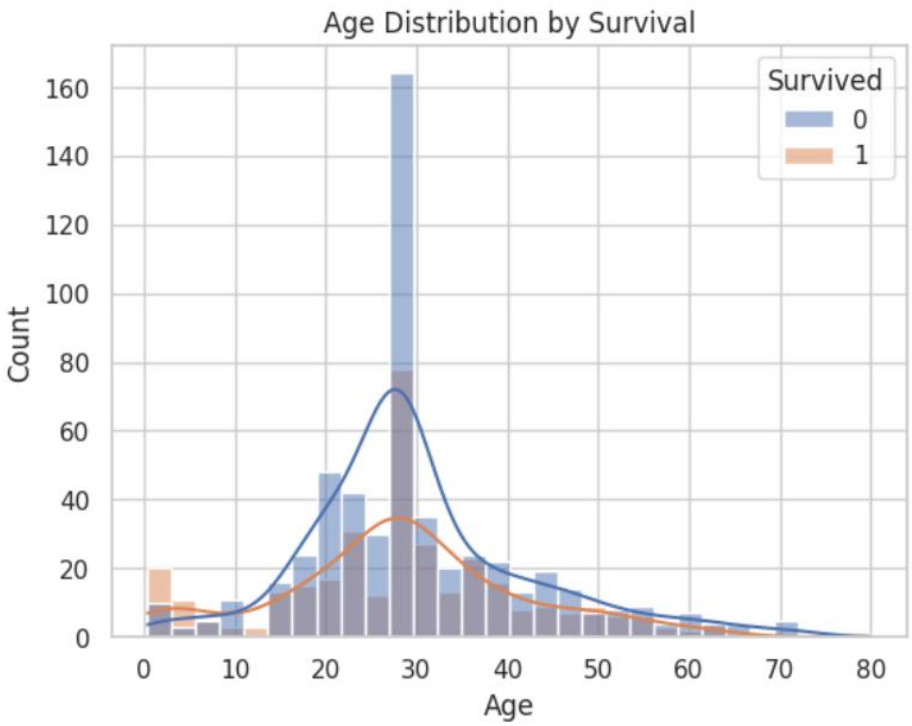

```
# Pclass vs Fare (Boxplot to compare distributions)
sns.boxplot(x='Pclass', y='Fare', data=df)
plt.title('Fare Distribution by Pclass')
plt.show()
```

- **Action**: Creates a boxplot showing the distribution of `Fare` for each passenger class (`Pclass`).
- **Purpose**: To visually compare fare ranges across different classes, including the median, quartiles. This helps understand how fare prices vary by travel class and highlights the presence of extreme values (e.g very high fares in 1st class)



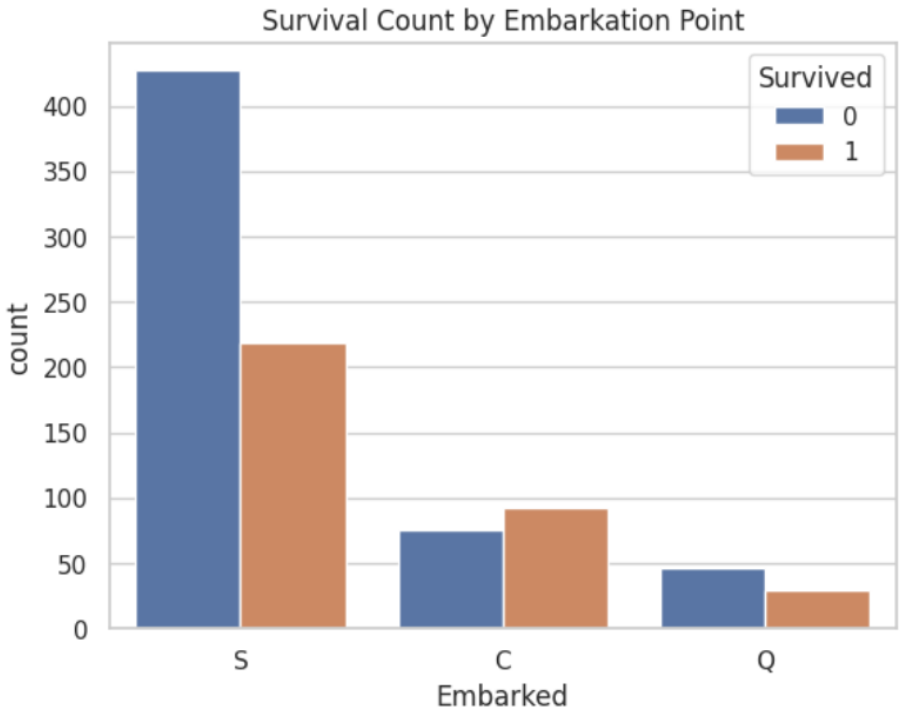

```
# Age vs Survival (using a histogram with hue)
sns.histplot(data=df, x='Age', hue='Survived', kde=True, bins=30)
plt.title('Age Distribution by Survival')
plt.show()
```

