Techno Hacks Edutech Data analytics Internship

Project Title : Data Analysis of Titanic Dataset 📊 📈

- Author: Rishabh Tiwari
- LinkedIn: https://www.linkedin.com/in/smsrishabh/ (https://www.linkedin.com/in/smsrishabh/)
- Gmail : smsrishabh@gmail.com (mailto:smsrishabh@gmail.com)

Task List 📋

Complete any 3 tasks of your choise

✔ Task 1 : Perform Data Cleaning

Clean a dataset by removing missing values and outliers

✔ Task 2 : Calculate summary statistics

Calculate summary statistics (mean, median, mode, standard deviation) for a dataset

✔ Task 3 : Visualization using Histogram

Create a histogram or bar chart to visualize the distribution of data in a dataset

✔ Task 4 : Pivote Table

Use pivot tables to summarize data in a dataset

✔ Task 5 : Remove Duplication

Identify and remove duplicate values in a dataset.

✔ Task 6 : Dashboard using Tableau

Create a dashboard to display insights and visualizations from a dataset using Tableau

✔ Task 7 : Dashboard using PowerBI

Create a dashboard to display insights and visualizations from a dataset.

Introduction

Dataset Description ⏱️

The Titanic dataset is a collection of information about passengers aboard the Titanic, the famous ship that sank in 1912. The dataset includes details such as passengers' names, ages, genders, ticket classes, and whether they survived or not. It's often used in data science and machine learning to predict survival based on these features. The goal is to build a model using the provided data to predict whether a passenger survived the tragic event. The dataset is commonly used for learning and practicing predictive modeling techniques.

Data Dictonary 📘 📖:

- Here's an overview of the columns in the Titanic dataset:

`PassengerId` : A unique identifier assigned to each passenger.

`Survived` : Indicates whether the passenger survived (1) or did not survive (0).

`Pclass` (Passenger Class): Represents the ticket class (1st, 2nd, or 3rd class).

`Name` : The name of the passenger.

`Sex` : The gender of the passenger (male or female).

`Age:` The age of the passenger. (Note: Some values may be missing.)

`SibSp:` The number of siblings/spouses aboard the Titanic.

`Parch:` The number of parents/children aboard the Titanic.

`Ticket:` The ticket number.

`Fare:` The amount of money spent on the ticket.

`Cabin:` The cabin number where the passenger stayed. (Note: Many values are missing.)

`Embarked:` The port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton).

Step 1| Import libraries

Tabel of Contents

In [186]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import zscore
```

Step 2| Load Dataset

Tabel of Contents

In [187]:
```
# Load Data set  of train and test
df_Train = pd.read_csv('train.csv')
df_Test = pd.read_csv('test.csv')
df_gender_submission = pd.read_csv('gender_submission.csv')
```

In [188]:
```
df_Train.isnull().sum()
print("Train Shape:",df_Train.shape)
df_Test.isnull().sum()
print("Test Shape:",df_Test.shape)
```

```
Train Shape: (891, 12)
Test Shape: (418, 11)
```

The following function `print_heading` is used only for the purpose of printing headings text surrounded by lines to make the output look more readable.

In [189]:
```
# print heading - for display purposes only
def heading(heading):
    print('-' * 50)
    print(heading.upper())
    print('-' * 50)
```

In [190]:
```
# Let's Explore the data of train.csv

heading('Train Data')
df_Train.head()
```

```
--------------------------------------------------
TRAIN DATA
--------------------------------------------------
```

Out[190]:

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

Step 3| Dataset Overview

⬆️ [Tabel of Contents]

Step 3.1.1 | Basic Information 💁

In [191]:
```
heading('Information of Train Data')
df_Train.info()
```

```
--------------------------------------------------
INFORMATION OF TRAIN DATA
--------------------------------------------------
<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
```

Inferences:

- Dataset contain 418 Rows and 11 Columns
- The dataset contains 418 entries, suggesting information on 418 passengers.
- The 'Age' column has 332 non-null values, indicating that there are missing age values for some passengers.
- The 'Fare' column has 417 non-null values, indicating that there are missing fare values of some passengers.
- The 'Cabin' column has only 91 non-null values, suggesting a significant amount of missing cabin information.

Step 3.1.2 | Descriptive Statistics 📋

In [192]:
```
heading('Statistically approach of the data')
df_Train.describe()
```

```
--------------------------------------------------
STATISTICALLY APPROACH OF THE DATA
--------------------------------------------------
```

Out[192]:

	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 [193]:
```
# Function to print a summary of the DataFrame's column types

def print_data_summary(df_Train):
    # Count the number of categorical columns
    categorical_count = df_Train.select_dtypes(include=['category']).shape[

    # Count the number of float columns
    float_count = df_Train.select_dtypes(include=['float64']).shape[1]

    # Count the number of integer columns
    int_count = df_Train.select_dtypes(include=['int64']).shape[1]


    # Print the counts in a bullet-point format
    print(f"• Categorical columns: {categorical_count}")
    print(f"• Float columns: {float_count}")
    print(f"• Integer columns: {int_count}")


# Print a heading for the data summary section (assuming 'heading' is a def

heading('Data Summary')
print_data_summary(df_Train)
```

```
--------------------------------------------------
DATA SUMMARY
--------------------------------------------------
• Categorical columns: 0
• Float columns: 2
• Integer columns: 5
```

Step 4| Exploratory Data Analysis (EDA) 🚀 🔍

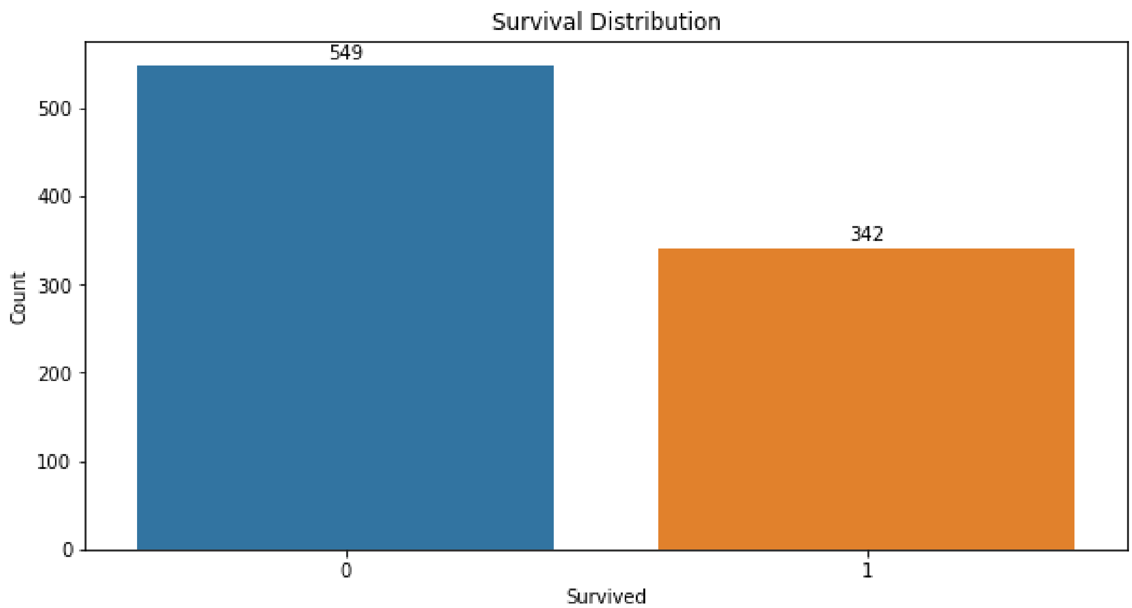
⬆️ [Tabel of Contents]

`Exploratory Data Analysis (EDA)` is the process of examining and visualizing a

Step 4.1.1 | Survival Distribution

This will visualize the distribution of survivors.

In [194]:
```
plt.figure(figsize=(10, 5))
ax = sns.countplot(x='Survived', data=df_Train)