- **Action**: Creates a histogram of the `Age` feature, separated by survival status (`Survived`) using the `hue` parameter, and includes a KDE curve for smoother visualization.
- **Purpose**: To compare the age distribution of passengers who survived versus those who did not. This helps reveal age-related survival patterns-such as whether children or young adults had higher survival rates-providing insight into how age influenced survival outcomes.




```
# Embarked vs Survival
sns.countplot(x='Embarked', hue='Survived', data=df)
plt.title('Survival Count by Embarkation Point')
plt.show()
```

- **Action**: Creates a count plot showing the number of passengers from each embarkation point (`Embarked`), grouped by survival status using the `hue='Survived'` parameter.
- **Purpose**: To analyze survival rates across embarkation points (C = Cherbourg, Q = Queenstown, S = Southampton). This helps identify whether passengers from certain ports had higher chances of survival, possibly due to differences in class distribution or boarding conditions.



Step 5: **Multivariate Analysis**

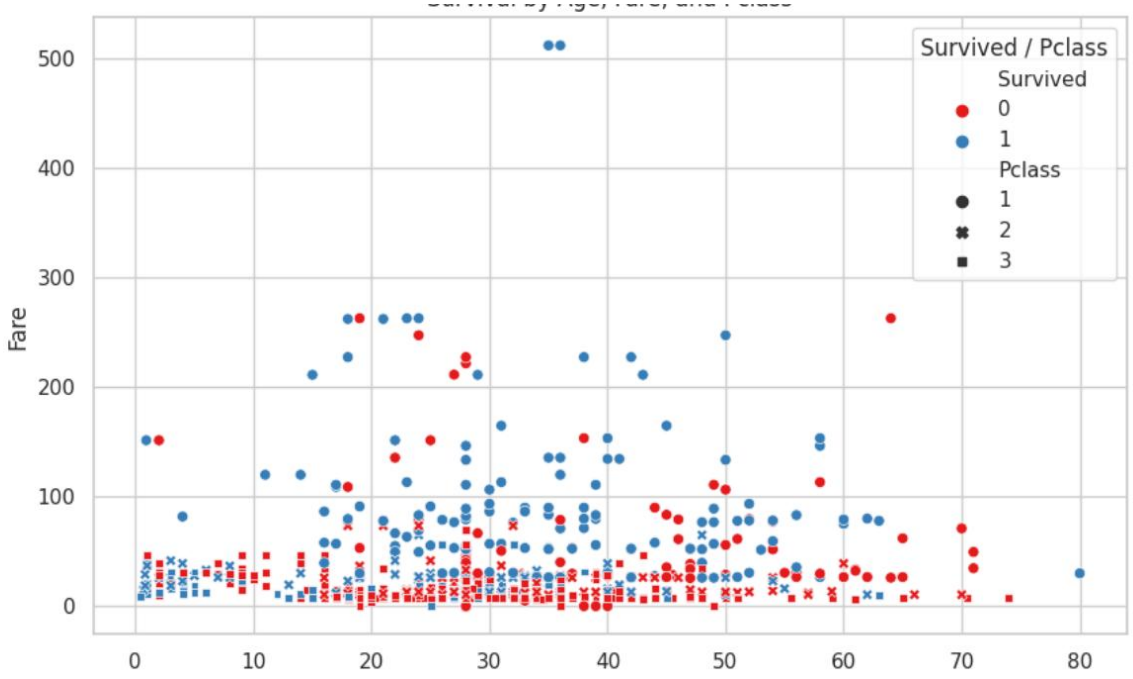
This involves examining three or more variables simultaneously to understand the combined effects they may have on one another. It is an important step when we want to uncover complex patterns or interactions that may not be visible when analyzing individual or paired variables alone.