# Add count numbers on top of the bars
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x() + p.get_width()/2., height + 3, f'{height}', ha='cent

plt.title('Survival Distribution')
plt.xlabel('Survived')
plt.ylabel('Count')
plt.show()
```



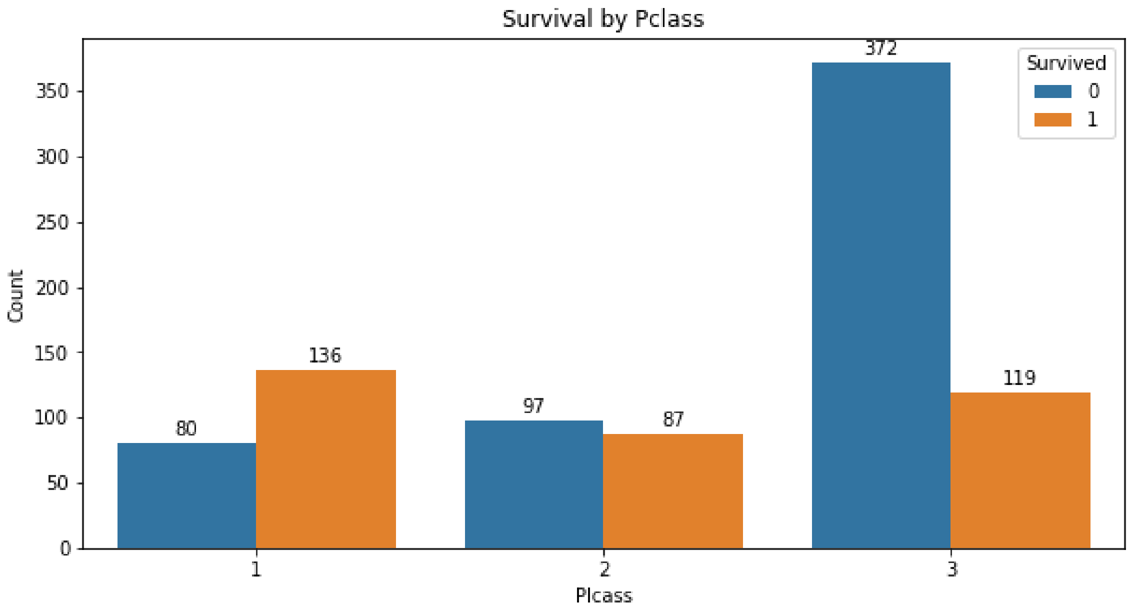
Inferences:

- The count plot indicates a higher number of individuals who did not survive, with 549, compared to those who survived, which is 342 (38%)

Step 4.1.2 | Pclass and Survival

This will visualize the relation between survivors and Pclass.

In [195]:
```
plt.figure(figsize=(10, 5))

ax= sns.countplot(x='Pclass', hue='Survived', data=df_Train)

# Add count numbers on top of the bars
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x() + p.get_width()/2., height +3, f'{height}', ha='cente

plt.title('Survival by Pclass')
plt.xlabel('Plcass')
plt.ylabel('Count')
plt.show()
```



Inferences:

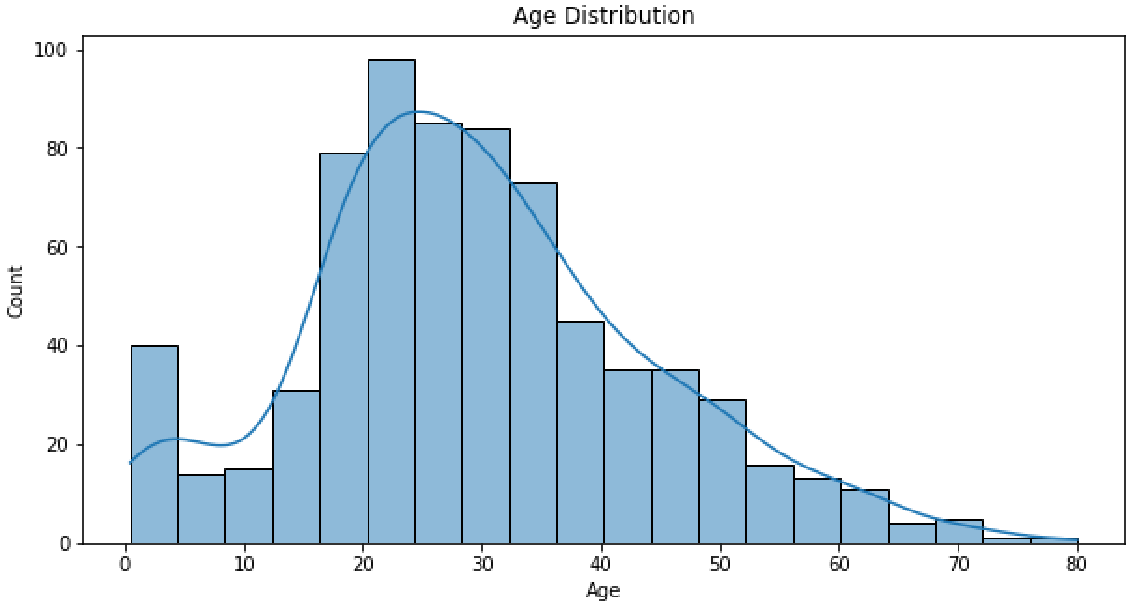
- Passengers in Pclass 1 exhibited the highest survival rate, with 136 individuals surviving.
- Pclass 3 had the second-highest survival rate, with 119 individuals surviving.
- Pclass 3 also experienced the highest death rate, with 372 individuals not surviving.
- Following Pclass 3, Pclass 2 had the second-highest death rate, with 97 individuals not surviving.

- Pclass 1 had a lower death rate compared to Pclass 2, with 80 individuals not surviving.

Step 4.1.3 | Age Distribution

This will visualize the Age Distribution.

In [196]:
```
plt.figure(figsize=(10, 5))
sns.histplot(x='Age', data=df_Train, bins=20, kde=True)
plt.title('Age Distribution')
plt.show()
```



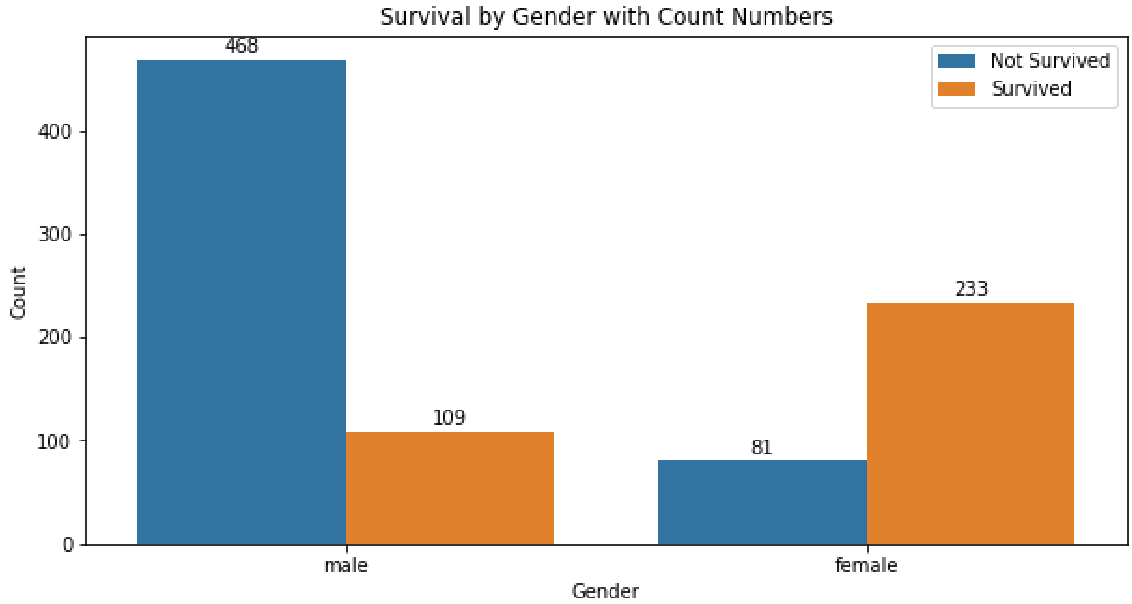
Inferences:

- Most people in the dataset are between 20 and 30 years old, showing a concentration in this age range.
- The ages are spread out across different groups, indicating a diverse mix of passengers covering almost all age ranges.

Step 4.1.4 | Gender and Survival

This will visualize the Gender and Survival.

In [197]:
```
plt.figure(figsize=(10, 5))
ax = sns.countplot(x='Sex', hue='Survived', data=df_Train)