In this step I used visualizations like scatter plots and count plots, to gain deeper insights into how multiple factors interact and impact the target variable (Survived).

```
import seaborn as sns
import matplotlib.pyplot as plt

# Plot 1: Age vs Fare with Survived and Pclass
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='Age', y='Fare', hue='Survived', style='Pclass', palette='Set1')
plt.title('Survival by Age, Fare, and Pclass')
plt.xlabel('Age')
plt.ylabel('Fare')
plt.legend(title='Survived / Pclass')
plt.show()
```

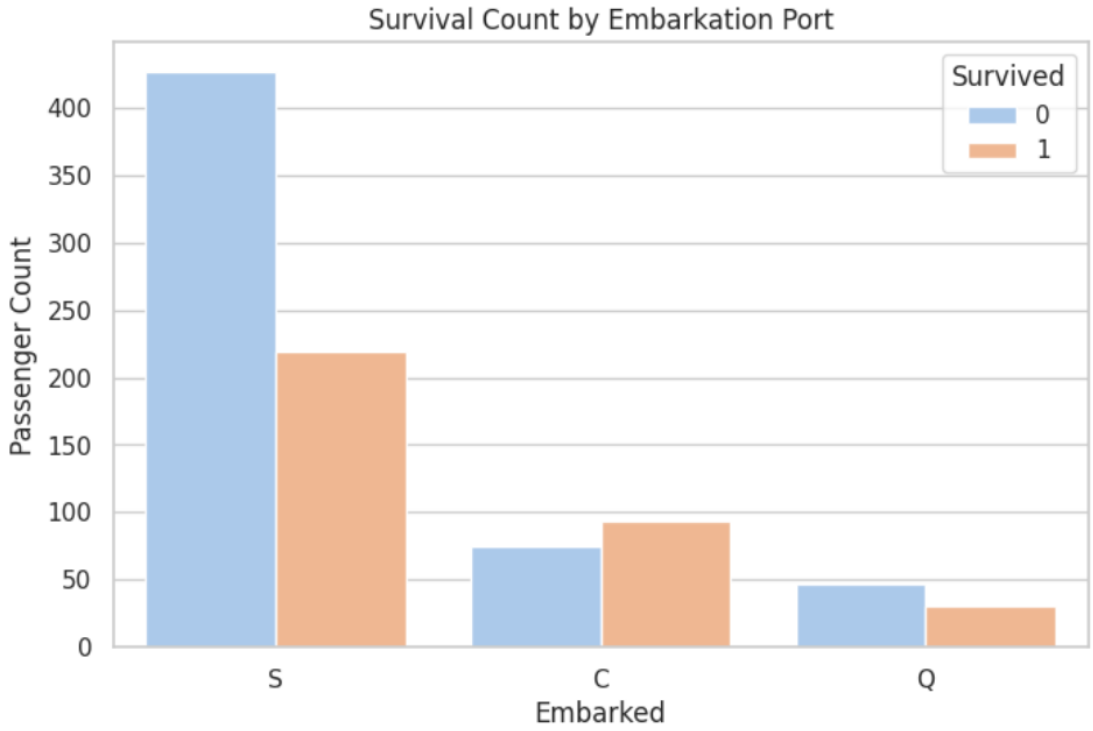
- **Action**: Creates a scatter plot with `Age` on the x-axis and `Fare` on the y-axis. Points are colored by Survival status (`hue='Survived'`) and styled by Passenger Class (`style='Pclass'`)

- **Purpose**: To visualize relationships between age, fare, passenger class, and survival in one plot. It helps reveal patterns-such as whether younger or higher-paying passengers in certain classes were more likely to survive-offering a multivariate view of the data