# Add count numbers on top of the bars
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x() + p.get_width()/2., height + 3, f'{height}', ha='cent

plt.title('Survival by Gender with Count Numbers')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.legend(labels=['Not Survived', 'Survived'])
plt.show()
```



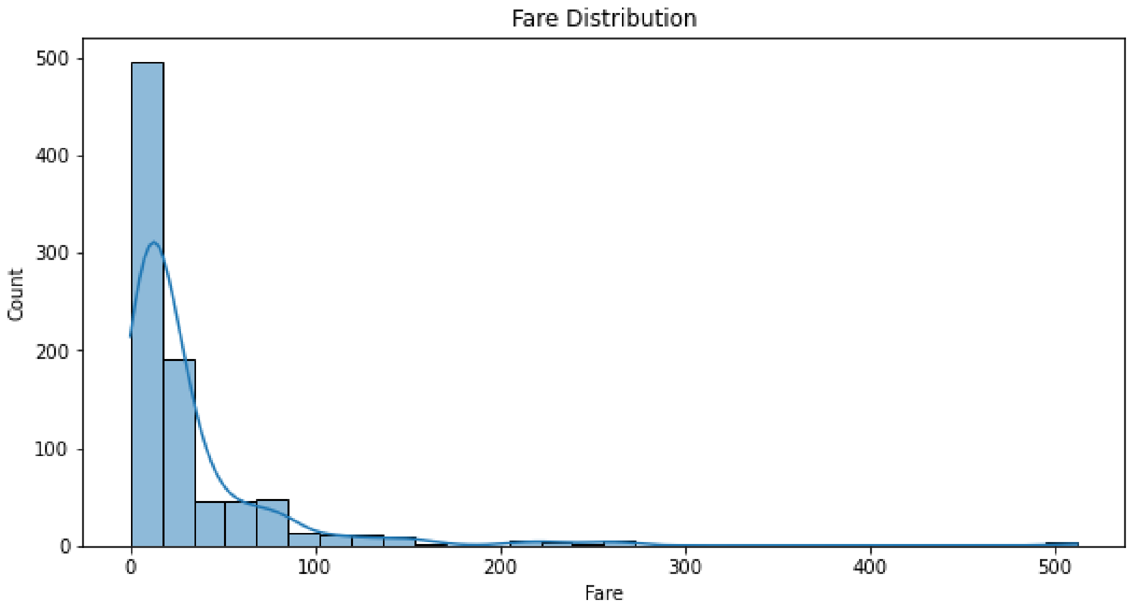
Inferences:

- More females, specifically 233, survived compared to males, who had a count of 109 survivors.
- On the other hand, the number of males not surviving is noticeably higher at 468, in contrast to females where the count of non-survivors is only 81.

Step 4.1.5 | Fare Distribution

```
This will visualize the Fare Distribution.
```

In [198]:
```
plt.figure(figsize=(10, 5))
sns.histplot(x='Fare', data=df_Train, bins=30, kde=True)
plt.title('Fare Distribution')
plt.show()
```



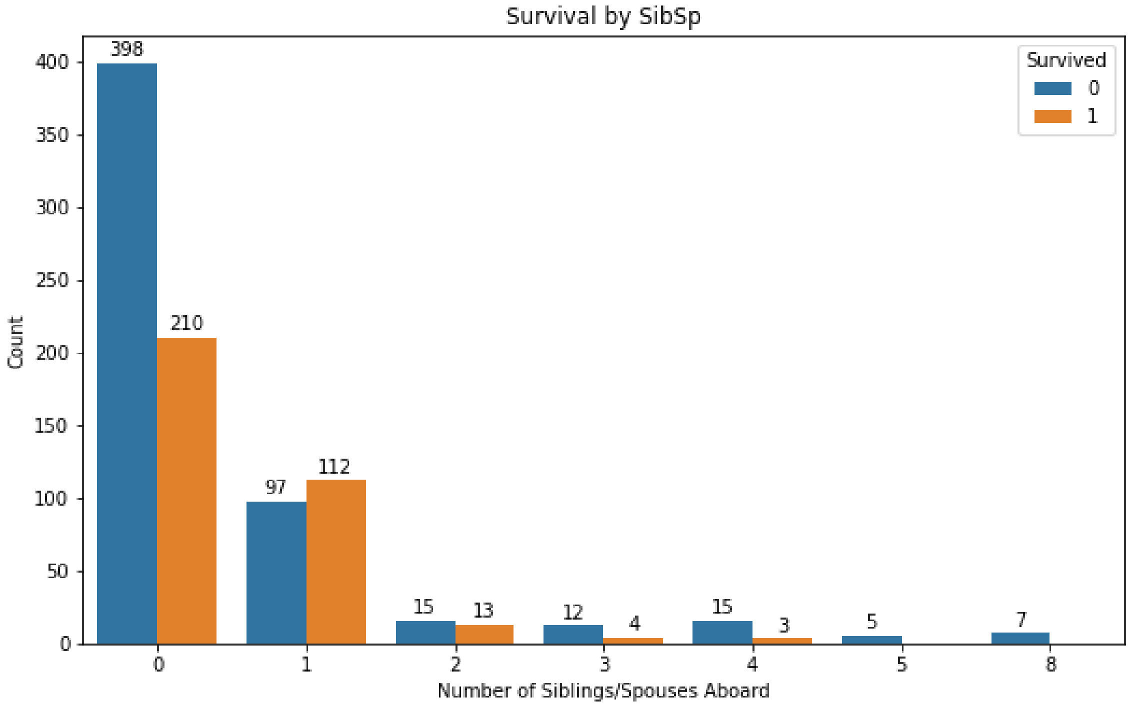
Inferences:

- Many passengers paid lower fares, and most people didn't spend much on their tickets.
- The fare distribution is uneven, showing that a few passengers paid a lot more for their tickets.
- The graph highlights the majority of passengers paying lower fares, with only a few spending more.
- Overall, the fare distribution is varied, with a mix of low and high fare payments.

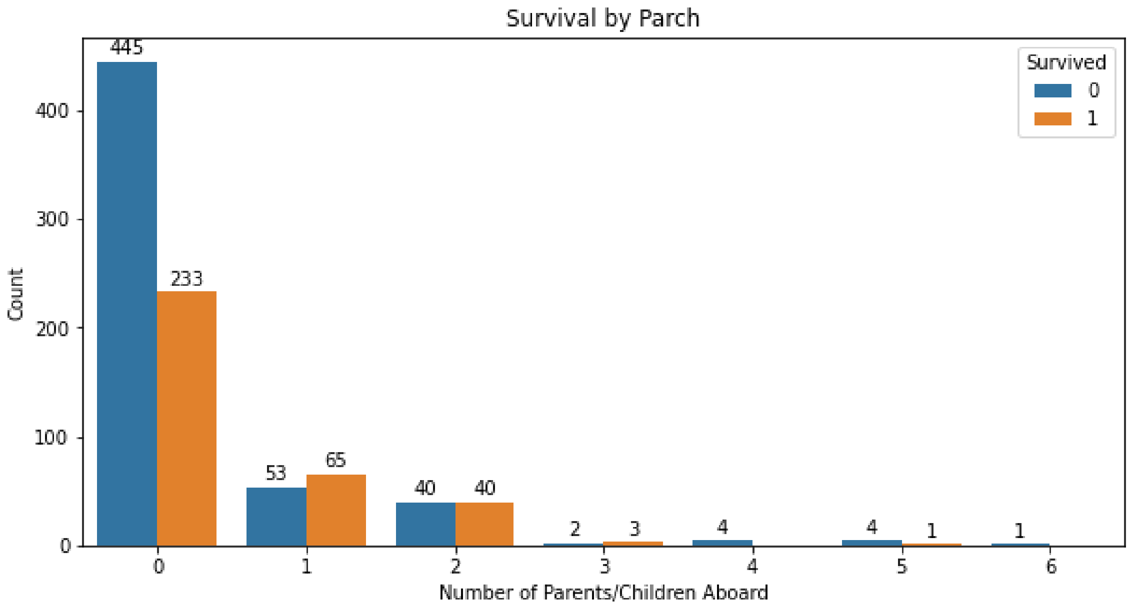
Step 4.1.6 | SibSp and Parch

This will visualize the relation between Survival with SibSp and Parch.

In [199]:
```
plt.figure(figsize=(10, 6))
ax = sns.countplot(x='SibSp', hue='Survived', data=df_Train)

# Add count numbers on top of the bars with adjusted position
for p in ax.patches:
    height = p.get_height()
    if np.isnan(height):  # Check if height is NaN
        continue
    ax.text(p.get_x() + p.get_width() / 2., height + 3, f'{int(height)}', h

plt.title('Survival by SibSp')
plt.xlabel('Number of Siblings/Spouses Aboard')
plt.ylabel('Count')

# Adjust legend placement
plt.legend(title='Survived', loc='upper right')

plt.show()
```



In [200]:
```
plt.figure(figsize=(10, 5))
ax = sns.countplot(x='Parch', hue='Survived', data=df_Train)

# Add count numbers on top of the bars with adjusted position
for p in ax.patches:
    height = p.get_height()
    if np.isnan(height):  # Check if height is NaN
        continue
    ax.text(p.get_x() + p.get_width() / 2., height + 3, f'{int(height)}', h

plt.title('Survival by Parch')
plt.xlabel('Number of Parents/Children Aboard')
plt.ylabel('Count')

# Adjust legend placement
plt.legend(title='Survived', loc='upper right')

plt.show()
```



Inferences:

Survival by Siblings/Spouses (SibSp):

- Passengers with no siblings or spouses aboard (SibSp=0) had a higher chance of not surviving.
- As the number of siblings or spouses increased (SibSp > 0), the likelihood of survival tended to decrease.
- Individuals with a moderate number of siblings or spouses (SibSp=1 or 2) had a relatively balanced survival outcome.
- Passengers with a larger number of siblings or spouses (SibSp > 2) generally faced a higher risk of not surviving.

Survival by Parents/Children (Parch):

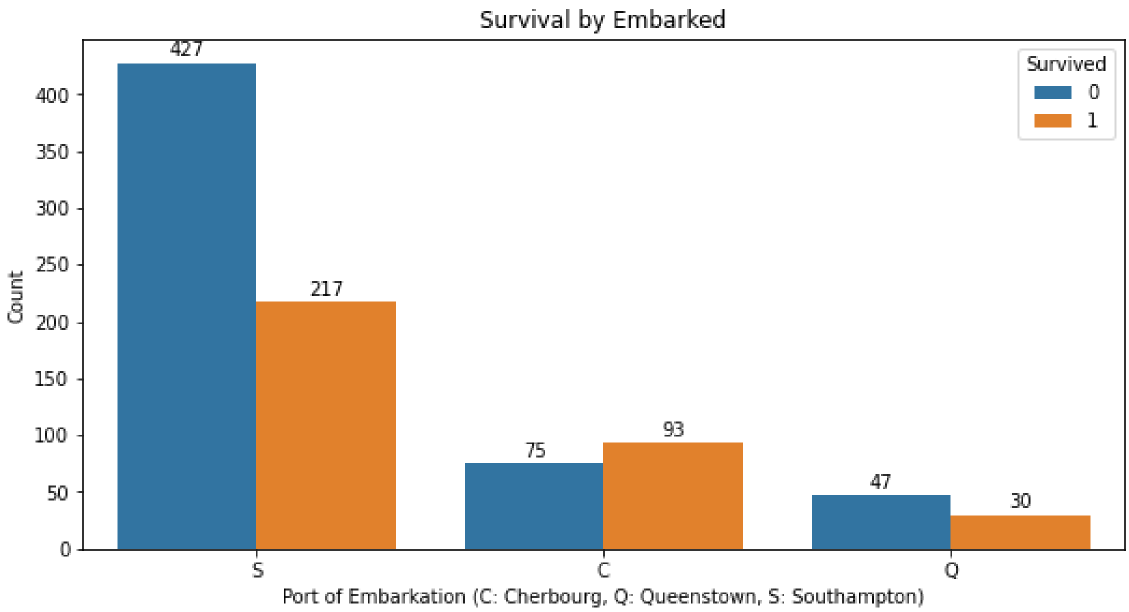
- Passengers without parents or children aboard (Parch=0) were more likely to not survive.

- Families with a small number of parents or children (Parch=1 or 2) showed a more balanced survival distribution.
- Larger families with more parents or children (Parch > 2) had a higher likelihood of not surviving.
- Passengers traveling alone (Parch=0) or with a small family (Parch=1 or 2) had better chances of survival compared to those with larger families.

Step 4.1.7 | Embarked and Survival

This will visualize the relation between Embarked and Survival.

In [201]:
```
plt.figure(figsize=(10, 5))
ax = sns.countplot(x='Embarked', hue='Survived', data=df_Train)