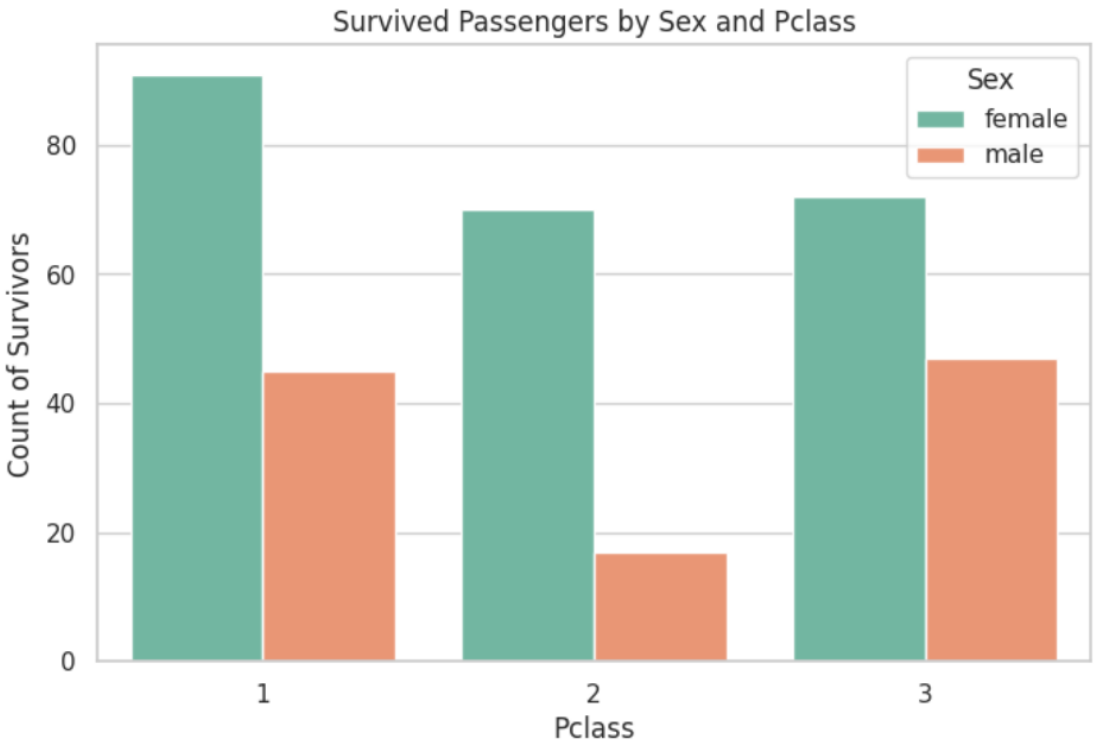

```
# Plot 2: Embarked vs Survival with hue as Survived
plt.figure(figsize=(8, 5))
sns.countplot(x='Embarked', hue='Survived', data=df, palette='pastel')
plt.title('Survival Count by Embarkation Port')
plt.xlabel('Embarked')
plt.ylabel('Passenger Count')
plt.show()
```

- **Action**: Creates a count plot showing the number of passengers from each embarkation port (`Embarked`), split by survival status using `hue='Survived'` and a pastel color palette
- **Purpose**: To compare survival rates across different embarkation points (C, Q, S). This helps determine if the port of boarding is related to passenger survival, possibly revealing location-based factors such as socio-economic class distribution.




```
# Plot 3: Pclass vs Sex for only Survived passengers
plt.figure(figsize=(8, 5))
sns.countplot(x='Pclass', hue='Sex', data=df[df['Survived'] == 1], palette='Set2')
plt.title('Survived Passengers by Sex and Pclass')
plt.xlabel('Pclass')
plt.ylabel('Count of Survivors')
plt.show()
```

- **Action** Creates a count plot of passenger class (`Pclass`), split by gender (`Sex`), but only for passengers who survived (`Survived == 1`), using a distinct color palette (`Set2`)
- **Purpose**: To analyze the distribution of survivors by class and gender, revealing patterns such as whether females or certain classes had higher survival rates. This helps highlight the interaction between gender, socio-economic status (via class), and survival outcomes.