# Add count numbers on top of the bars with adjusted position
for p in ax.patches:
    height = p.get_height()
    if np.isnan(height):  # Check if height is NaN
        continue
    ax.text(p.get_x() + p.get_width() / 2., height + 3, f'{int(height)}', h

plt.title('Survival by Embarked')
plt.xlabel('Port of Embarkation (C: Cherbourg, Q: Queenstown, S: Southampto
plt.ylabel('Count')

# Adjust legend placement
plt.legend(title='Survived', loc='upper right')

plt.show()
```



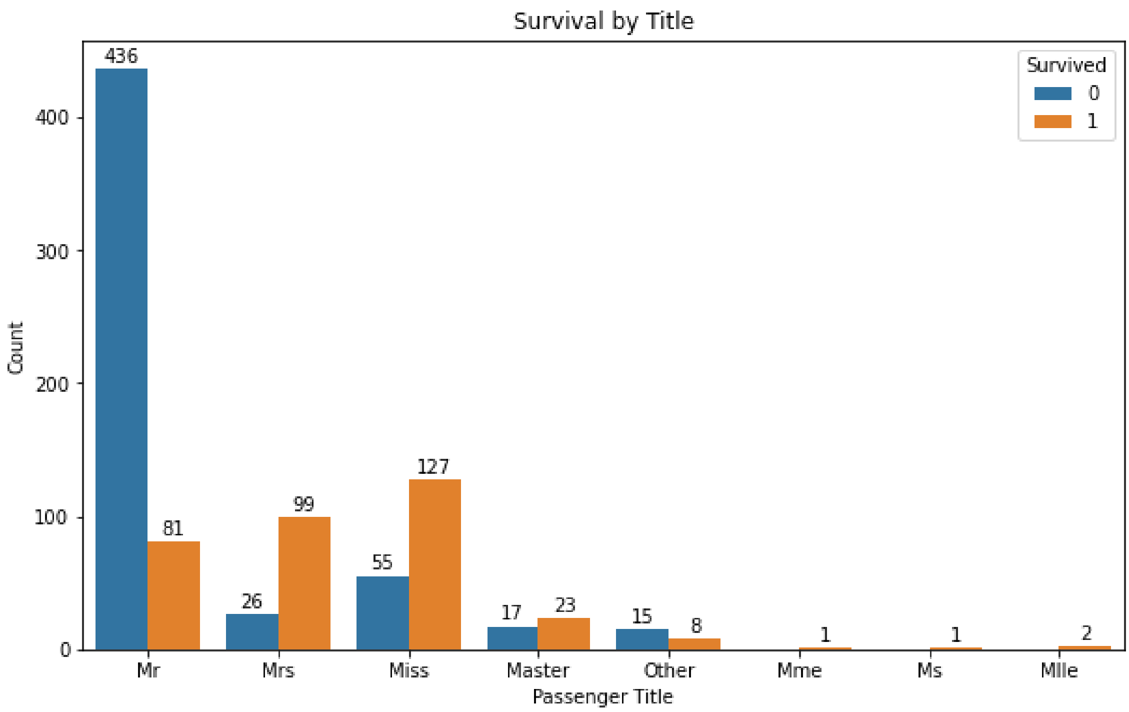
Inferences:

- Passengers who boarded from port 'C' (Cherbourg) had a higher chance of survival compared to those from ports 'S' (Southampton) and 'Q' (Queenstown).
- The majority of passengers from ports 'S' and 'Q' did not survive, contributing to a higher count of non-survivors. Port 'C' shows a relatively balanced distribution between survivors and non-survivors, indicating a more favorable survival rate.

Step 4.1.8 | Survival by Title

This will visualize the relation between Survival by there respective title.

In [202]:
```
# Create a new feature 'Title' from the 'Name' column
df_Train['Title'] = df_Train['Name'].str.extract(' ([A-Za-z]+)\.', expand=F

# Group rare titles into 'Other'
df_Train['Title'] = df_Train['Title'].replace(['Lady', 'Countess','Capt', '

plt.figure(figsize=(10, 6))
ax = sns.countplot(x='Title', hue='Survived', data=df_Train)

# Add count numbers on top of the bars with adjusted position
for p in ax.patches:
    height = p.get_height()
    if np.isnan(height):  # Check if height is NaN
        continue
    ax.text(p.get_x() + p.get_width() / 2., height + 3, f'{int(height)}', h

plt.title('Survival by Title')
plt.xlabel('Passenger Title')
plt.ylabel('Count')

# Adjust legend placement
plt.legend(title='Survived', loc='upper right')

plt.show()
```



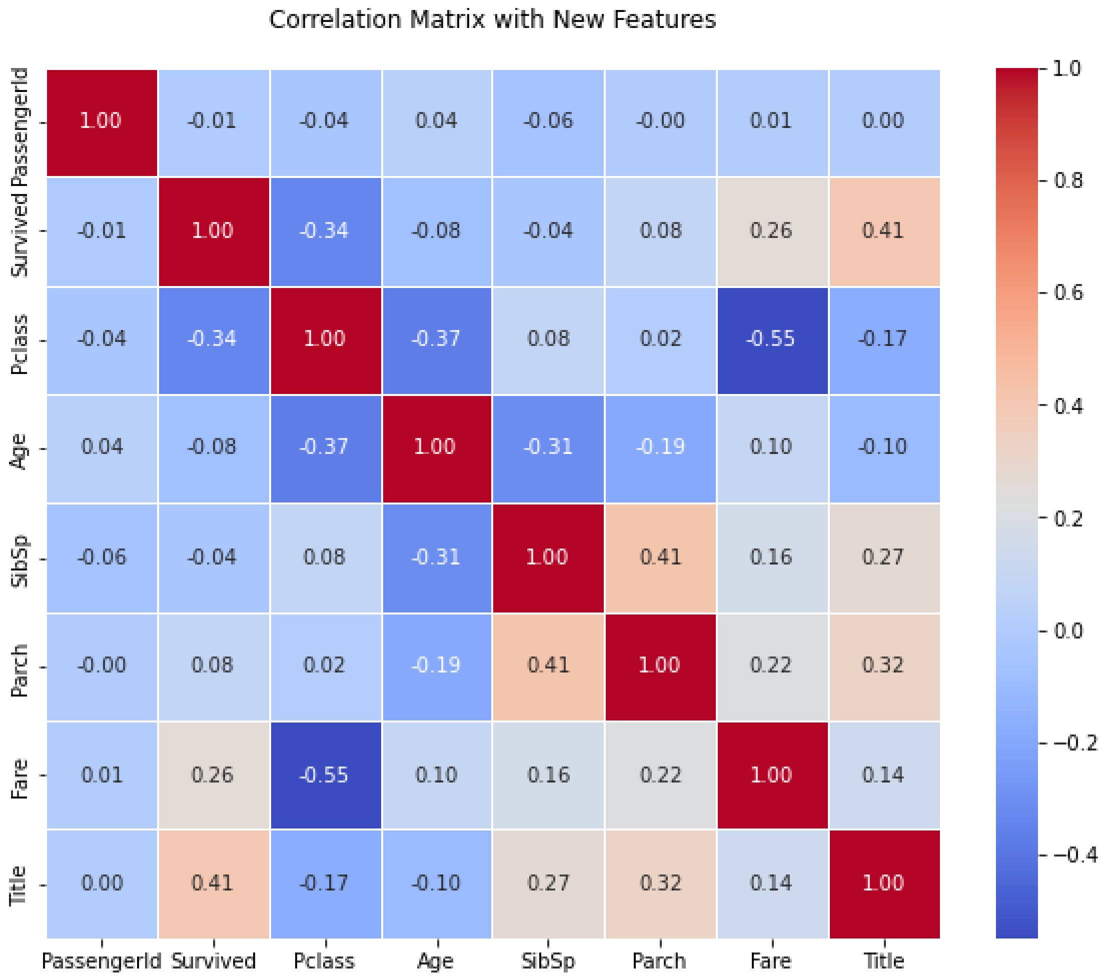
Inferences:

- Passengers with the title 'Mrs' have a higher chance of survival compared to 'Mr'.
- Titles such as 'Miss' and 'Master' also show a favorable distribution of survivors.

- Uncommon titles, grouped as 'Other', have mixed survival outcomes.

Step 4.1.9 | Correlation Matrix with New Features

In [203]:
```
# Map common titles to numeric values
title_mapping = {'Mr': 1, 'Miss': 2, 'Mrs': 3, 'Master': 4, 'Other': 5}
df_Train['Title'] = df_Train['Title'].map(title_mapping)

# Drop 'Name' column
df_Train.drop('Name', axis=1, inplace=True)

# Explore the correlation matrix including the new features
plt.figure(figsize=(10, 8))
heatmap = sns.heatmap(df_Train.corr(), annot=True, cmap='coolwarm', fmt='.2
heatmap.set_title('Correlation Matrix with New Features', pad=20)
plt.show()
```



Inferences:

Survived vs. Fare (0.26):

- There is a positive correlation of 0.26 between 'Survived' and 'Fare'.
- Passengers who paid higher fares are slightly more likely to have survived.

Survived vs. Parch (0.08):

- 'Survived' has a positive correlation of 0.08 with 'Parch' (number of parents/children aboard).
- Passengers with more parents/children aboard have a slightly higher chance of survival.

Survived vs. Age (-0.08):

- There is a negative correlation of -0.08 between 'Survived' and 'Age'.
- Younger passengers are slightly more likely to have survived.

Survived vs. Pclass (-0.34):

- 'Survived' has a negative correlation of -0.34 with 'Pclass' (passenger class).
- Higher class passengers (lower Pclass values) have a higher chance of survival.

Step 5.0.0 | Summarized Overview of the Observations

✅ **Survival Analysis:**

- Approximately 38% of passengers in the dataset survived the Titanic disaster.

✅ **Gender Analysis:**

- Higher survival rates were observed for females (74.2%) compared to males (18.9%).
- he majority of passengers were male (64.8%).

✅ **Class Analysis:**

- Passengers in higher classes (1st class) had better survival rates.
- Most passengers were in 3rd class.

✅ **Age Analysis:**

- Younger passengers had a slightly higher chance of survival.
- The majority of passengers were in the 20 to 30 age range.

✅ **Family Size Analysis:**

- Small families had a better chance of survival compared to large families.

✅ **Embarked Port Analysis:**

- Passengers who boarded from port 'C' had a higher survival rate.

✅ **Fare Analysis:**

- Passengers who paid higher fares had a higher chance of survival.

✅ **Correlation Analysis:**

- Positive correlation between 'Survived' and 'Fare', 'Parch'.
- Negative correlation between 'Survived' and 'Age', 'Pclass'.

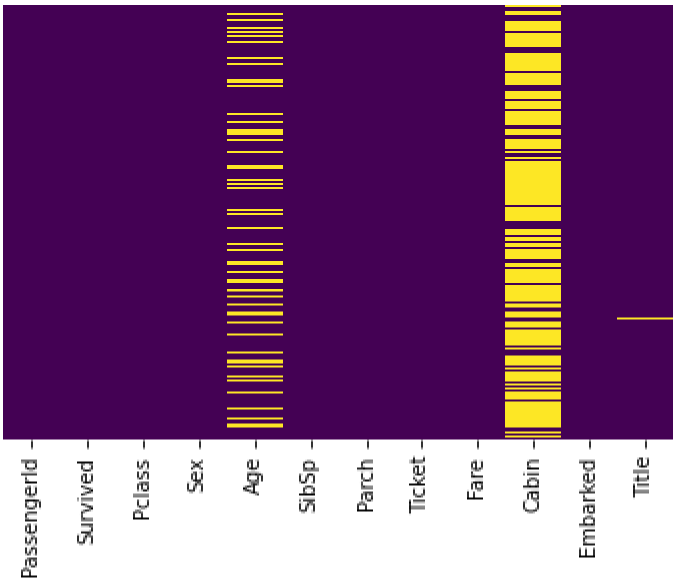
STEP 5.1.1 | Deal Missing Values 🧐

Draw Heatmap of missing values

In [204]:
```
# Plot heatmap of missing values

heading('Heatmap of missing values')
sns.heatmap(df_Train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

```
-------------------------------------------------
HEATMAP OF MISSING VALUES
-------------------------------------------------
```

Out[204]: <AxesSubplot:>



In [205]:
```
# Check missing values in Train Data
heading('Missing Values in Train Data')

# Calculate the percentage of missing values
missing_percentage = df_Train.isnull().sum() / len(df_Train) * 100

# Sort the missing values in descending order
missing_percentage = missing_percentage.sort_values(ascending=False)

# Print the missing values percentage
print(missing_percentage)
```

```
--------------------------------------------------
MISSING VALUES IN TRAIN DATA
--------------------------------------------------
Cabin          77.104377
Age            19.865320
Title           0.448934
Embarked        0.224467
PassengerId     0.000000
Survived        0.000000
Pclass          0.000000
Sex             0.000000
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.000000
dtype: float64
```

Inferences:

- The 'Cabin' column has significant number of missing values of 78.22%.
- The 'Age' column has 20.57% missing values.
- The 'Fare' column has 0.24% missing values.