Step 6: **Target Variable Analysis**

The target variable in the Titanic dataset is **Survived**. It indicates whether a passenger survived (1) or not (0).
The goal is to:

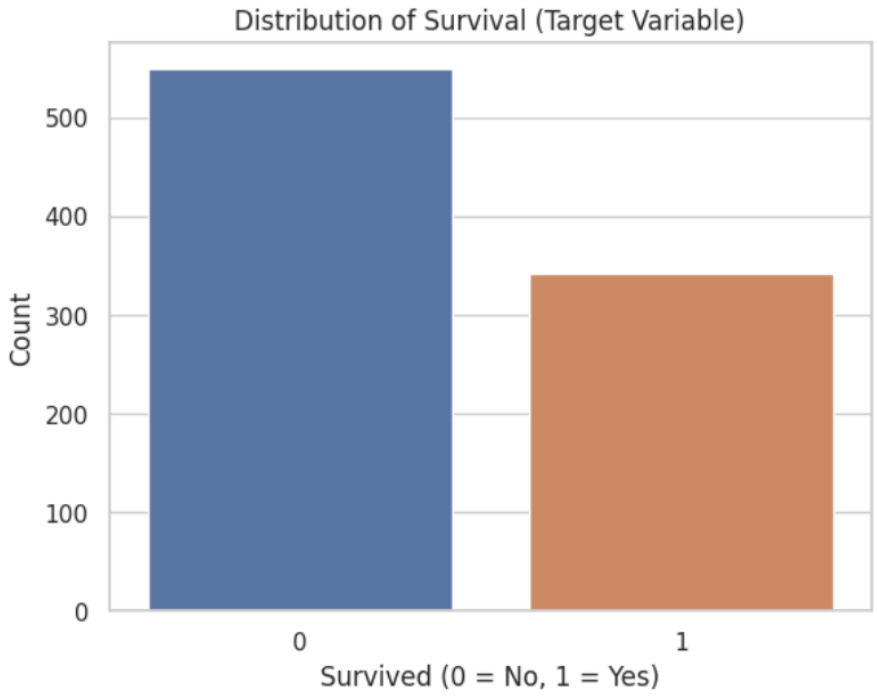
- Understand the distribution of the target variable
- Check for class imbalance
- Analyze how survival correlates with other key features
- Detect interaction effects

First analyze the distribution of the target variable, Survived

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(x='Survived', data=df)
plt.title('Distribution of Survival (Target Variable)')
plt.xlabel('Survived (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.show()
```

- **Action**: Generates a count plot showing the number of passengers who survived (`1`) and did not survive (`0`) using the `Survived` column as the x-axis.

- **Purpose**: To visualize the distribution of the target variable, helping to assess class balance. This is important for understanding the overall survival rate and for evaluating whether there's an imbalance that might affect model training or analysis.



Is the dataset balanced?

A balanced dataset means classes have almost equal instances. We will check the proportion of survivors to non-survivors.

```
survival_counts = df['Survived'].value_counts(normalize=True)
print(survival_counts)
```

- **Action**: Calculates the proportion of survivors and non-survivors in the dataset using `value_counts(normalize=True)`, and prints the result.
- **Purpose**: To get a quick overview of survival rates in percentage terms. This helps in understanding class imbalance (e.g., if far fewer passengers survived), which is important for interpreting the data and building predictive models fairly.

```
Survived
0    0.616162
1    0.383838
Name: proportion, dtype: float64
```

The target variable Vs Key features

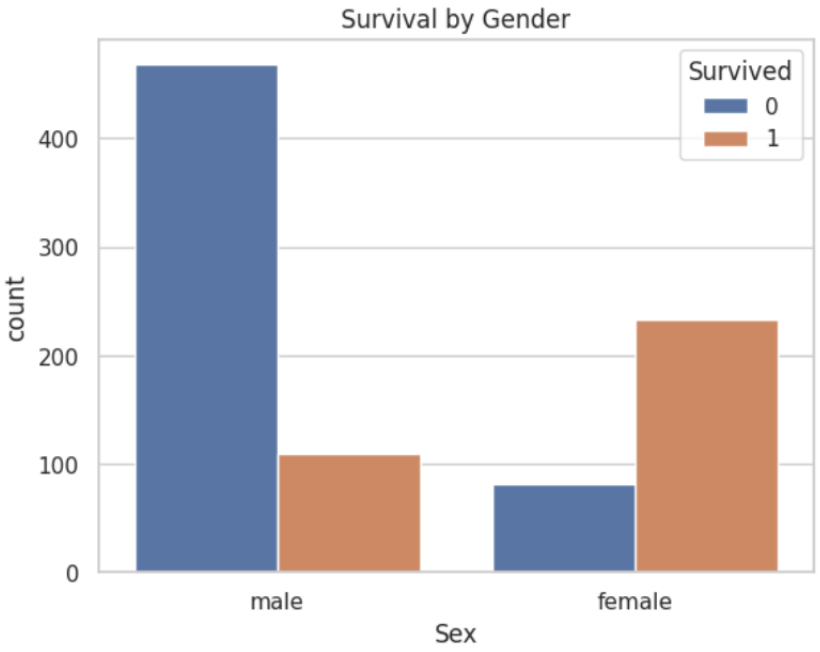
This shows how survival relates to Sex, Pclass, Age, and Embarked.

For example,

Survival by Gender:

```
sns.countplot(x='Sex', hue='Survived', data=df)
plt.title('Survival by Gender')
plt.show()
```

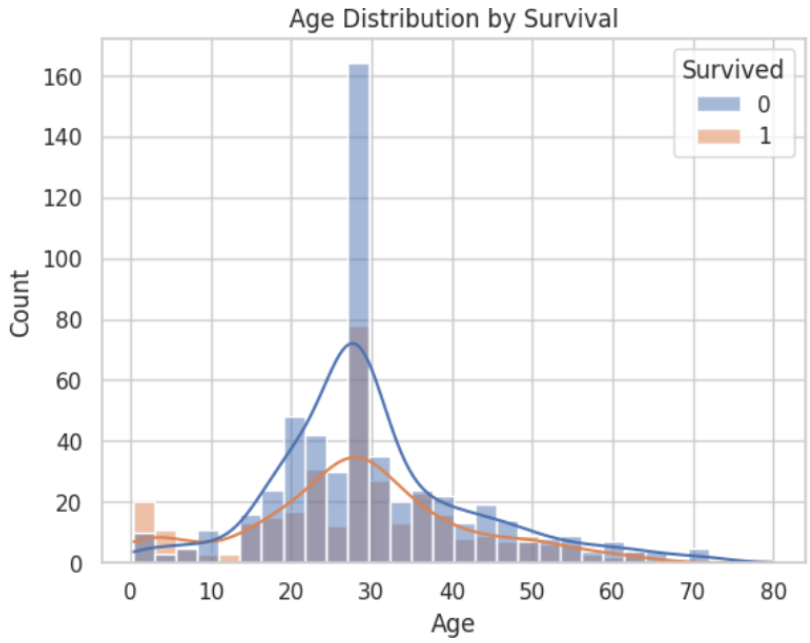
- **Action**: Creates a count plot of passengers by gender (`Sex`), with bars grouped by survival status (`Survived`) using the `hue` parameter.
- **Purpose**: To compare survival outcomes between males and females**,** making it easy to observe gender-based differences in survival rates-which is an important pattern in the Titanic dataset that shows a higher survival rate among females.



Survival by Age (Histogram)

```
sns.histplot(data=df, x='Age', hue='Survived', kde=True, bins=30)
plt.title('Age Distribution by Survival')
plt.show()
```

- **Action**: Creates a histogram of the `Age` column with overlaid KDE curves, using `hue='Survived'` to differentiate between those who survived and those who didn't.
- **Purpose**: To compare the age distribution of survivors and non-survivors. This helps identify age groups that were more or less likely to survive-for example, showing whether children or younger adults had higher survival rates compared to older passengers.



Link to Code:

https://www.kaggle.com/code/shirleensimon/titanic-dataset-eda-project-shirleen

Conclusion

In this Exploratory Data Analysis (EDA), I explored the Titanic dataset to uncover patterns related to passenger survival. By cleaning missing data and analyzing distributions and relationships, I identified key factors affecting survival outcomes.

This process helped me understand which variables are most influential and laid a strong foundation for building predictive models in future steps.