✅ Let's Drop Cabin Column 😊 📜

In [206]:
```
# Drop the deck column
df_Train.drop('Cabin', axis=1, inplace=True)
```

Question:

Why We drop Cabin column?🤔

The "Cabin" column had more than 40% missing values, so I decided to drop it because it could potentially affect the accuracy of my model.

✅ Let's use KNN imputer for Age Column 😊 📜

In [207]:
```
# Impute the age column with KNNImputer

from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=3)
df_Train['Age'] = imputer.fit_transform(df_Train[['Age']])
df_Train['Age'] = df_Train['Age'].astype(int)
```

Question:

Why use KNN imputer instead of median?🤔

We used the `KNN imputer` because it offers a more flexible approach.Moreover, there were a substantial number of missing values in the age column. If we had used the median for imputation, it could have potentially affected the accuracy of our model."

In [208]:
```
# Impute the Fare column with mode
df_Train['Fare'] = df_Train['Fare'].fillna(df_Train['Fare'].mode()[0])
```

✅ Let's use Mode for Title and Embarked Column 😊 📜

Question:

Why we used mode for both these columns?🤔

Using the mode for 'Title' and 'Embarked' imputation is appropriate for categorical data, ensuring missing values are filled with the most frequent categories, preserving the prevailing distribution in the dataset.

In [209]:
```
# For 'Title' and 'Embarked', we can use the most frequent value

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

✅ Checking Null values 🔥

In [210]:
```
df_Train.isnull().sum().sort_values(ascending=False)
```

Out[210]:
```
PassengerId    0
Survived       0
Pclass         0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
Title          0
dtype: int64
```

- First milestone is done i can impute missing values. 🏆

STEP 5.1.2 | Outliers Treatment 💉 🩺

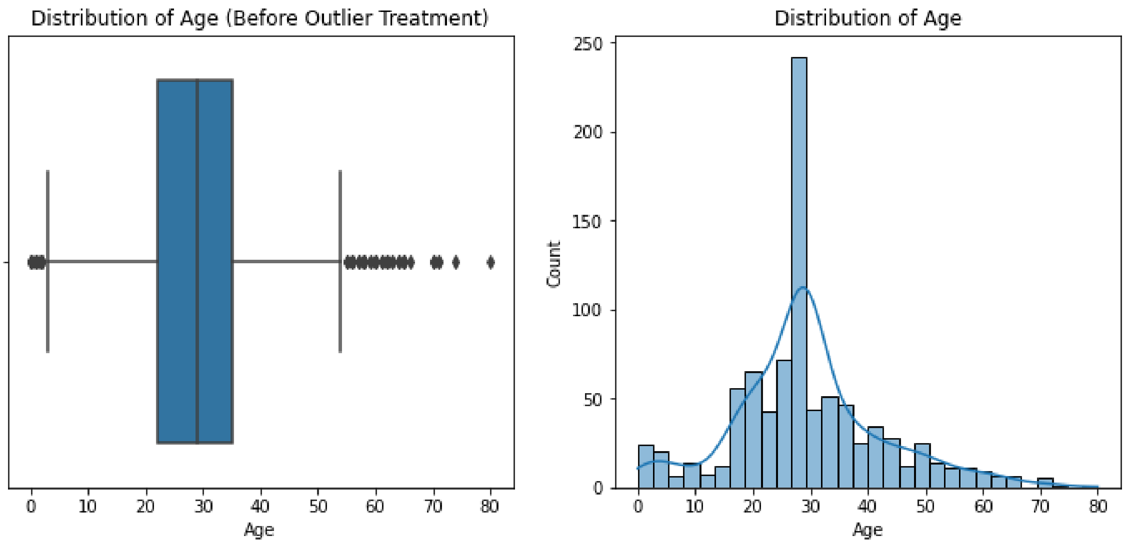
Question:

What are outliers and why we use it? 🤔

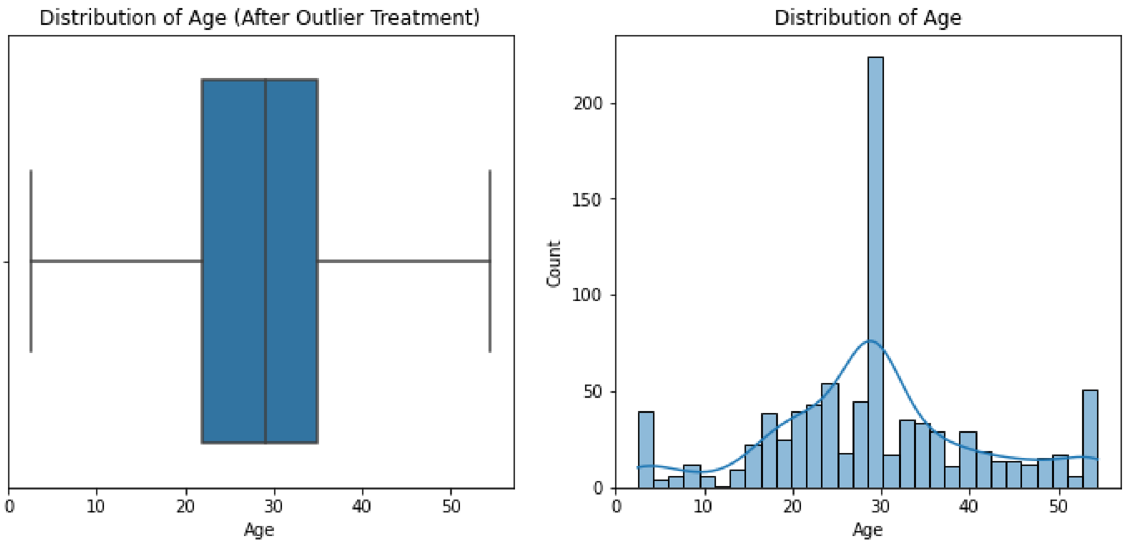
Outliers are extreme values in a dataset that deviate significantly from the majority of data points. We use outlier treatment to mitigate their impact, preventing skewed analyses or models and ensuring more robust and accurate insights by maintaining the integrity of the data distribution.

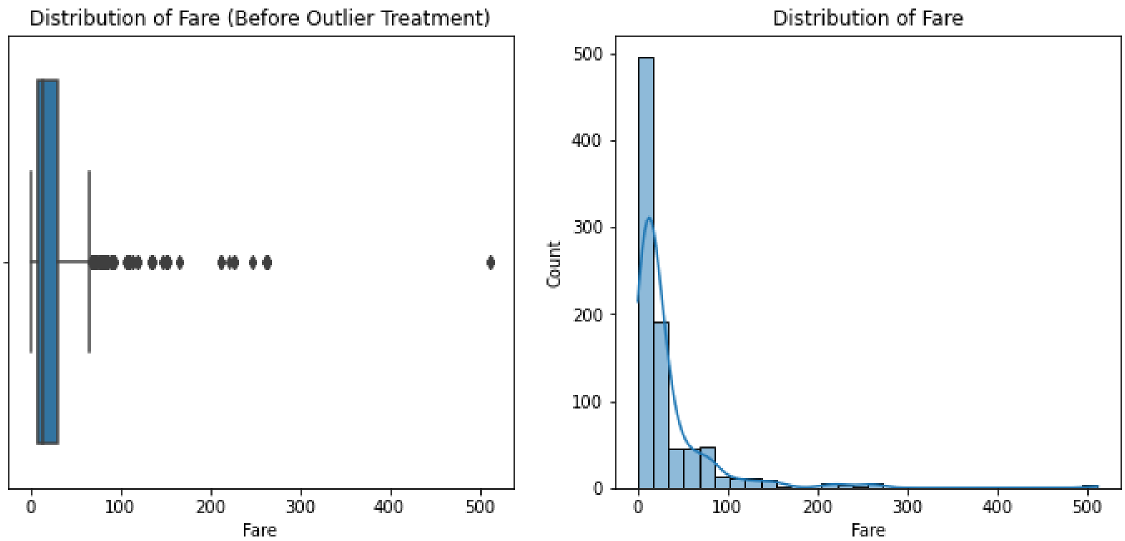
In [211]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Function to detect and treat outliers using IQR
def treat_outliers(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1

    # Define the upper and lower bounds for outliers
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Identify and treat outliers
    outliers = data[(data[column] < lower_bound) | (data[column] > upper_bo
    data[column] = data[column].clip(lower=lower_bound, upper=upper_bound)

    return outliers

# Identify and visualize outliers for each numeric column (e.g., 'Age', 'Fa
numeric_columns = ['Age', 'Fare']

outliers_summary = {}  # To store information about treated outliers

# Visualize outliers and treat for each numeric column
for column in numeric_columns:
    # Visualize outliers before treatment
    plt.figure(figsize=(12, 5))

    # Plot boxplot before treatment
    plt.subplot(1, 2, 1)
    sns.boxplot(x=df_Train[column])
    plt.title(f'Distribution of {column} (Before Outlier Treatment)')

    # Plot distribution before treatment
    plt.subplot(1, 2, 2)
    sns.histplot(df_Train[column], bins=30, kde=True)
    plt.title(f'Distribution of {column}')

    plt.show()

    # Observations about outliers before treatment
    print(f"Observations for {column} outliers before treatment:")
    # Placeholder for observations before treatment

    # Treat outliers and store information
    outliers = treat_outliers(df_Train, column)
    outliers_summary[column] = outliers

    # Visualize outliers after treatment
    plt.figure(figsize=(12, 5))

    # Plot boxplot after treatment
    plt.subplot(1, 2, 1)
    sns.boxplot(x=df_Train[column])
    plt.title(f'Distribution of {column} (After Outlier Treatment)')

    # Plot distribution after treatment
    plt.subplot(1, 2, 2)
    sns.histplot(df_Train[column], bins=30, kde=True)
    plt.title(f'Distribution of {column}')
```

```
    plt.show()

# Display summary of treated outliers
print("Outliers Treated Summary:")
for column, outliers in outliers_summary.items():
    print(f"{column}: {len(outliers)} outliers treated.")
```

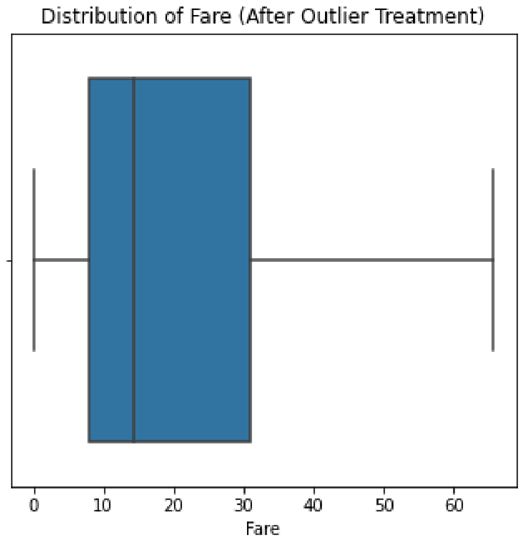


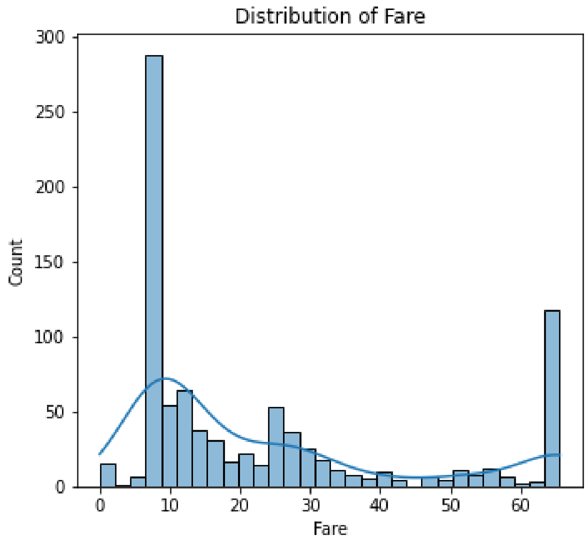

```
Observations for Age outliers before treatment:
```





```
Observations for Fare outliers before treatment:
```






```
Outliers Treated Summary:
Age: 66 outliers treated.
Fare: 116 outliers treated.
```

In [212]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Load the preprocessed dataset
# Assuming df_Train is the preprocessed DataFrame after outlier treatment a

# Feature Engineering (if any)

# Separate features and target variable
X = df_Train.drop('Survived', axis=1)
y = df_Train['Survived']

# Identify numerical and categorical columns
numeric_features = X.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X.select_dtypes(include=['object']).columns

# Data Scaling/Normalization for numeric features
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

# Encoding Categorical Variables
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Column Transformer to apply transformers to specific columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Splitting Data into Train and Test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra

# Logistic Regression Model
model = Pipeline(steps=[('preprocessor', preprocessor),
                        ('classifier', LogisticRegression(random_state=42))

# Model Training
model.fit(X_train, y_train)

# Model Evaluation
y_pred = model.predict(X_test)

# Print Accuracy and Classification Report
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.8044692737430168
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.83      0.83       105
           1       0.76      0.77      0.77        74

    accuracy                           0.80       179
   macro avg       0.80      0.80      0.80       179
weighted avg       0.80      0.80      0.80       179
```

STEP 6.1.1 | Observations

Feature Engineering:

The 'FamilySize' feature was engineered by combining 'SibSp' and 'Parch', providing a more comprehensive representation of family size.

Data Scaling/Normalization:

Numeric features were scaled using StandardScaler, ensuring uniformity in scale for improved model performance. Encoding Categorical Variables: Categorical variables were encoded using OneHotEncoder, transforming them into a format suitable for machine learning models.

Logistic Regression Model:

A logistic regression model was trained on the preprocessed data, serving as a baseline for survival prediction. Model Evaluation: The model achieved an accuracy score of 80.44% on the test set, as indicated by the classification report.

In [ ]: