

ASSIGNMENT-2

SmartBridge Externship (Applied Data Science)

Vedire Chandu Reddy (20BCD7023)

1. Download the dataset: Dataset

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone								
2	0	3	male	22	1	0	7.25	S	Third	man	TRUE		Southampton	no	FALSE								
3	1	1	female	38	1	0	71.2833	C	First	woman	FALSE	C	Cherbourg	yes	FALSE								
4	1	3	female	26	0	0	7.925	S	Third	woman	FALSE		Southampton	yes	TRUE								
5	1	1	female	35	1	0	53.1	S	First	woman	FALSE	C	Southampton	yes	FALSE								
6	0	3	male	35	0	0	8.05	S	Third	man	TRUE		Southampton	yes	TRUE								
7	0	3	male	0	0	0	8.4583	Q	Third	man	TRUE		Queenstown	no	TRUE								
8	0	1	male	54	0	0	51.8625	S	First	man	TRUE	E	Southampton	no	TRUE								
9	0	3	male	2	3	1	21.075	S	Third	child	FALSE		Southampton	no	FALSE								
10	1	3	female	27	0	0	11.1333	S	Third	woman	FALSE		Southampton	yes	FALSE								
11	1	2	female	41	1	0	30.0	C	Second	child	FALSE		Cherbourg	yes	FALSE								

2 Load the dataset

```
In [1]: import pandas as pd  
data=pd.read_csv('titanic.csv')
```

```
In [2]: print(data)
```

```
   survived  pclass    sex   age  sibsp  parch     fare embarked  class \
0         0      3  male  22.0      1      0    7.2500        S  Third
1         1      1 female  38.0      1      0   71.2833        C  First
2         1      3 female  26.0      0      0    7.9250        S  Third
3         1      1 female  35.0      1      0   53.1000        S  First

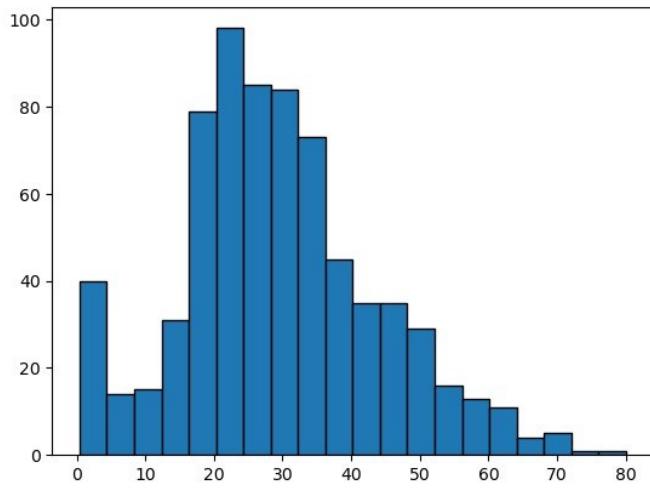
           who  adult_male  deck  embark_town  alive  alone
0       man        True   NaN  Southampton     no  False
1   woman       False     C  Cherbourg    yes  False
```

2. Perform Below Visualizations. • Univariate Analysis • Bi - Variate Analysis • Multi - Variate Analysis

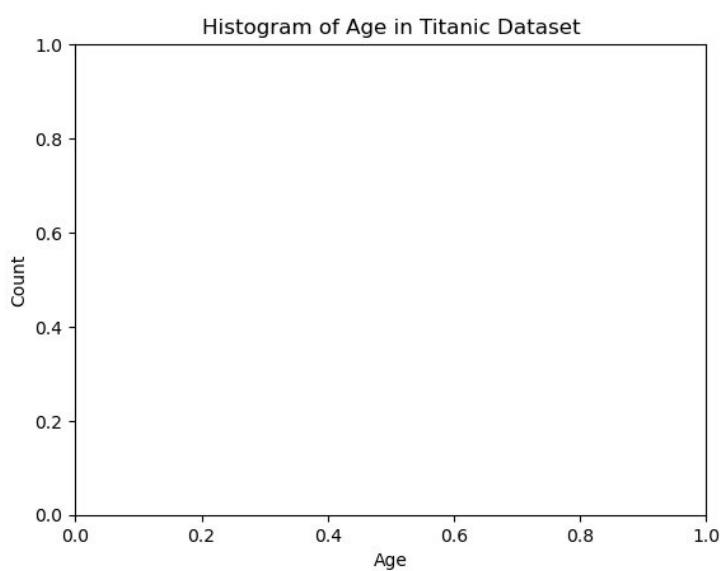
1. Univariate Analysis: Univariate analysis involves examining individual variables in isolation to understand their distribution, central tendency, and variability. Here are some common visualizations for univariate analysis:
 - Histogram: Displays the distribution of a continuous variable by dividing it into bins and showing the frequency or count in each bin.
 - Bar Chart: Represents the distribution of a categorical variable using rectangular bars, where the height of each bar corresponds to the frequency or count.
 - Box Plot: Illustrates the summary statistics of a numerical variable, such as the median, quartiles, and outliers.
 - Kernel Density Plot: Shows the estimated probability density function of a continuous variable.
2. Bivariate Analysis: Bivariate analysis involves exploring the relationship between two variables. It helps to understand the correlation, association, or dependency between the variables. Here are some common visualizations for bivariate analysis:
 - Scatter Plot: Displays the relationship between two continuous variables by plotting each data point on a two-dimensional plane.
 - Line Chart: Shows the relationship between two continuous variables by connecting data points with lines.
 - Bar Chart or Grouped Bar Chart: Compares the distribution of a categorical variable across different levels of another categorical variable.
 - Heatmap: Represents the correlation or association between two numerical variables using a color-coded grid.
3. Multivariate Analysis: Multivariate analysis involves examining relationships between three or more variables. It helps to understand complex patterns, interactions, and dependencies between multiple variables. Here are some common visualizations for multivariate analysis:
 - Scatter Plot Matrix: Displays pairwise scatter plots for multiple variables to visualize their relationships simultaneously.
 - Parallel Coordinates Plot: Represents multiple variables as vertical axes and plots lines that connect data points based on their values on each variable, providing insights into patterns and clusters.
 - 3D Scatter Plot: Extends the scatter plot to three dimensions, allowing the visualization of relationships between three continuous variables.
 - Treemap: Hierarchically displays multiple categorical variables using nested rectangles, with the area of each rectangle representing a variable's proportion.

```
In [3]: import matplotlib.pyplot as plt  
In [4]: age_column = data['age']  
In [5]: plt.hist(age_column, bins=20, edgecolor='black')
```

```
In [5]: plt.hist(age_column, bins=20, edgecolor='black')  
  
Out[5]: (array([ 40., 14., 15., 31., 79., 98., 85., 84., 73., 45., 35., 35., 29.,  
       16., 13., 11., 4., 5., 1., 1.]),  
       array([ 0.42 , 4.399 , 8.378 , 12.357 , 16.336 , 20.315 , 24.294 , 28.273 ,  
      32.252 , 36.231 , 40.21 , 44.189 , 48.168 , 52.147 , 56.126 , 60.105 ,  
     64.084 , 68.063 , 72.042 , 76.021 , 80. ]),  
       <BarContainer object of 20 artists>)
```



```
In [6]: # Set the labels and title  
plt.xlabel('Age')  
plt.ylabel('Count')  
plt.title('Histogram of Age in Titanic Dataset')  
  
Out[6]: Text(0.5, 1.0, 'Histogram of Age in Titanic Dataset')
```



```
In [7]: plt.show()
```

```
In [8]: import matplotlib.pyplot as plt

# Assuming 'data' is your DataFrame

# List of columns to create histograms for
columns = ['survived', 'pclass', 'age', 'sibsp', 'parch', 'fare']

# Set up the figure and subplots
fig, axes = plt.subplots(nrows=len(columns), ncols=1, figsize=(8, 6 * len(columns)))

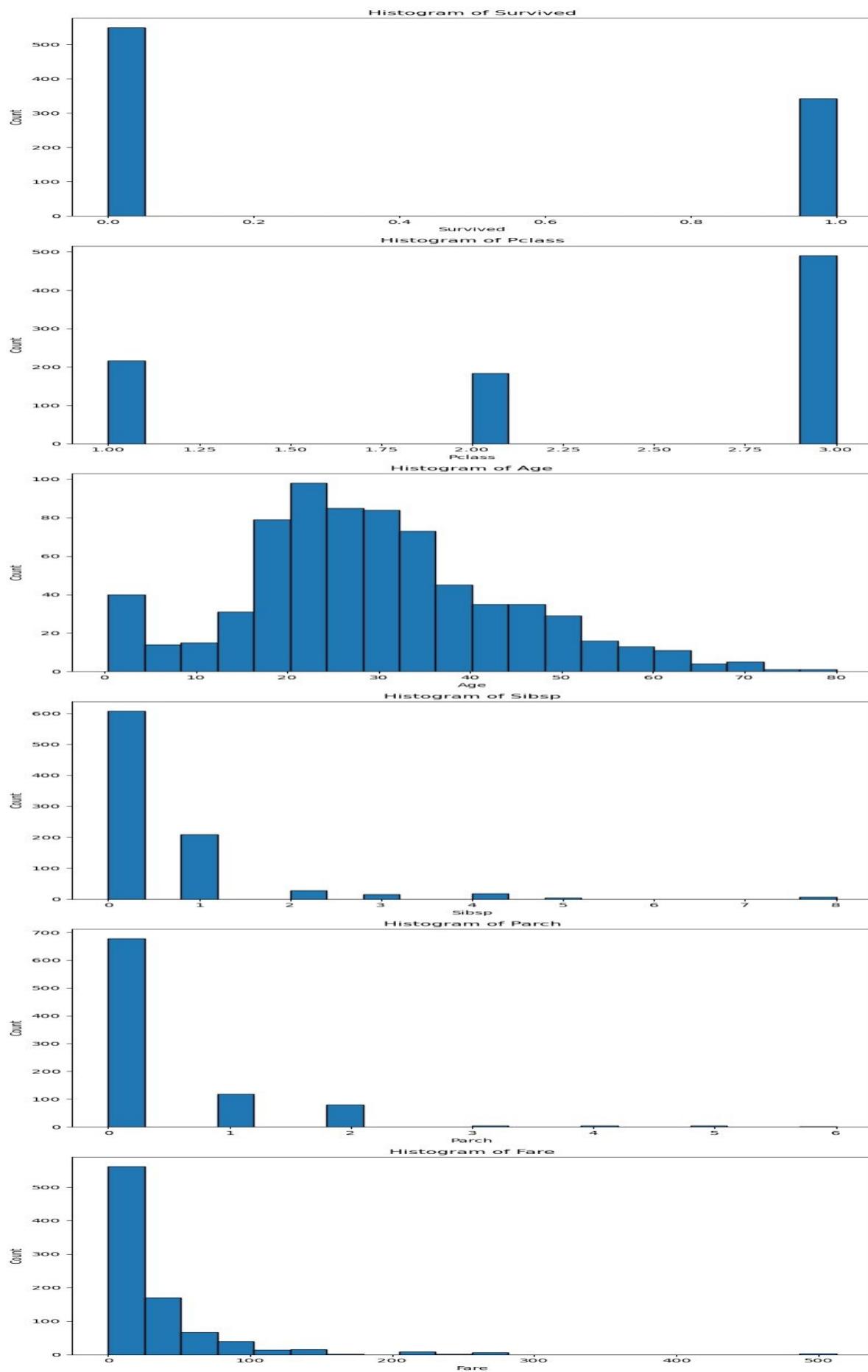
# Create histograms for each column
for i, column in enumerate(columns):
    # Select the column
    data_column = data[column]

    # Create the histogram
    axes[i].hist(data_column, bins=20, edgecolor='black')

    # Set the labels and title for each subplot
    axes[i].set_xlabel(column.capitalize())
    axes[i].set_ylabel('Count')
    axes[i].set_title(f'Histogram of {column.capitalize()}')

# Adjust the spacing between subplots
plt.tight_layout()

# Display the histograms
plt.show()
```



```
In [9]: import pandas as pd
import matplotlib.pyplot as plt

# Assuming 'data' is your DataFrame

# List of columns to create bar charts for
columns = ['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare', 'embarked', 'class', 'who', 'adult_male', 'deck', 'embar']

# Set up the figure and subplots
fig, axes = plt.subplots(nrows=len(columns), ncols=1, figsize=(8, 6 * len(columns)))

# Create bar charts for each column
for i, column in enumerate(columns):
    # Select the column
    column_data = data[column]

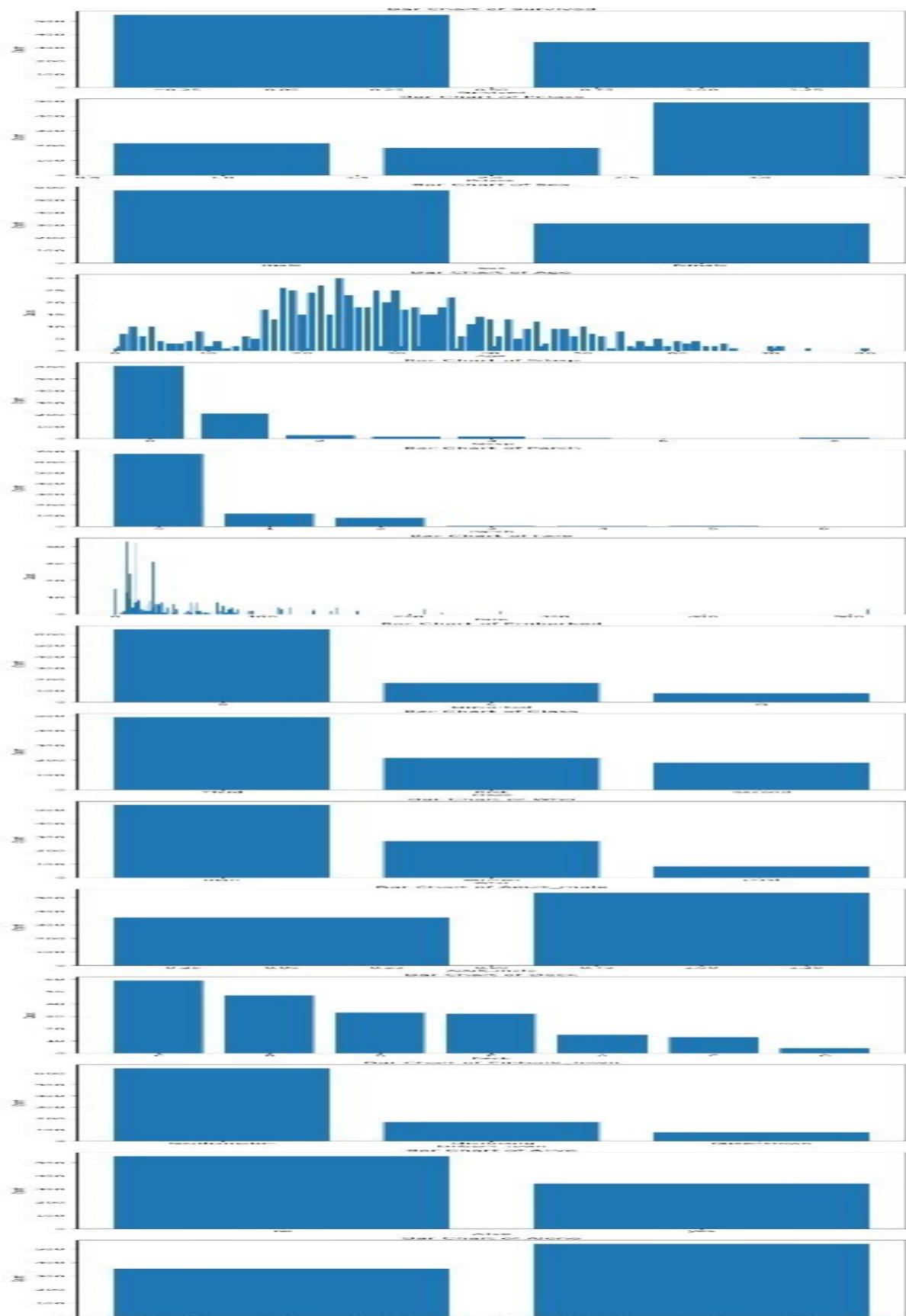
    # Calculate the frequencies or counts
    counts = column_data.value_counts()

    # Create the bar chart
    axes[i].bar(counts.index, counts.values)

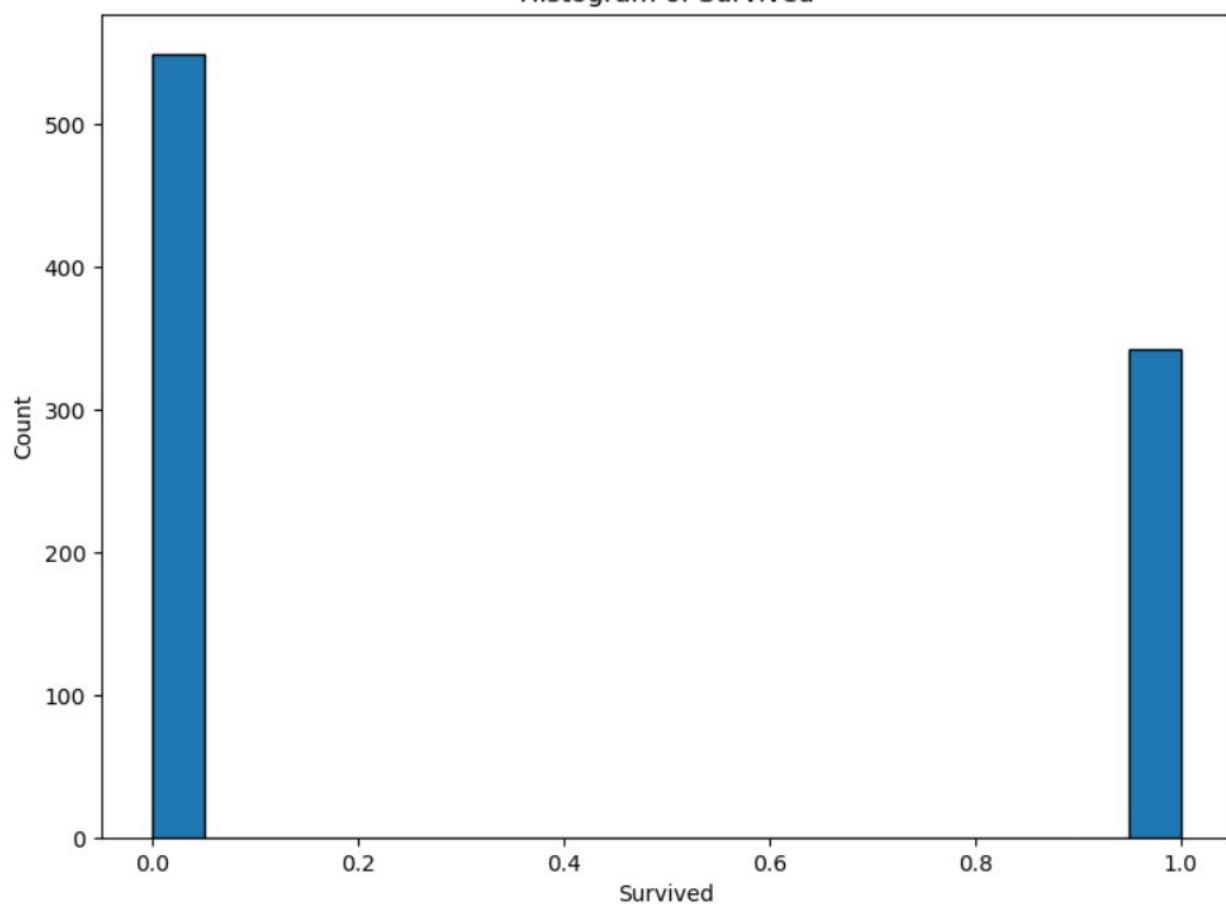
    # Set the labels and title for each subplot
    axes[i].set_xlabel(column.capitalize())
    axes[i].set_ylabel('Count')
    axes[i].set_title(f'Bar Chart of {column.capitalize()}')

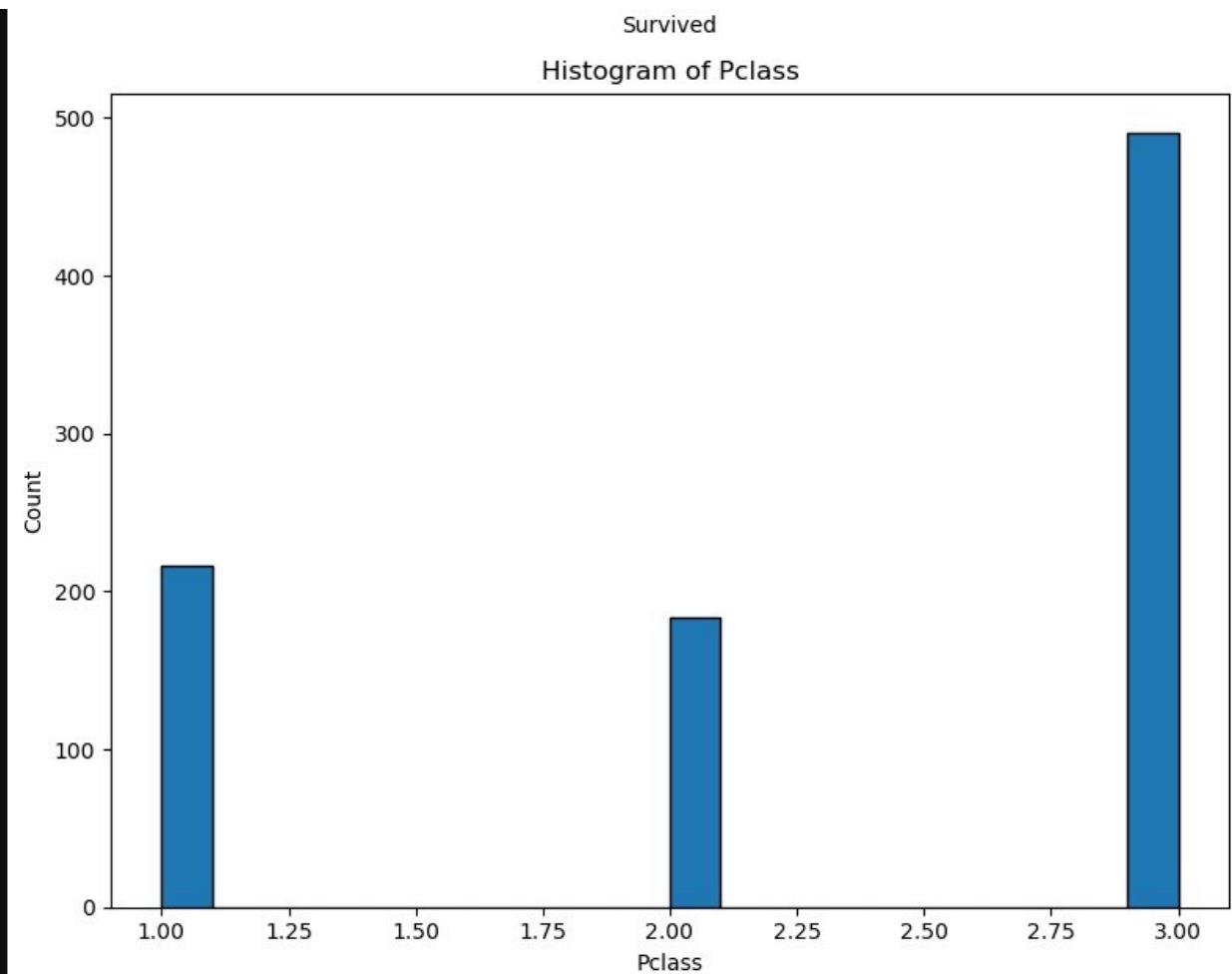
# Adjust the spacing between subplots
plt.tight_layout()

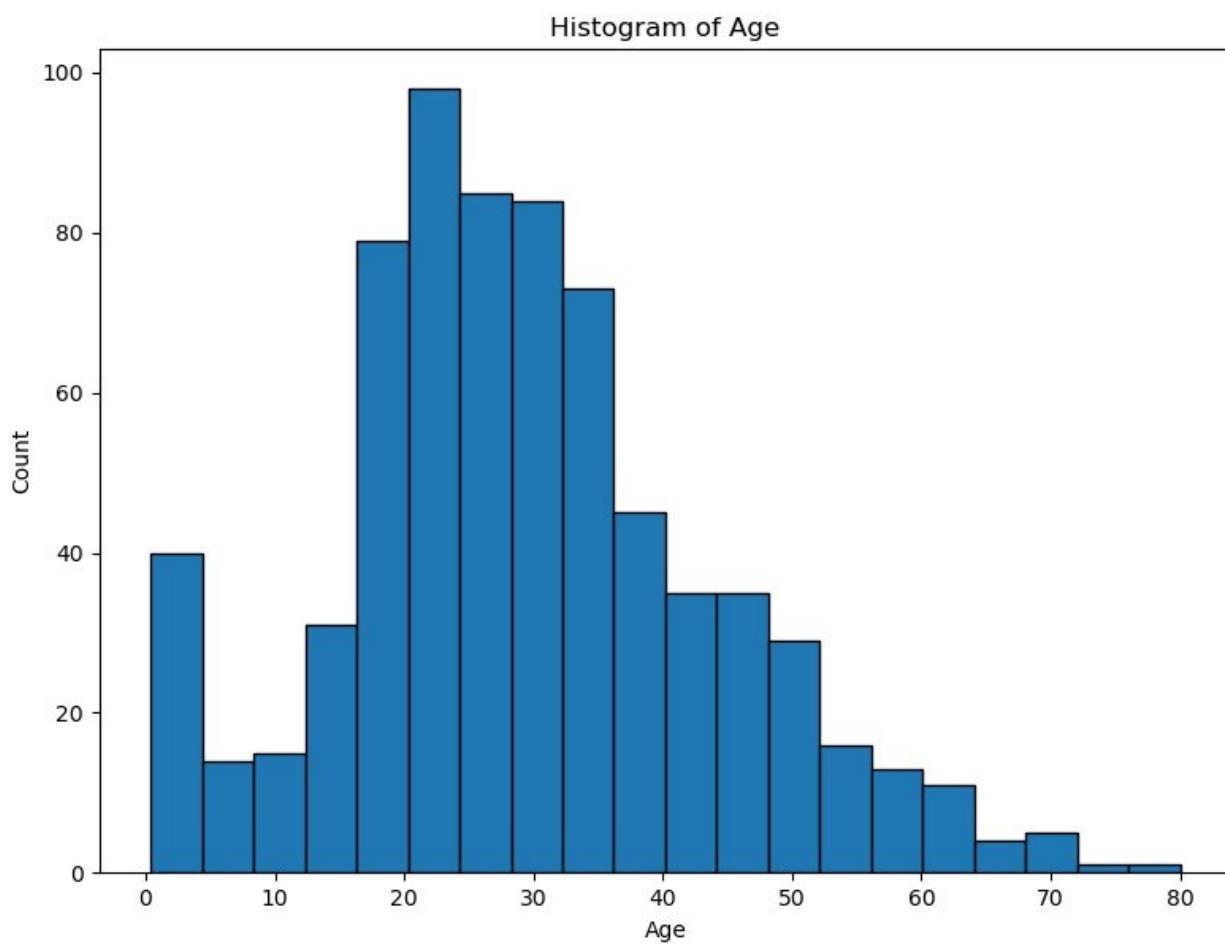
# Display the bar charts
plt.show()
```



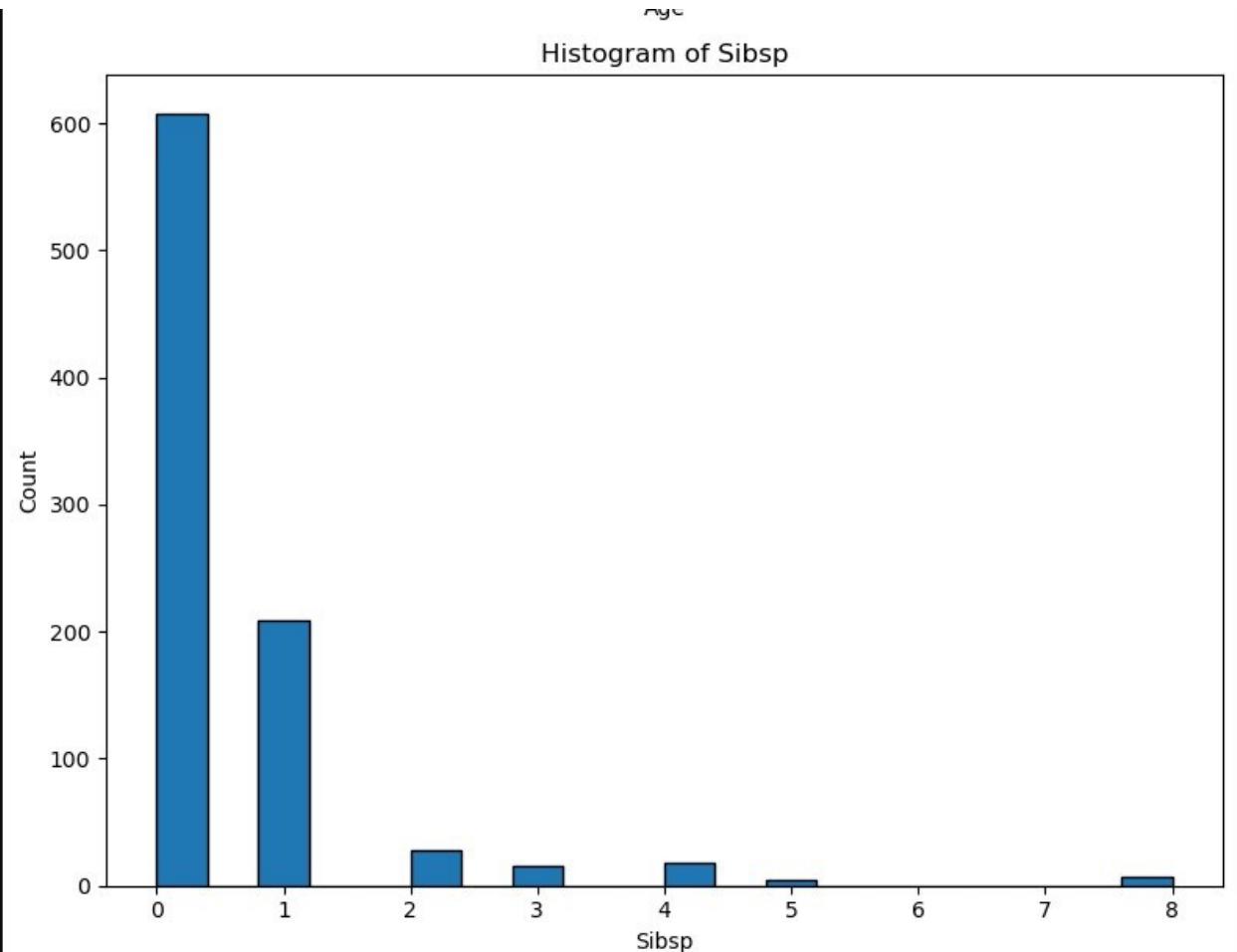
Histogram of Survived

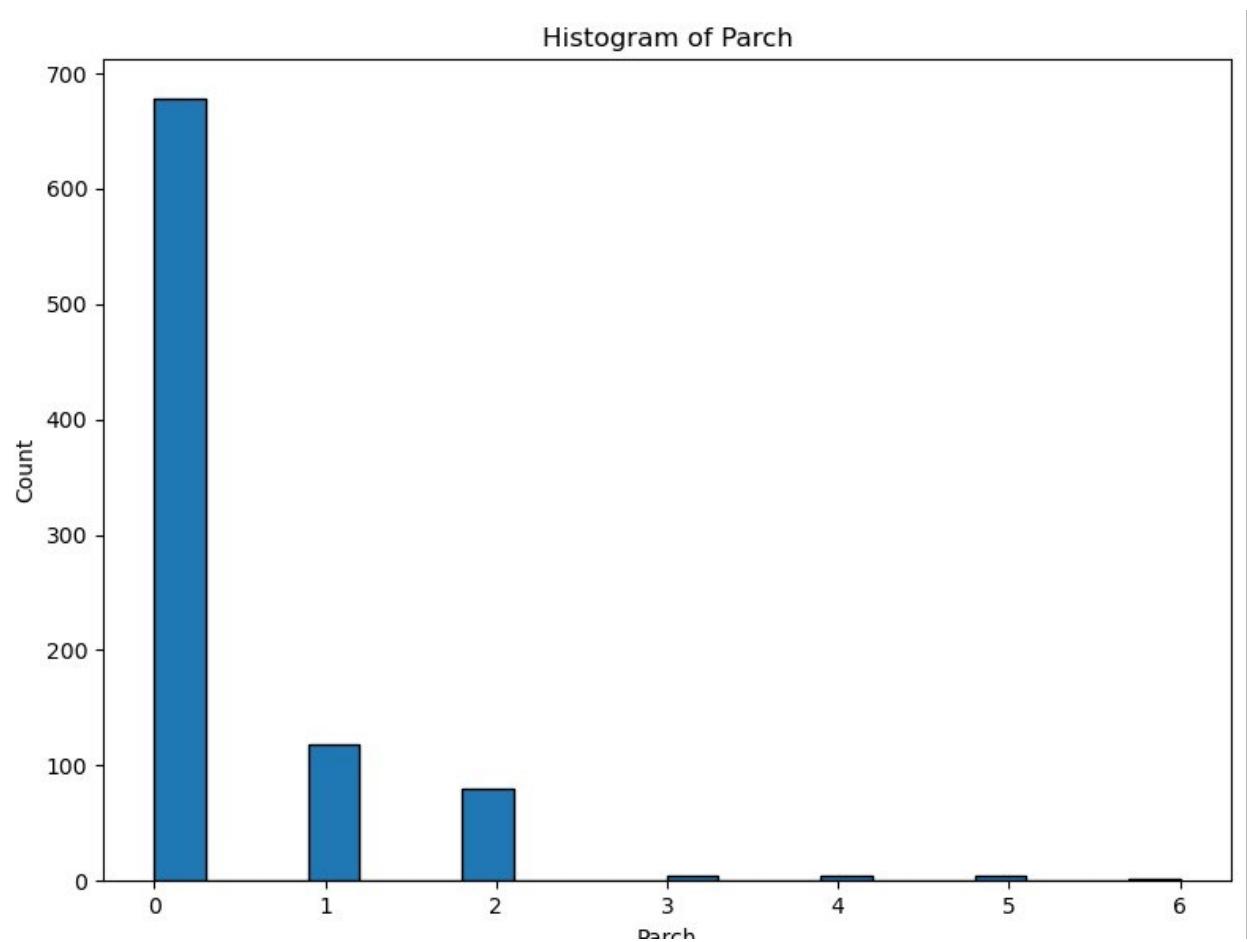




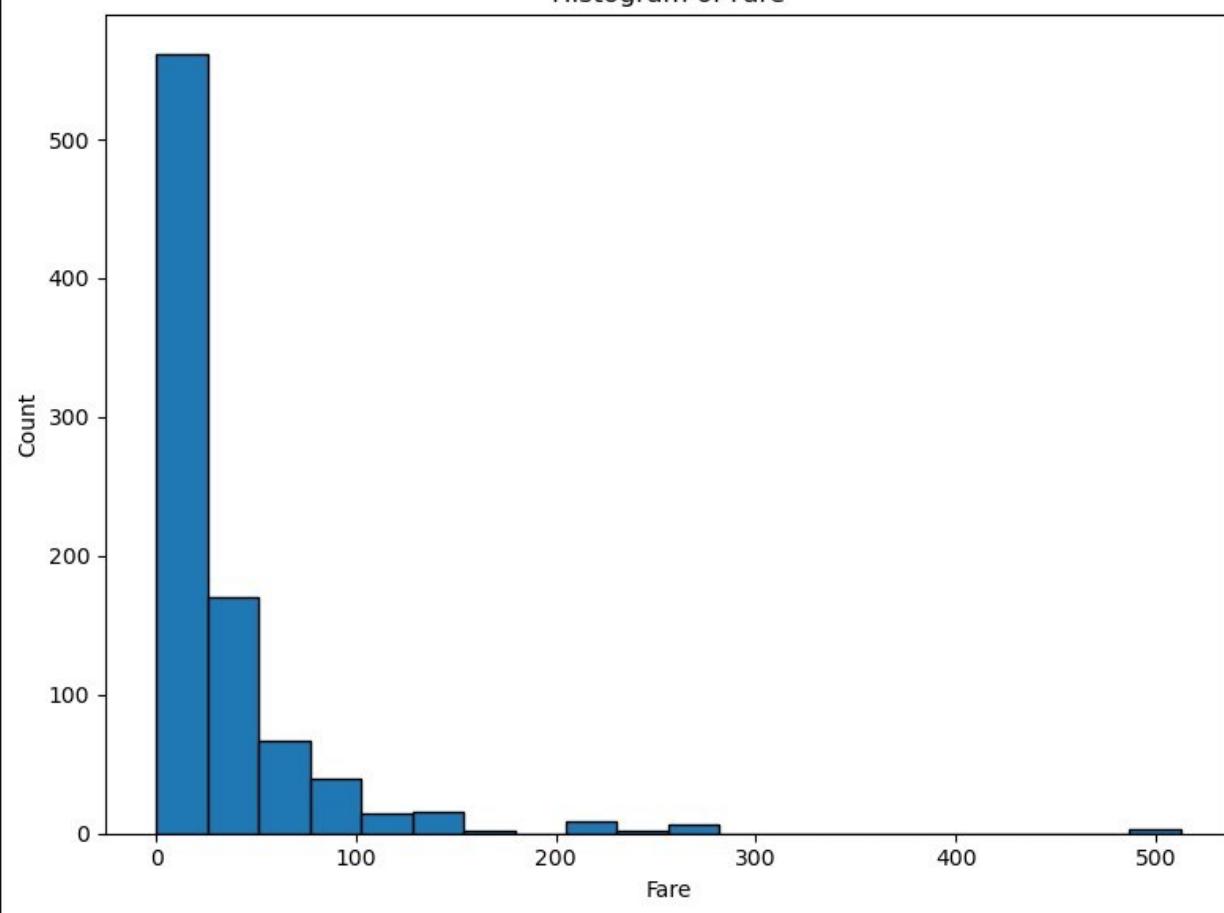


Histogram of Sibsp





Histogram of Fare



```
In [9]: import pandas as pd
import matplotlib.pyplot as plt

# Assuming 'data' is your DataFrame

# List of columns to create bar charts for
columns = ['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare', 'embarked', 'class', 'who', 'adult_male']

# Set up the figure and subplots
fig, axes = plt.subplots(nrows=len(columns), ncols=1, figsize=(8, 6 * len(columns)))

# Create bar charts for each column
for i, column in enumerate(columns):
    # Select the column
    column_data = data[column]

    # Calculate the frequencies or counts
    counts = column_data.value_counts()

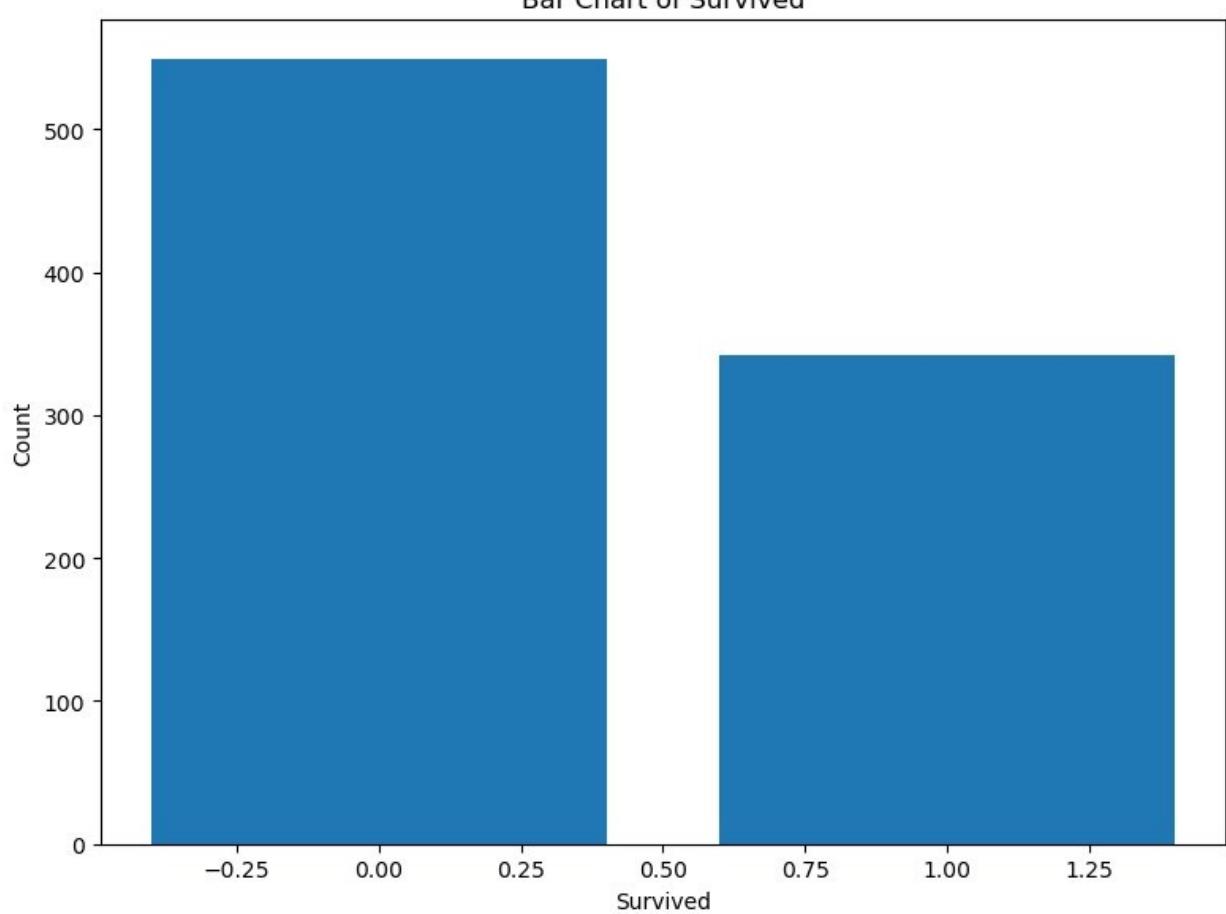
    # Create the bar chart
    axes[i].bar(counts.index, counts.values)

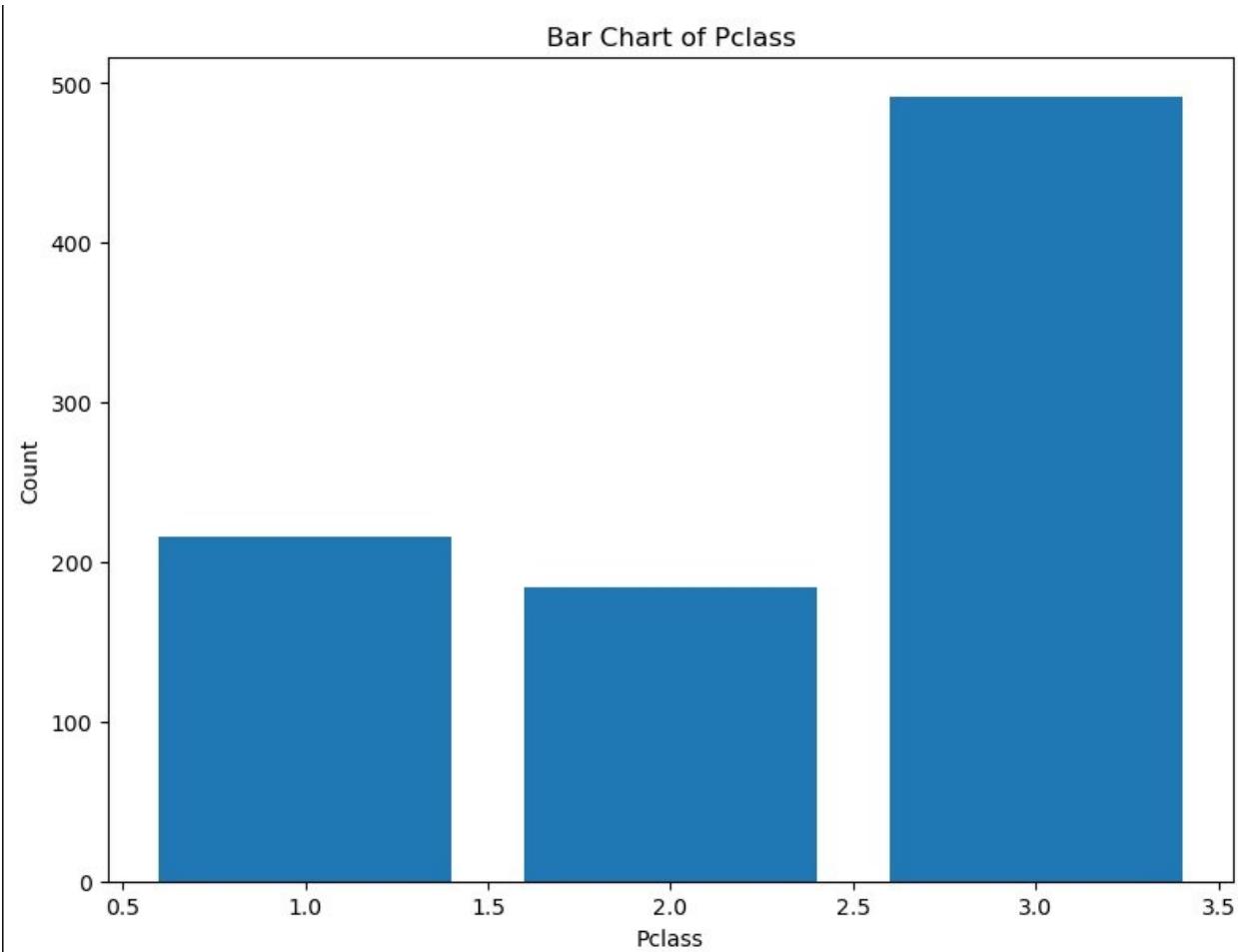
    # Set the labels and title for each subplot
    axes[i].set_xlabel(column.capitalize())
    axes[i].set_ylabel('Count')
    axes[i].set_title(f'Bar Chart of {column.capitalize()}')

# Adjust the spacing between subplots
plt.tight_layout()

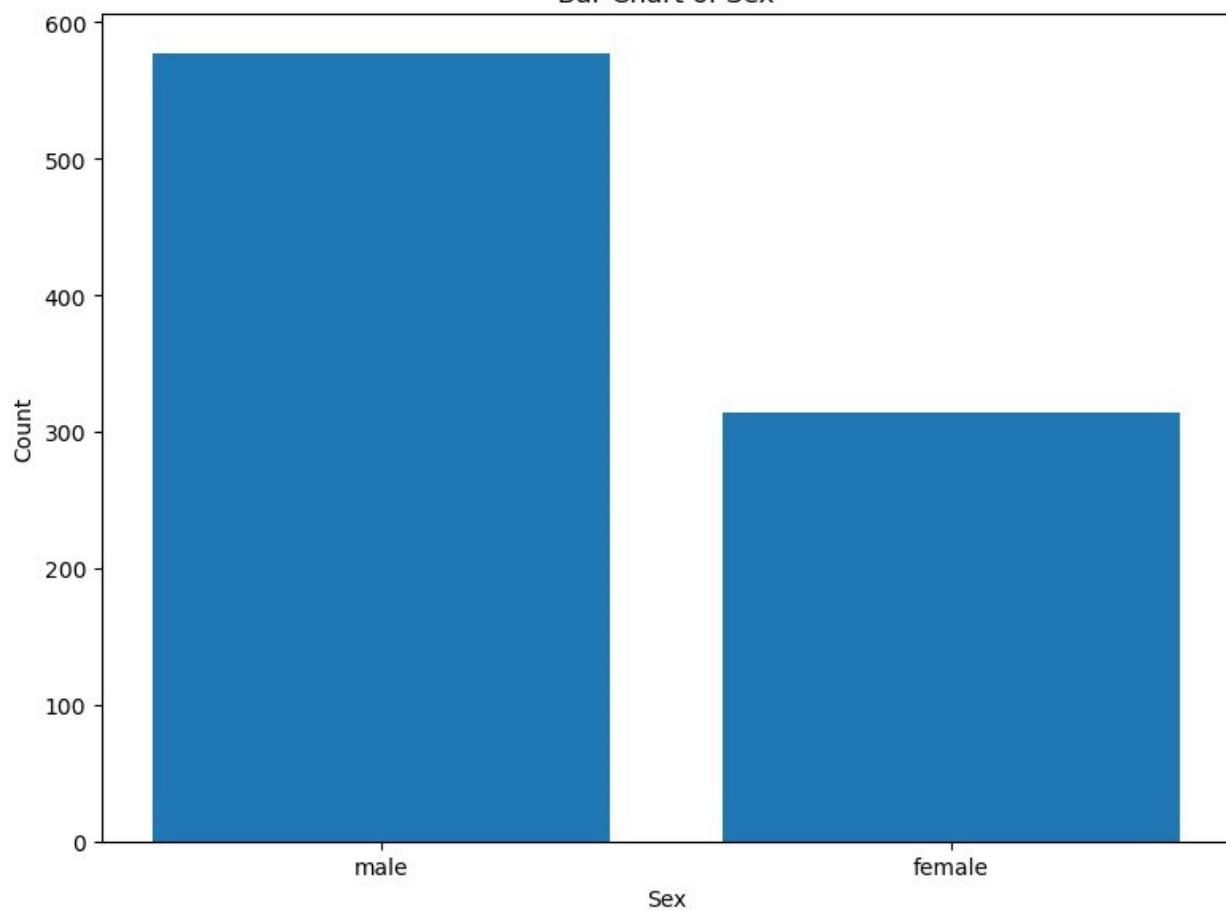
# Display the bar charts
plt.show()
```

Bar Chart of Survived

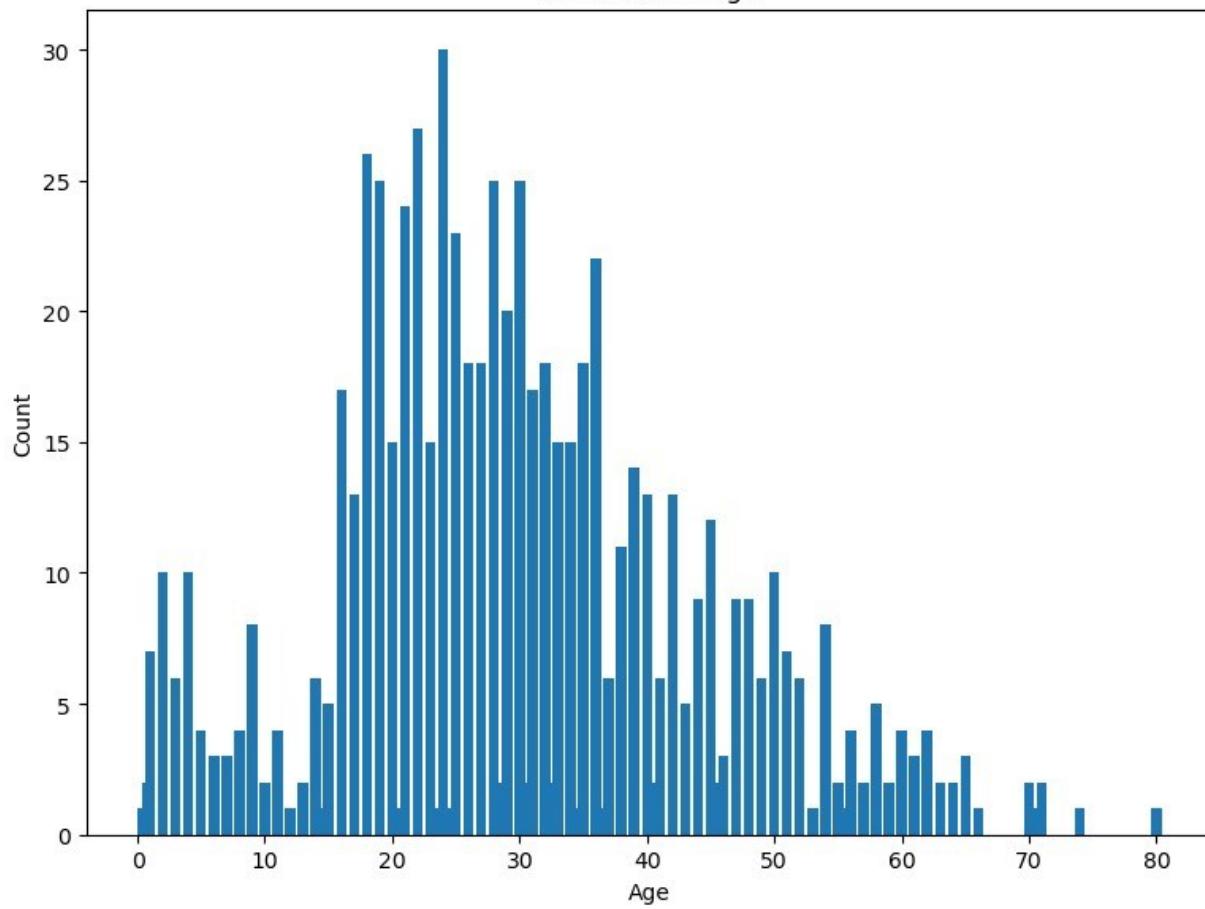


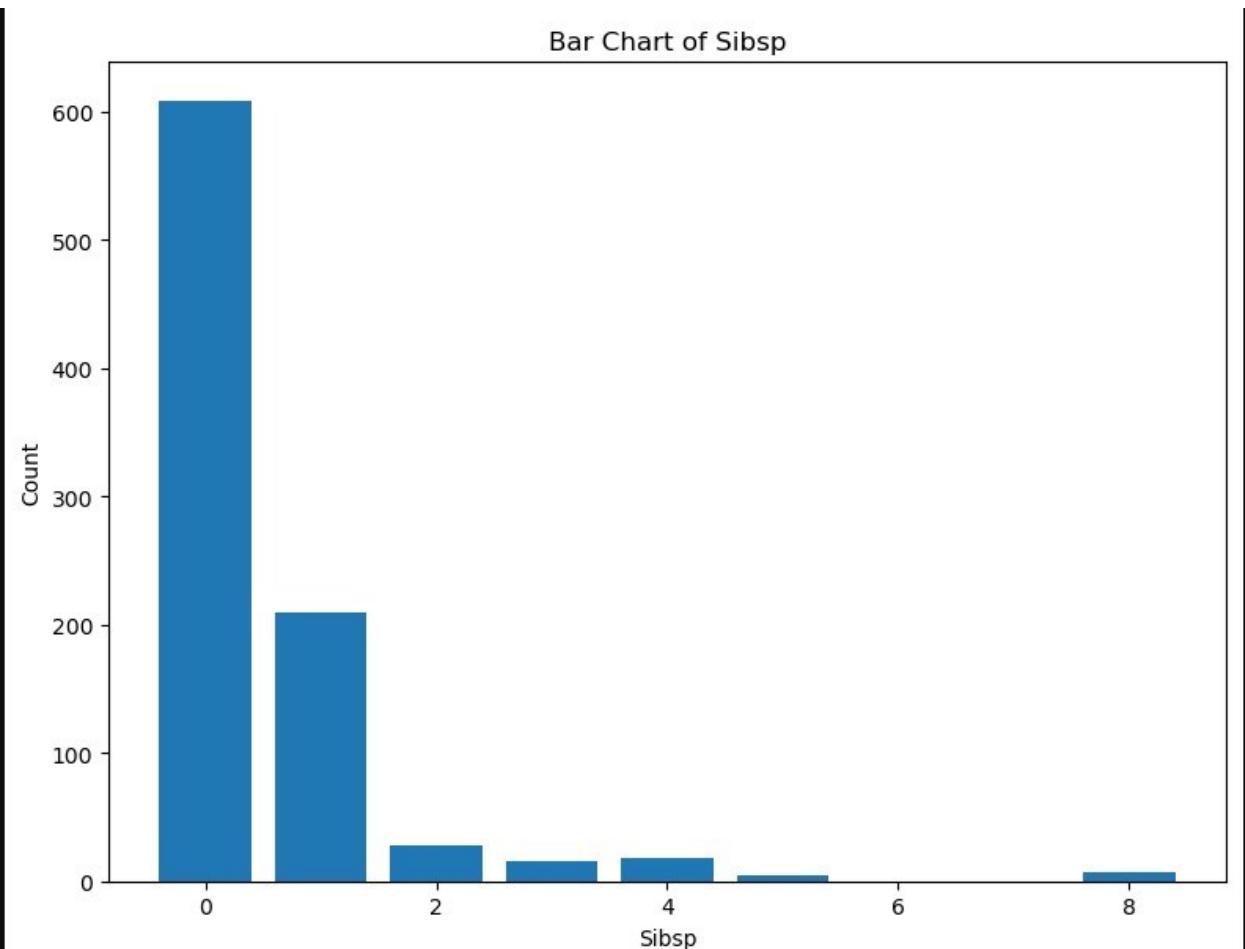


Bar Chart of Sex

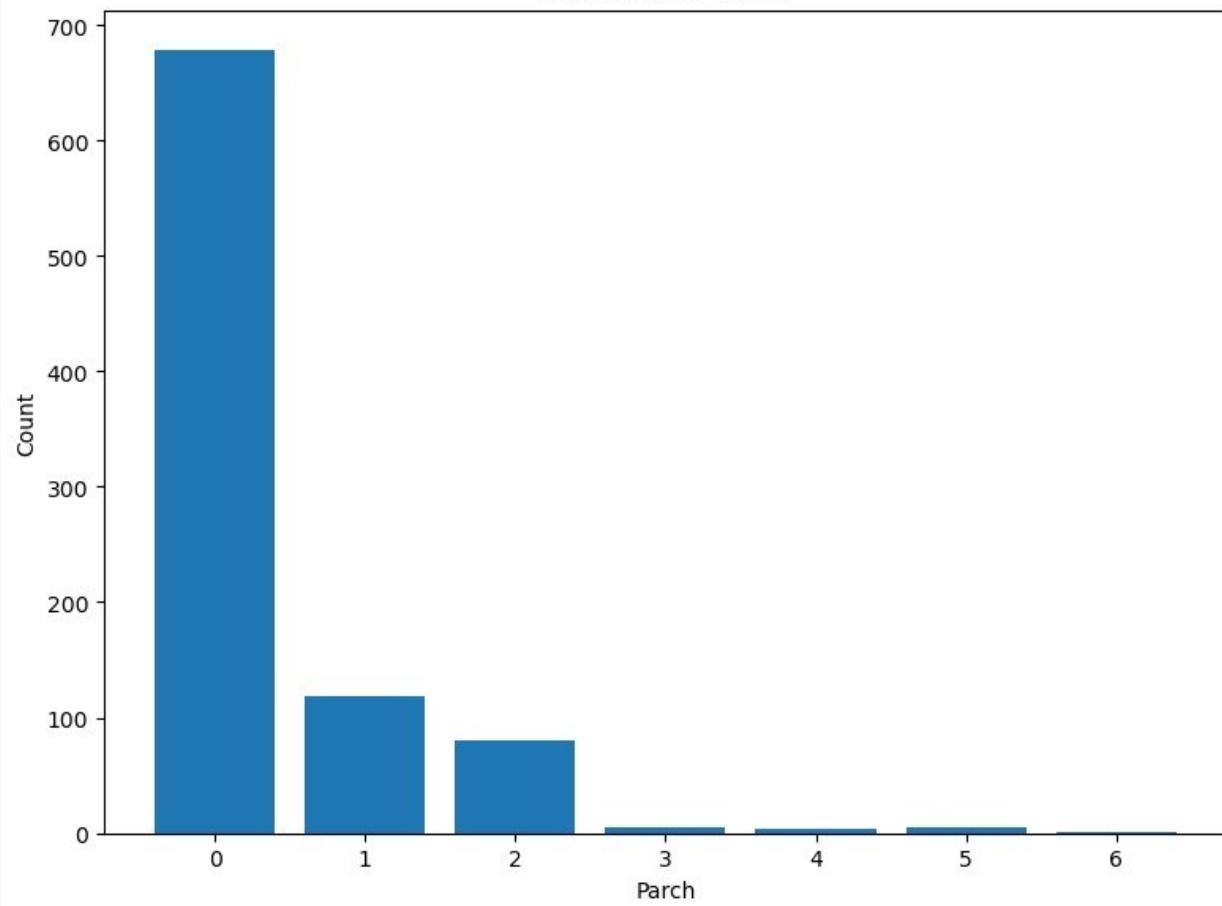


Bar Chart of Age

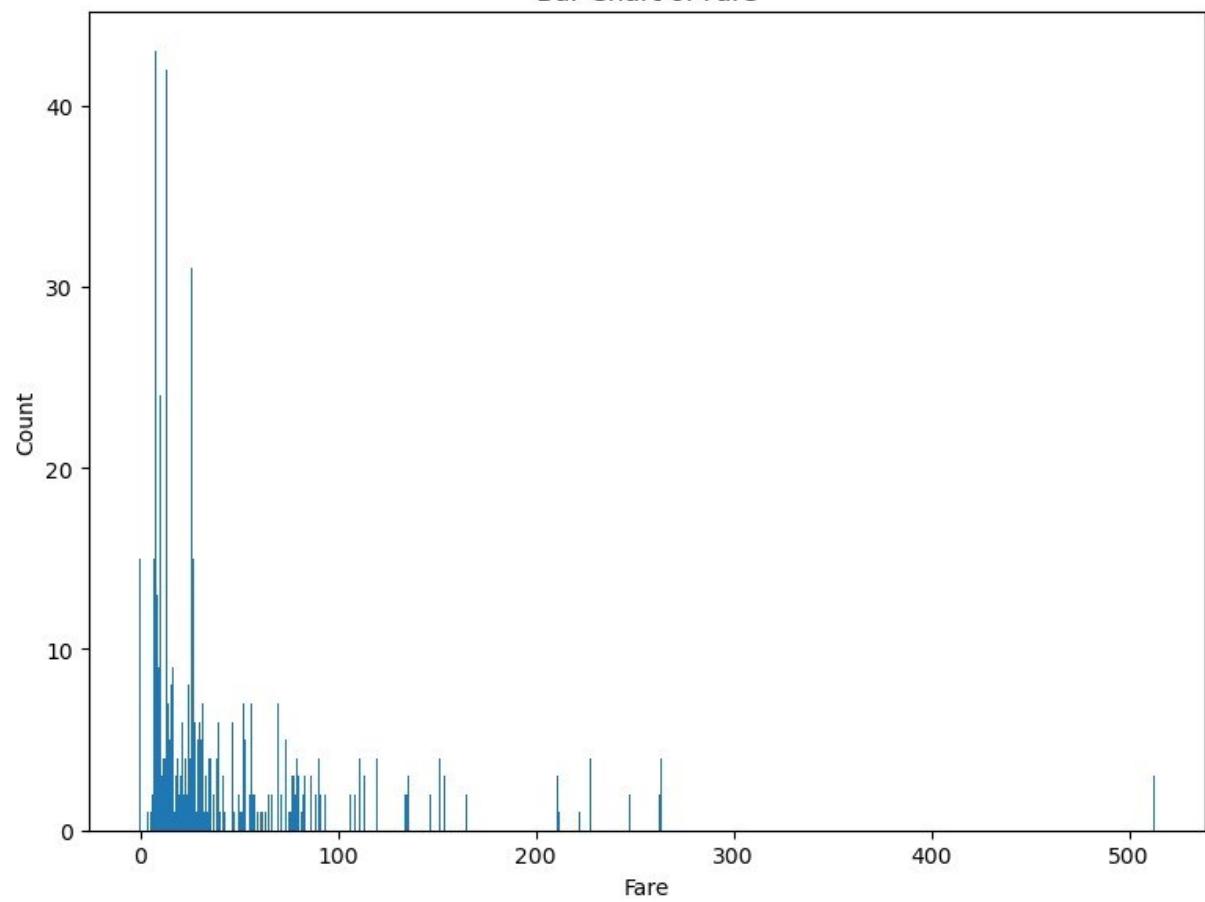




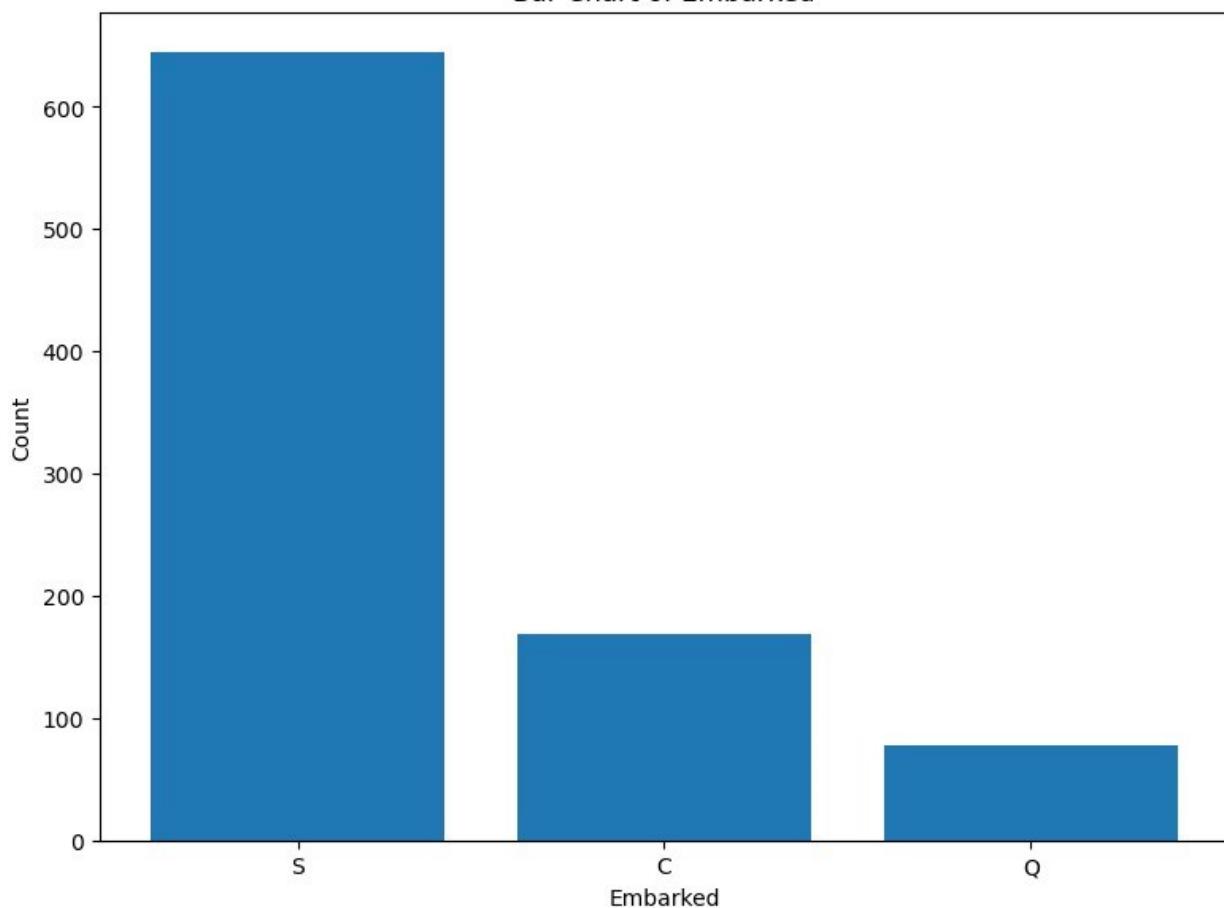
Susp
Bar Chart of Parch



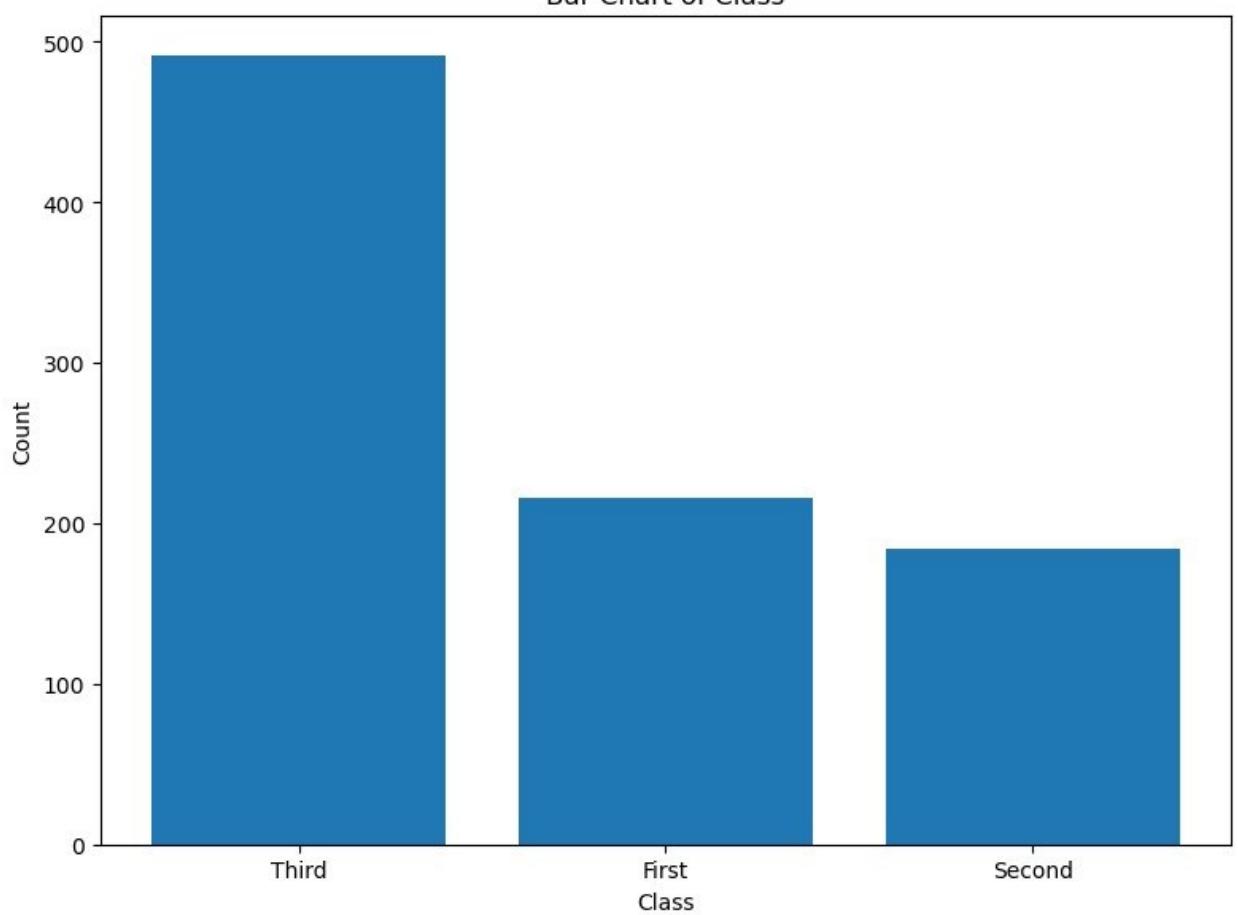
Bar Chart of Fare



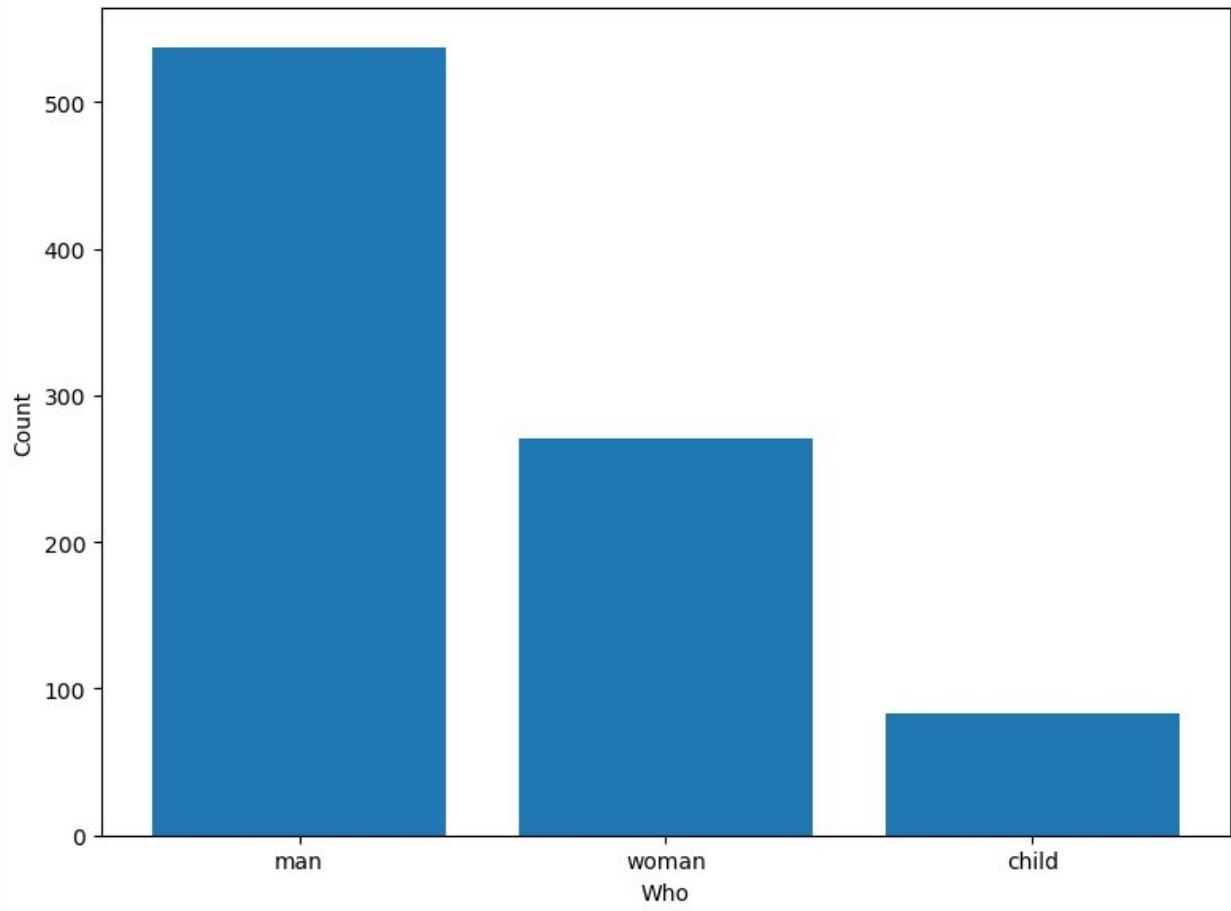
Bar Chart of Embarked



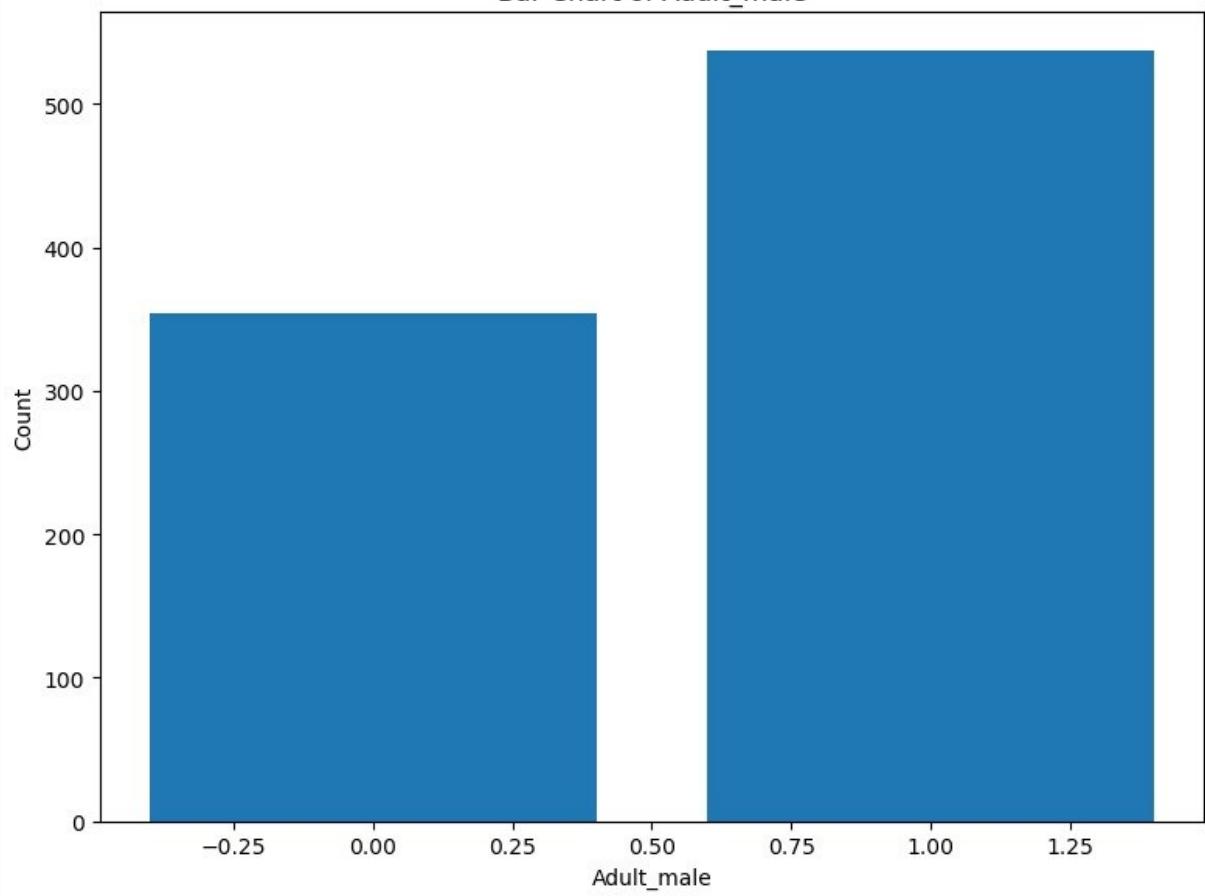
Bar Chart of Class

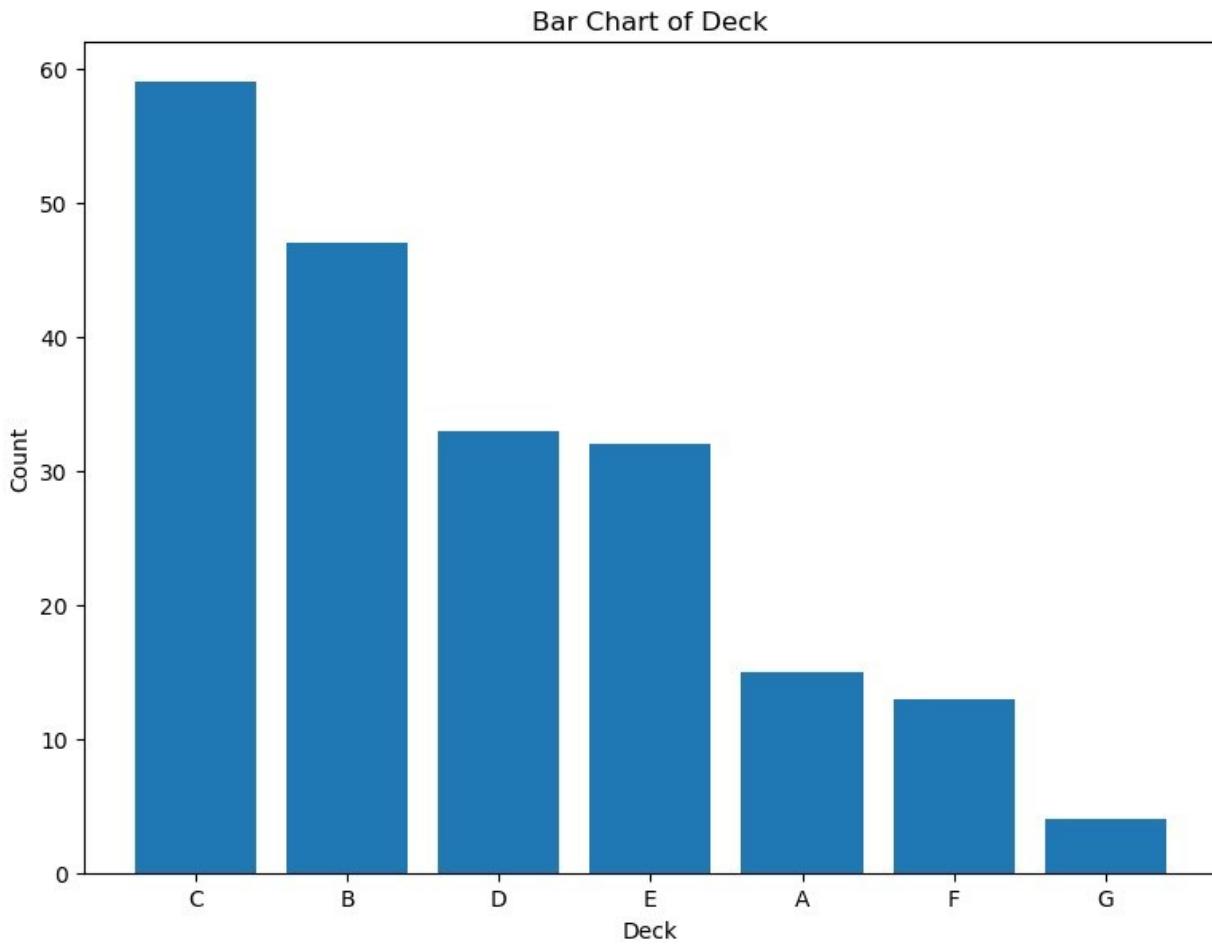


Class
Bar Chart of Who



Bar Chart of Adult_male





```
In [11]: import pandas as pd
import matplotlib.pyplot as plt

# Assuming 'data' is your DataFrame

# List of columns to create box plots for
columns = ['survived', 'pclass', 'age', 'sibsp', 'parch', 'fare']

# Set up the figure and subplots
fig, axes = plt.subplots(nrows=len(columns), ncols=1, figsize=(8, 6 * len(columns)))

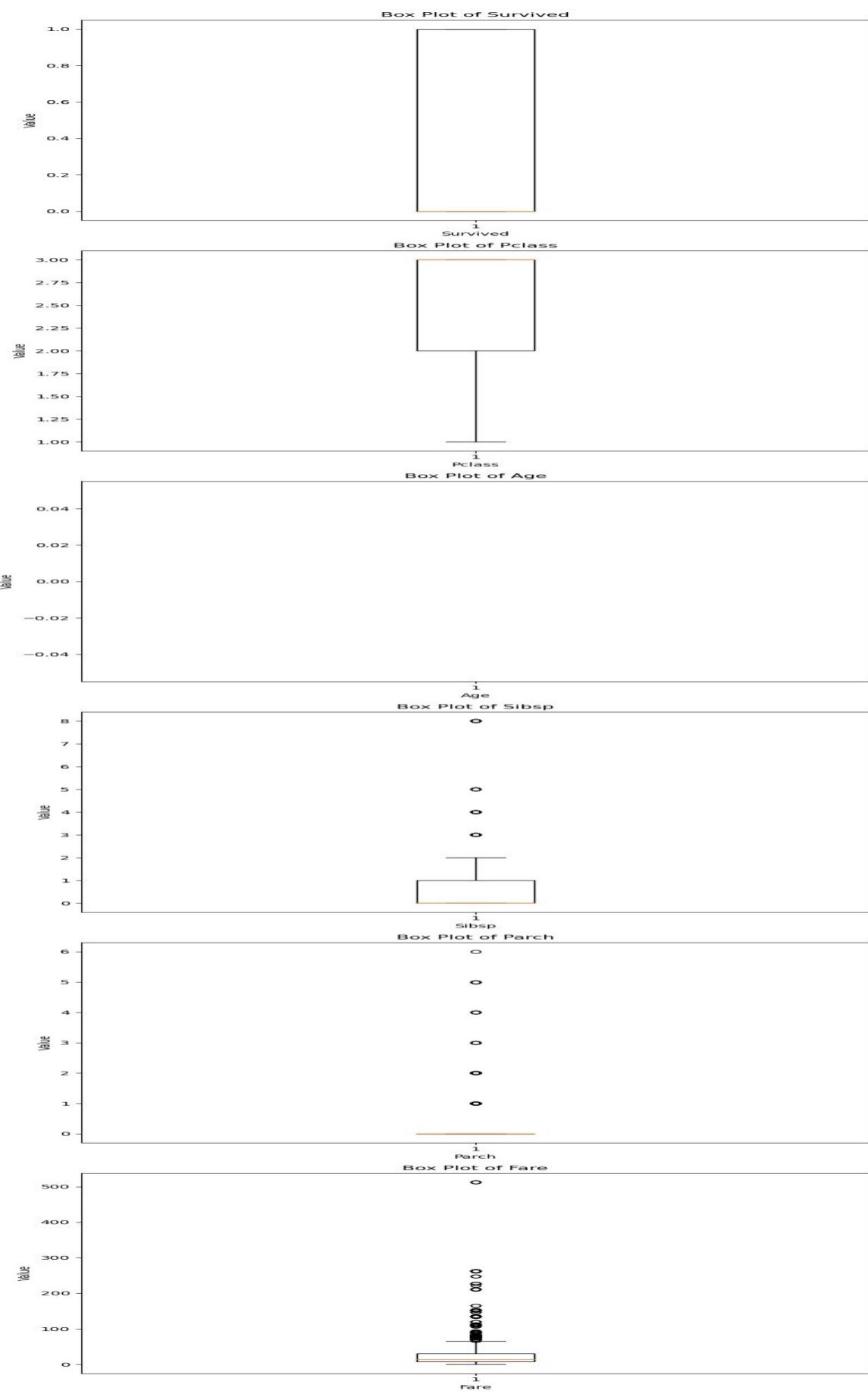
# Create box plots for each column
for i, column in enumerate(columns):
    # Select the column
    column_data = data[column]

    # Create the box plot
    axes[i].boxplot(column_data)

    # Set the labels and title for each subplot
    axes[i].set_xlabel(column.capitalize())
    axes[i].set_ylabel('Value')
    axes[i].set_title(f'Box Plot of {column.capitalize()}')

# Adjust the spacing between subplots
plt.tight_layout()

# Display the box plots
plt.show()
```



```
In [14]: import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'data' is your DataFrame

# List of columns to create KDE plots for
columns = ['survived', 'pclass', 'age', 'sibsp', 'parch', 'fare']

# Set up the figure and subplots
fig, axes = plt.subplots(nrows=len(columns), ncols=1, figsize=(8, 6 * len(columns)))

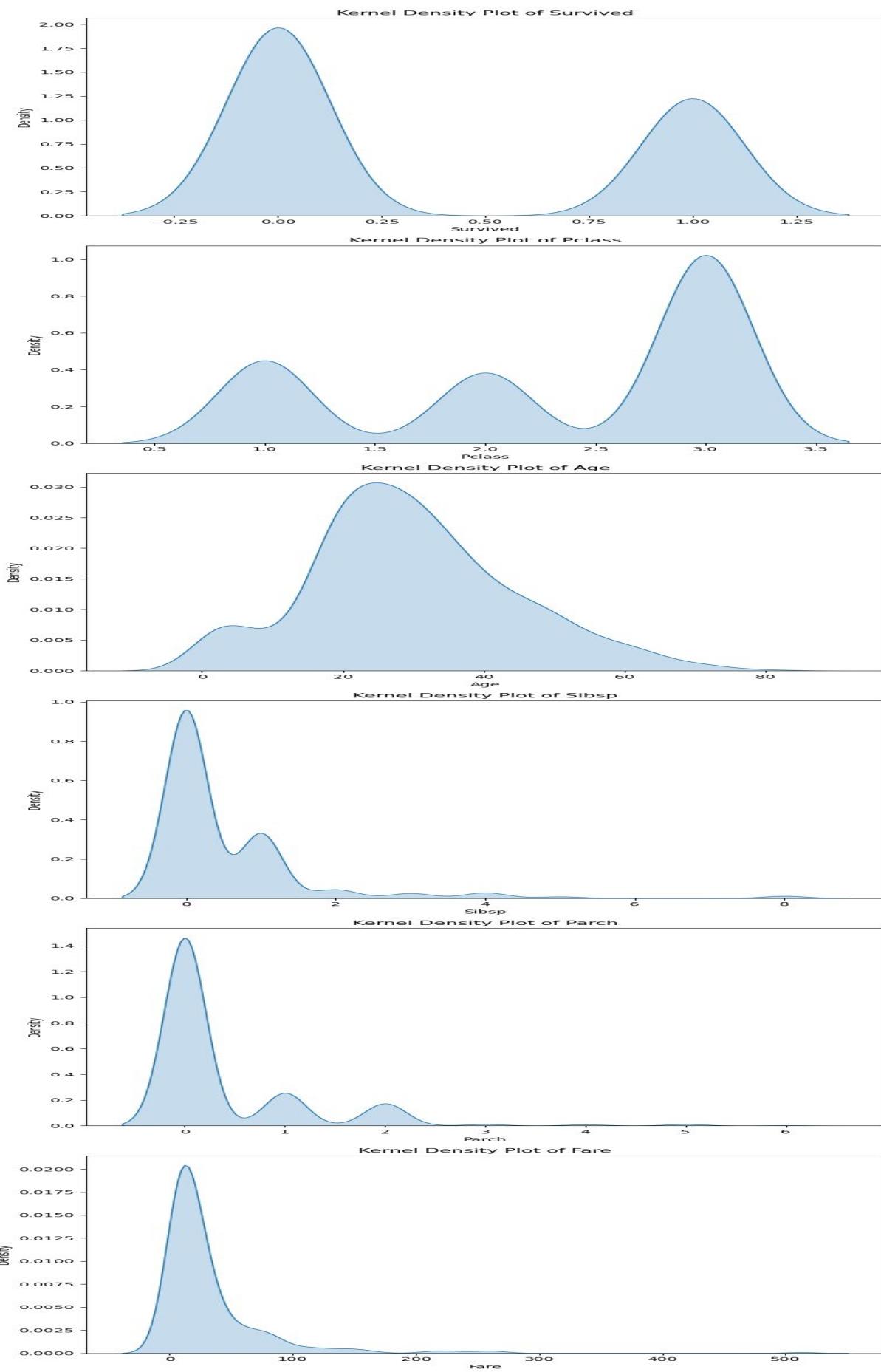
# Create KDE plots for each column
for i, column in enumerate(columns):
    # Select the column
    column_data = data[column]

    # Create the KDE plot
    sns.kdeplot(column_data, ax=axes[i], fill=True)

    # Set the labels and title for each subplot
    axes[i].set_xlabel(column.capitalize())
    axes[i].set_ylabel('Density')
    axes[i].set_title(f'Kernel Density Plot of {column.capitalize()}')

# Adjust the spacing between subplots
plt.tight_layout()

# Display the KDE plots
plt.show()
```



```
In [15]: import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'data' is your DataFrame

# List of columns to create scatter plots for
columns = ['survived', 'pclass', 'age', 'sibsp', 'parch', 'fare']

# Set up the figure and subplots
fig, axes = plt.subplots(nrows=len(columns), ncols=1, figsize=(8, 6 * len(columns)))

# Create scatter plots for each column
for i, column in enumerate(columns):
    # Select the column
    column_data = data[column]

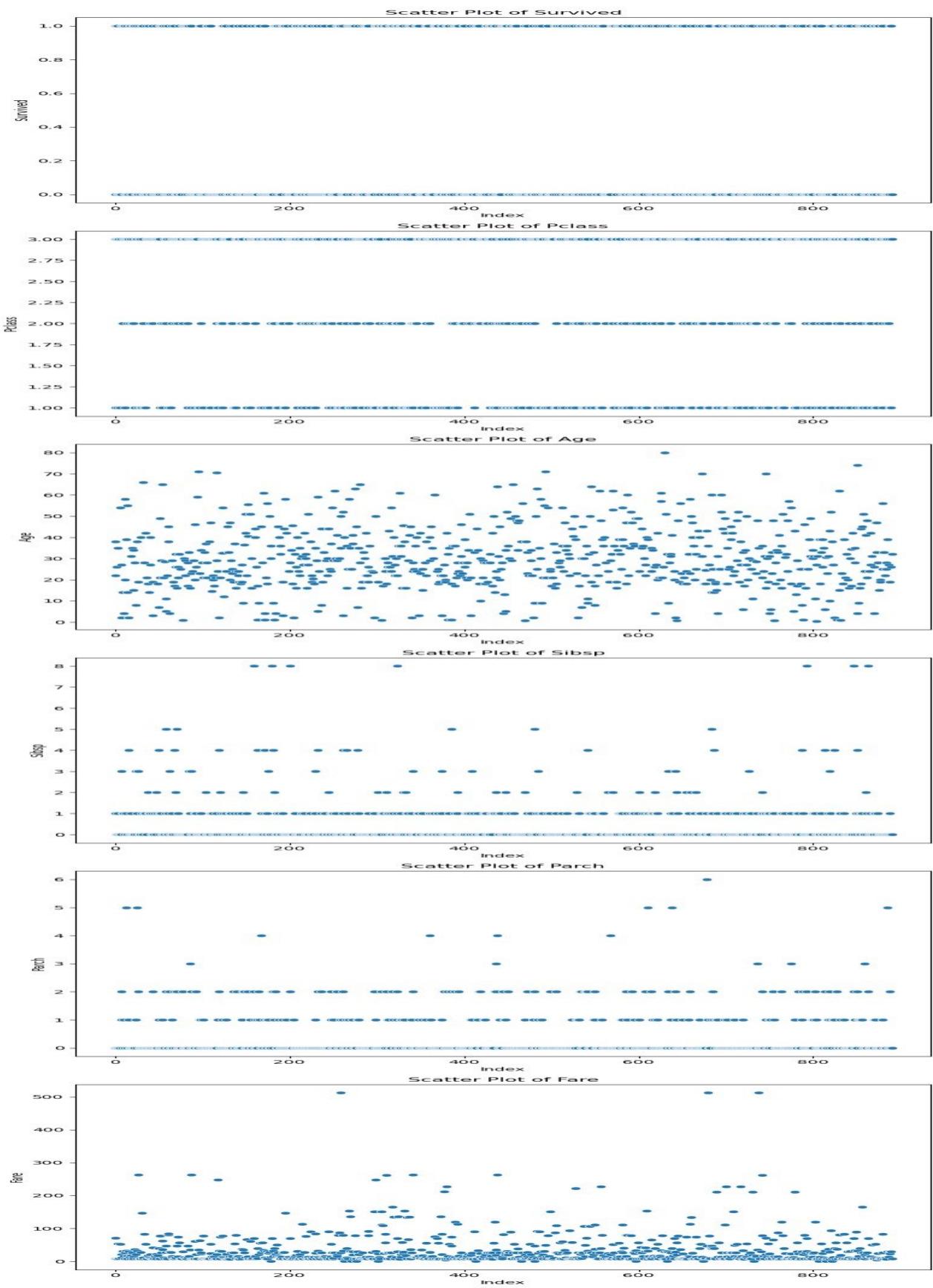
    # Generate x-coordinates for scatter plot
    x = range(len(column_data))

    # Create the scatter plot
    sns.scatterplot(x=x, y=column_data, ax=axes[i])

    # Set the labels and title for each subplot
    axes[i].set_xlabel('Index')
    axes[i].set_ylabel(column.capitalize())
    axes[i].set_title(f'Scatter Plot of {column.capitalize()}')

# Adjust the spacing between subplots
plt.tight_layout()

# Display the scatter plots
plt.show()
```



```
In [16]: import pandas as pd
import matplotlib.pyplot as plt

# Assuming 'data' is your DataFrame

# List of columns to create line charts for
columns = ['survived', 'pclass', 'age', 'sibsp', 'parch', 'fare']

# Set up the figure and subplots
fig, axes = plt.subplots(nrows=len(columns), ncols=1, figsize=(8, 6 * len(columns)))

# Create line charts for each column
for i, column in enumerate(columns):
    # Select the column
    column_data = data[column]

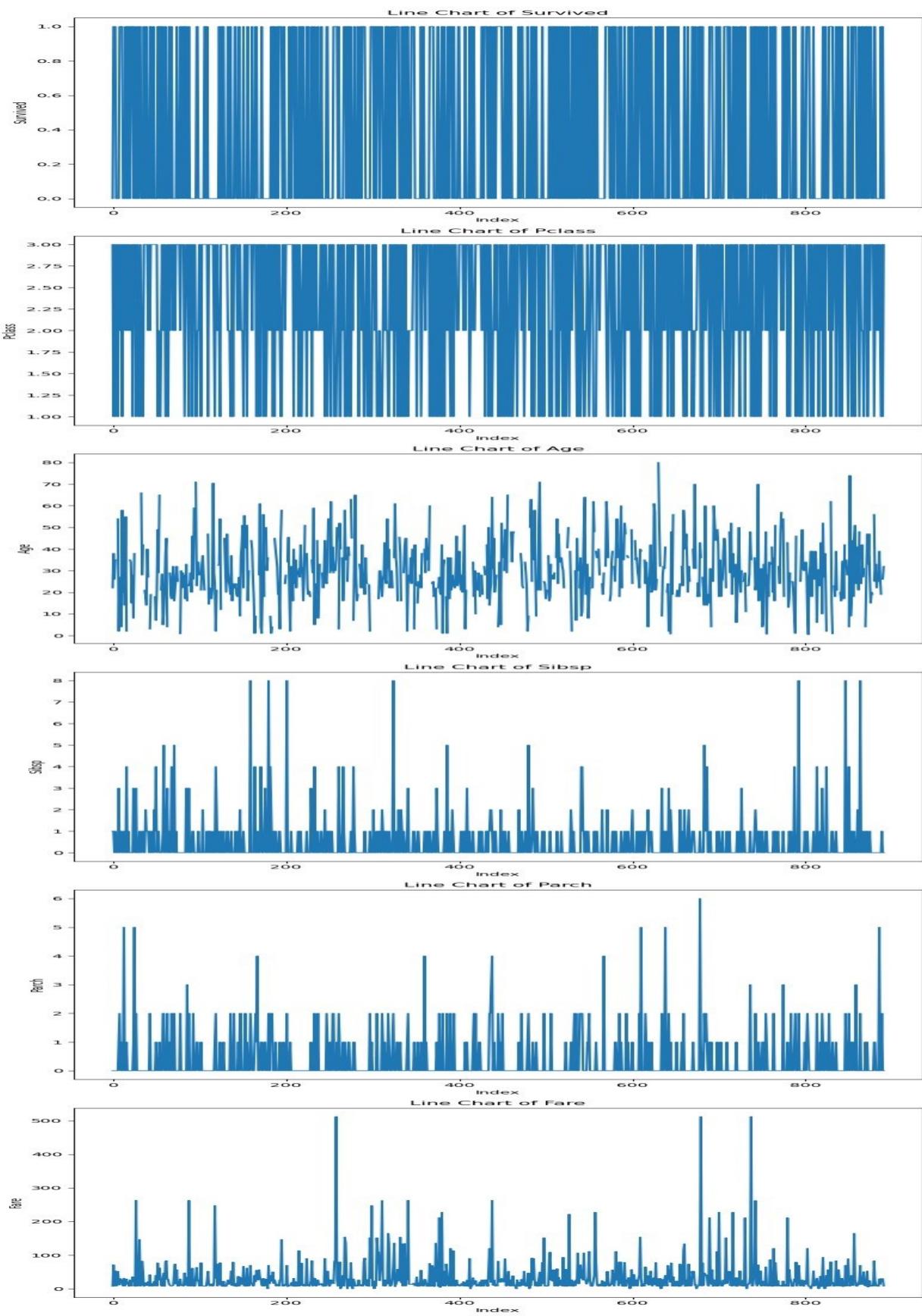
    # Generate x-coordinates for line chart
    x = range(len(column_data))

    # Create the line chart
    axes[i].plot(x, column_data)

    # Set the labels and title for each subplot
    axes[i].set_xlabel('Index')
    axes[i].set_ylabel(column.capitalize())
    axes[i].set_title(f'Line Chart of {column.capitalize()}')

# Adjust the spacing between subplots
plt.tight_layout()

# Display the line charts
plt.show()
```



```
In [17]: import pandas as pd
import matplotlib.pyplot as plt

# Assuming 'data' is your DataFrame

# List of columns for the bar chart
columns = ['survived', 'pclass']

# Set up the figure and subplots
fig, ax = plt.subplots(figsize=(8, 6))

# Set the positions and width for the bars
positions = range(len(data))
width = 0.35

# Create the bar chart
for i, column in enumerate(columns):
    # Select the column
    column_data = data[column]

    # Generate the x-coordinates for the bars
    x = [pos + width * i for pos in positions]

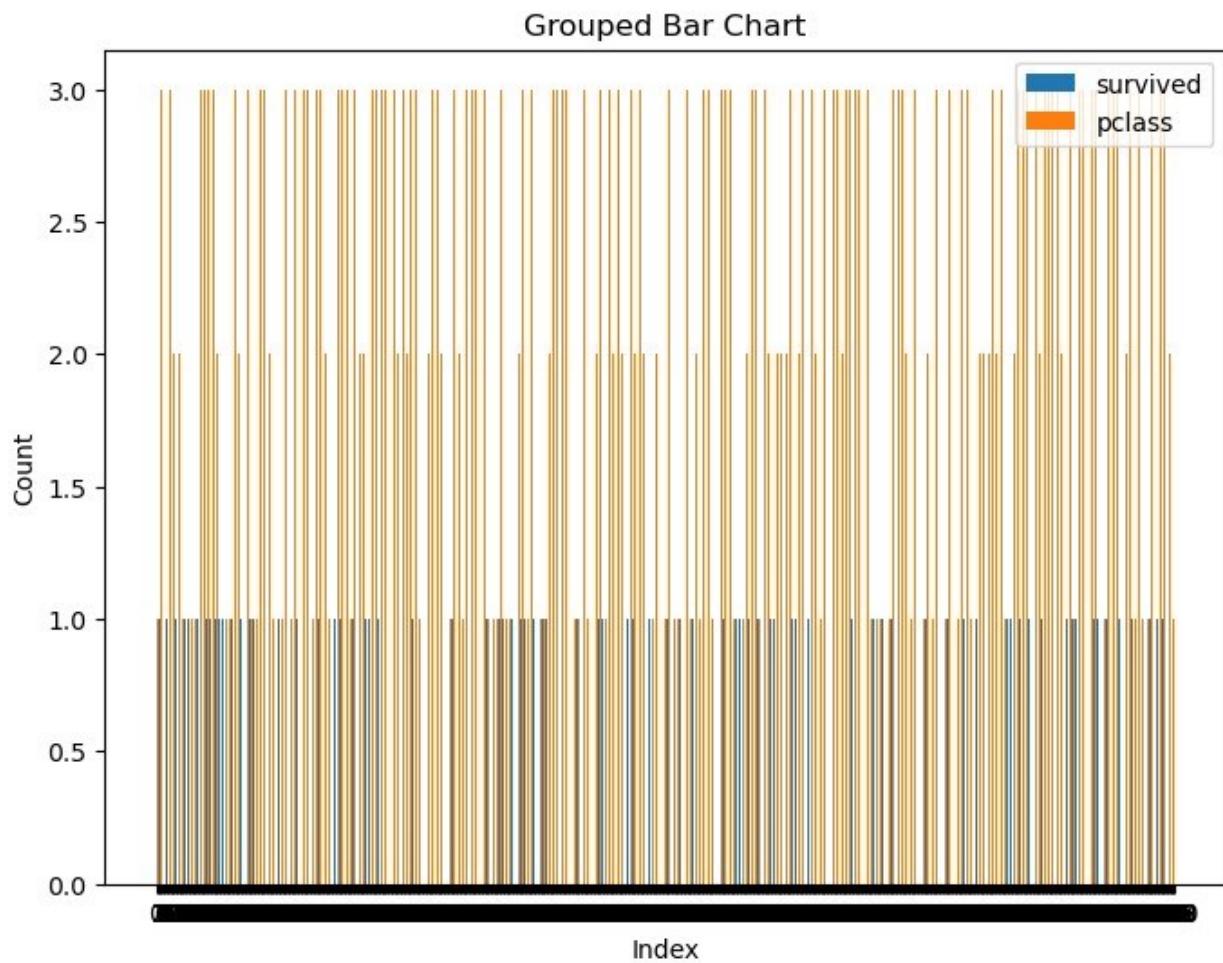
    # Create the bars
    ax.bar(x, column_data, width, label=column)

# Set the labels and title
ax.set_xlabel('Index')
ax.set_ylabel('Count')
ax.set_title('Grouped Bar Chart')

# Set the x-axis ticks and labels
ax.set_xticks([pos + width for pos in positions])
ax.set_xticklabels(data.index)

# Add a legend
ax.legend()

# Display the bar chart
plt.show()
```



```
In [19]: import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'data' is your DataFrame

# Select numeric columns for correlation calculation
numeric_columns = data.select_dtypes(include='number')

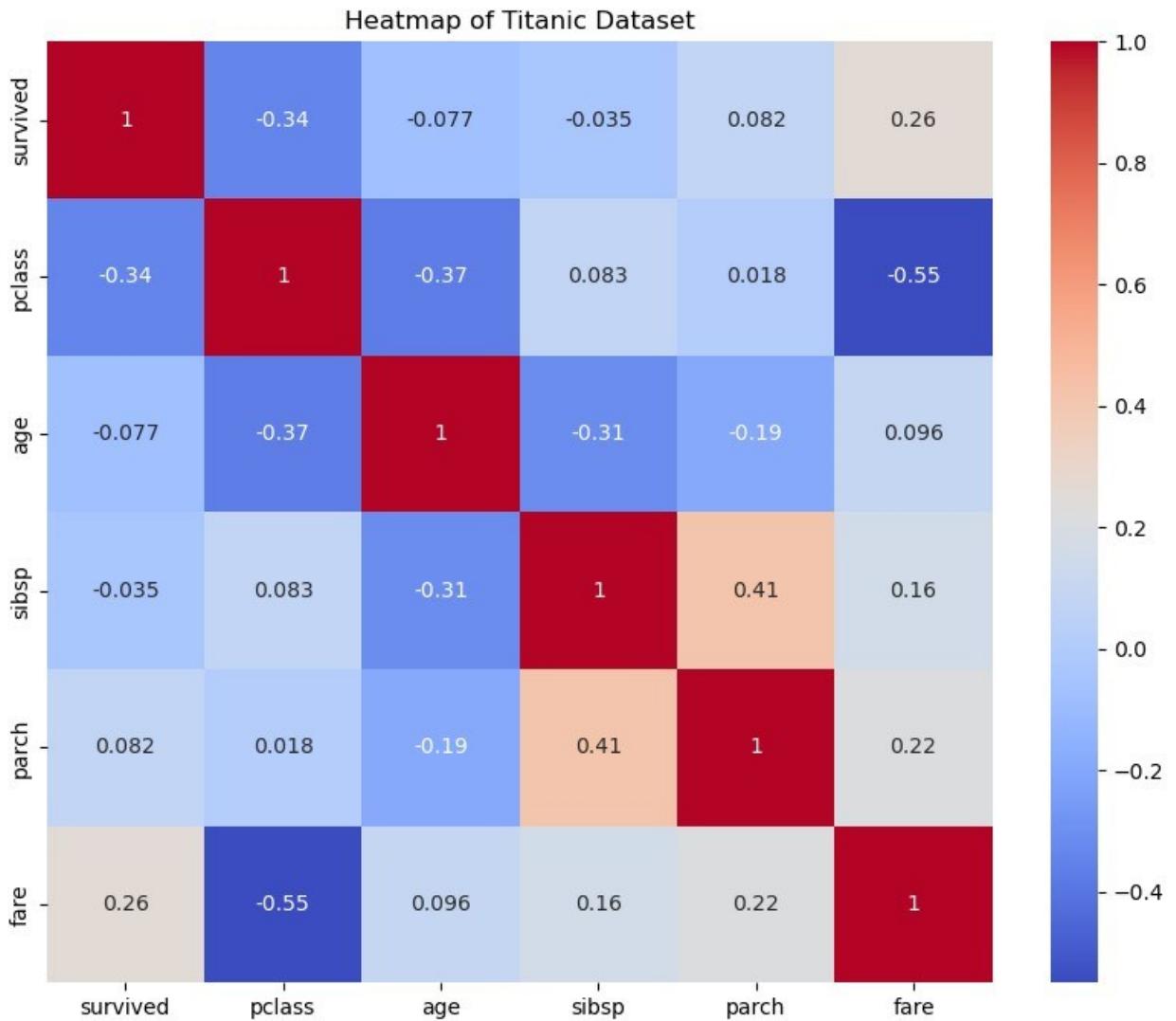
# Compute the correlation matrix
correlation_matrix = numeric_columns.corr()

# Set up the figure and axes
fig, ax = plt.subplots(figsize=(10, 8))

# Create the heatmap
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', ax=ax)

# Set the title
ax.set_title('Heatmap of Titanic Dataset')

# Display the heatmap
plt.show()
```



```
In [20]: import pandas as pd
import matplotlib.pyplot as plt

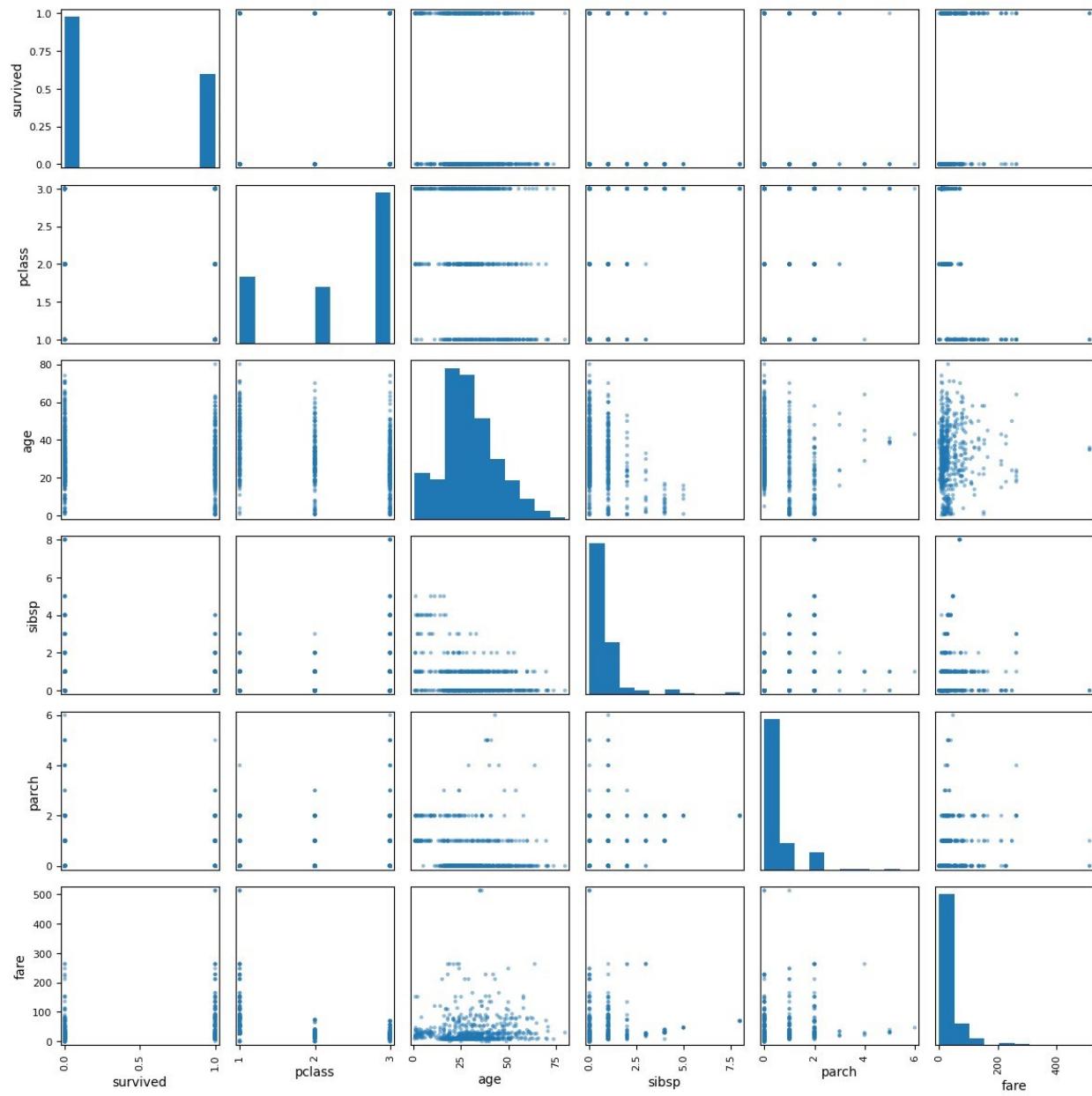
# Assuming 'data' is your DataFrame

# Select the columns for the scatter plot matrix
columns = ['survived', 'pclass', 'age', 'sibsp', 'parch', 'fare']

# Create the scatter plot matrix
scatter_matrix = pd.plotting.scatter_matrix(data[columns], figsize=(12, 12))

# Adjust the spacing between subplots
plt.tight_layout()

# Display the scatter plot matrix
plt.show()
```



```
In [26]: import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

# Assuming 'data' is your DataFrame

# Select the columns for the Parallel Coordinates Plot
columns = ['pclass', 'sex', 'age', 'sibsp', 'parch', 'fare', 'embarked']

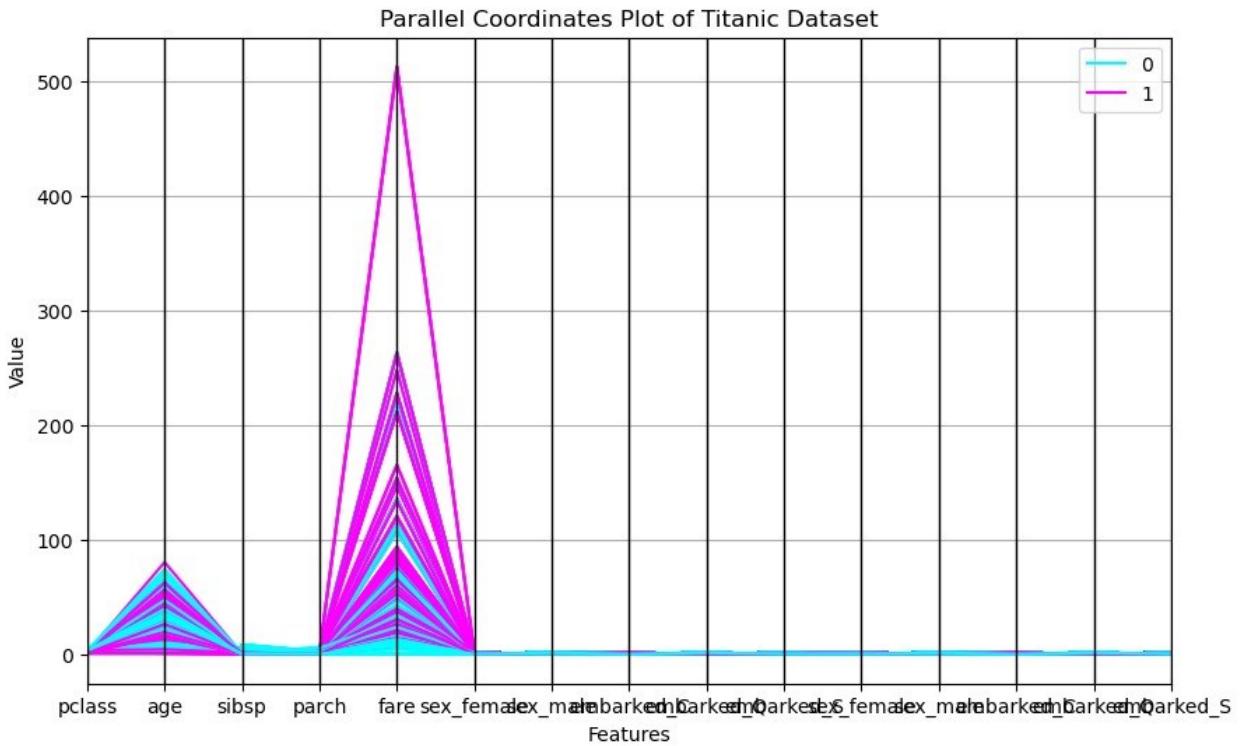
# Encode the 'survived' column
label_encoder = LabelEncoder()
data['survived_encoded'] = label_encoder.fit_transform(data['survived'])

# Encode categorical columns using one-hot encoding
categorical_columns = ['sex', 'embarked']
data_encoded = pd.get_dummies(data[columns + categorical_columns])

# Merge the encoded columns with the target column
data_final = pd.concat([data_encoded, data['survived_encoded']], axis=1)

# Create the Parallel Coordinates Plot using pandas.plotting
plt.figure(figsize=(10, 6))
pd.plotting.parallel_coordinates(data_final, 'survived_encoded', colormap='cool')
plt.title('Parallel Coordinates Plot of Titanic Dataset')
plt.xlabel('Features')
plt.ylabel('Value')
plt.legend()

# Display the Parallel Coordinates Plot
plt.show()
```



```
In [28]: import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Assuming 'data' is your DataFrame

# Select the columns for the 3D scatter plot
columns = ['age', 'fare', 'survived']

# Create a subset of the data with the selected columns
subset = data[columns]

# Remove rows with missing values
subset = subset.dropna()

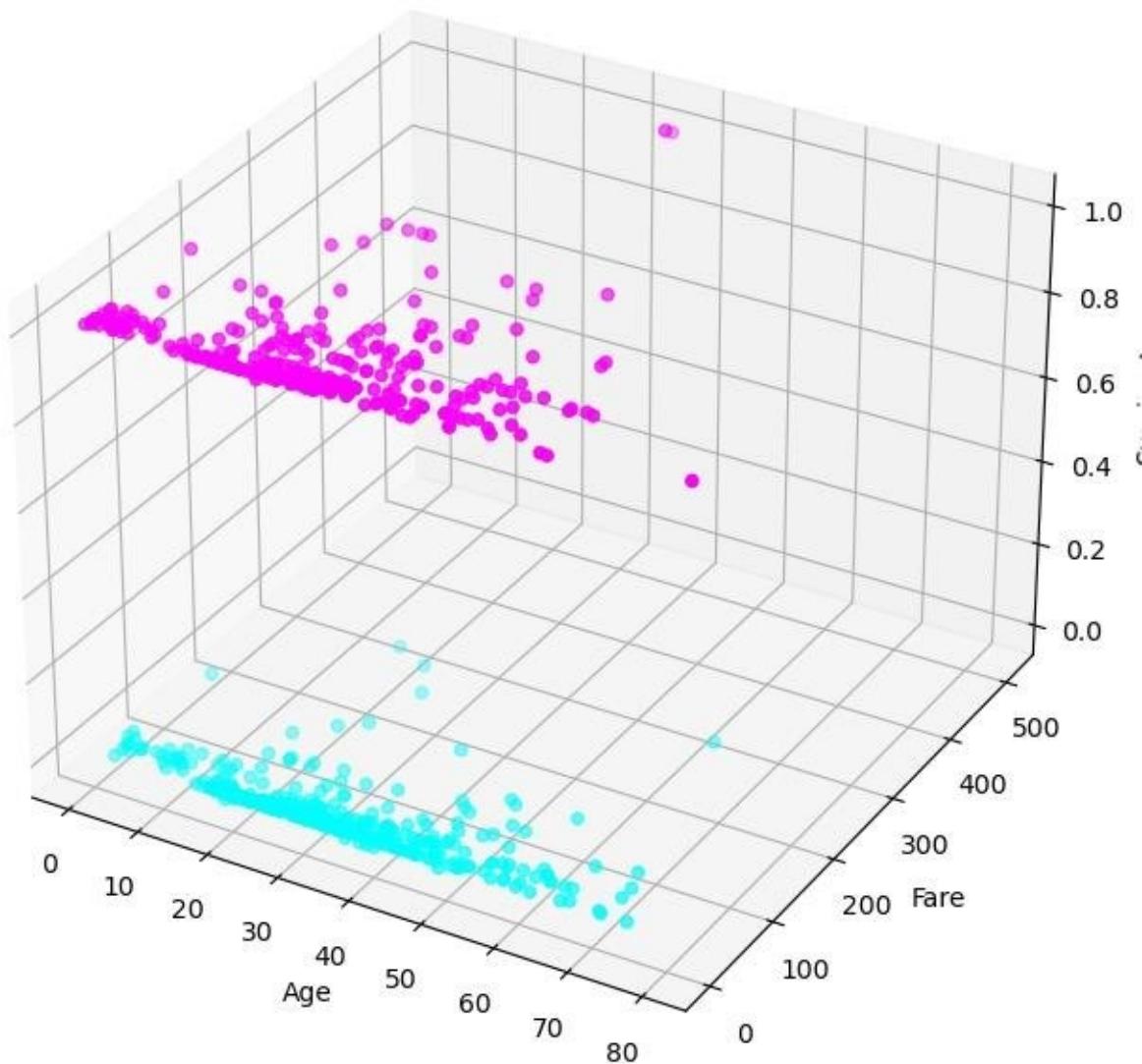
# Create a 3D scatter plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(subset['age'], subset['fare'], subset['survived'], c=subset['survived'], cmap='cool')

# Set Labels for each axis
ax.set_xlabel('Age')
ax.set_ylabel('Fare')
ax.set_zlabel('Survived')

# Set the title of the plot
plt.title('3D Scatter Plot of Titanic Dataset')

# Show the plot
plt.show()
```

3D Scatter Plot of Titanic Dataset



```
In [29]: import pandas as pd
import plotly.express as px

# Assuming 'data' is your DataFrame

# Select the columns for the treemap
columns = ['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare', 'embarked']

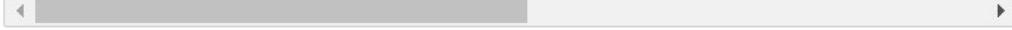
# Create a subset of the data with the selected columns
subset = data[columns]

# Remove rows with missing values
subset = subset.dropna()

# Create the treemap
fig = px.treemap(subset, path=columns)

# Set the title of the treemap
fig.update_layout(title='Treemap of Titanic Dataset')

# Show the treemap
fig.show()
```



4. Perform descriptive statistics on the dataset

```
In [30]: import pandas as pd

# Assuming 'data' is your DataFrame

# Perform descriptive statistics on the dataset
statistics = data.describe(include='all')

# Print the descriptive statistics
print(statistics)
```

	survived	pclass	sex	age	sibsp	parch	\		
count	891.000000	891.000000	891	714.000000	891.000000	891.000000			
unique	NaN	NaN	2	NaN	NaN	NaN			
top	NaN	NaN	male	NaN	NaN	NaN			
freq	NaN	NaN	577	NaN	NaN	NaN			
mean	0.383838	2.308642	NaN	29.699118	0.523008	0.381594			
std	0.486592	0.836071	NaN	14.526497	1.102743	0.806057			
min	0.000000	1.000000	NaN	0.420000	0.000000	0.000000			
25%	0.000000	2.000000	NaN	20.125000	0.000000	0.000000			
50%	0.000000	3.000000	NaN	28.000000	0.000000	0.000000			
75%	1.000000	3.000000	NaN	38.000000	1.000000	0.000000			
max	1.000000	3.000000	NaN	80.000000	8.000000	6.000000			
	fare	embarked	class	who	adult_male	deck	embark_town	alive	\
count	891.000000	889	891	891	891	203	889	891	
unique	NaN	3	3	3	2	7	3	2	
top	NaN	S	Third	man	True	C	Southampton	no	
freq	NaN	644	491	537	537	59	644	549	
mean	32.204208	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
std	49.693429	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
min	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
25%	7.910400	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
50%	14.454200	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
75%	31.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
max	512.329200	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
	alone	sex_encoded	survived_encoded						
count	891	891.000000	891.000000						
unique	2	NaN	NaN						

5. Handle the Missing values

```
In [31]: import pandas as pd

# Load the Titanic dataset
data = pd.read_csv('titanic.csv')

# Check for missing values
print(data.isnull().sum())

# Drop rows with missing values
data = data.dropna()

# Fill missing values with a specific value
data['age'] = data['age'].fillna(data['age'].mean())
data['embarked'] = data['embarked'].fillna(data['embarked'].mode()[0])

# Perform Linear interpolation to fill missing fare values
data['fare'] = data['fare'].interpolate(method='linear')

# Drop columns with a high percentage of missing values
data = data.drop('deck', axis=1)

# Check for missing values again to confirm
print(data.isnull().sum())
```

```
survived      0
pclass        0
sex          0
age         177
sibsp        0
parch        0
fare          0
embarked     2
class        0
who          0
adult_male   0
deck       688
embark_town  2
alive        0
alone        0
dtype: int64
survived      0
pclass        0
sex          0
age          0
sibsp        0
parch        0
fare          0
embarked     0
class        0
who          0
adult_male   0
embark_town  0
alive        0
alone        0
dtype: int64
```

6. Find the outliers and replace the outliers

```
In [33]: import pandas as pd
import numpy as np
from scipy import stats

# Load the Titanic dataset
data = pd.read_csv('titanic.csv')

# Identify outliers using z-score
z_scores = np.abs(stats.zscore(data['fare']))
threshold = 3
outliers = np.where(z_scores > threshold)

# Replace outliers with the median value
median_fare = data['fare'].median()
data.loc[outliers[0], 'fare'] = median_fare
|
# Check for outliers again to confirm
z_scores_after = np.abs(stats.zscore(data['fare']))
new_outliers = np.where(z_scores_after > threshold)
print("Number of outliers after replacement:", len(new_outliers[0]))
```

Number of outliers after replacement: 22

```
In [34]: import pandas as pd
import numpy as np
from scipy import stats

# Load the Titanic dataset
data = pd.read_csv('titanic.csv')

# Calculate z-scores for the 'fare' column
z_scores = np.abs(stats.zscore(data['fare']))

# Set the threshold for identifying outliers
threshold = 3

# Find the outliers based on the z-scores
outliers = data[z_scores > threshold]

# Print the outliers
print("Outliers in the 'fare' column:")
print(outliers)
```

Outliers in the 'fare' column:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	\
27	0	1	male	19.0	3	2	263.0000	S	First	
88	1	1	female	23.0	3	2	263.0000	S	First	
118	0	1	male	24.0	0	1	247.5208	C	First	
258	1	1	female	35.0	0	0	512.3292	C	First	
299	1	1	female	50.0	0	1	247.5208	C	First	
311	1	1	female	18.0	2	2	262.3750	C	First	
341	1	1	female	24.0	3	2	263.0000	S	First	
377	0	1	male	27.0	0	2	211.5000	C	First	
380	1	1	female	42.0	0	0	227.5250	C	First	
438	0	1	male	64.0	1	4	263.0000	S	First	
527	0	1	male	NaN	0	0	221.7792	S	First	
557	0	1	male	NaN	0	0	227.5250	C	First	
679	1	1	male	36.0	0	1	512.3292	C	First	
689	1	1	female	15.0	0	1	211.3375	S	First	
700	1	1	female	18.0	1	0	227.5250	C	First	
716	1	1	female	38.0	0	0	227.5250	C	First	
730	1	1	female	29.0	0	0	211.3375	S	First	
737	1	1	male	35.0	0	0	512.3292	C	First	
742	1	1	female	21.0	2	2	262.3750	C	First	
779	1	1	female	43.0	0	1	211.3375	S	First	

	who	adult_male	deck	embark_town	alive	alone
27	man	True	C	Southampton	no	False
88	woman	False	C	Southampton	yes	False
118	man	True	B	Cherbourg	no	False
258	woman	False	NaN	Cherbourg	yes	True
299	woman	False	B	Cherbourg	yes	False
311	woman	False	B	Cherbourg	yes	False
341	woman	False	C	Southampton	yes	False
377	man	True	C	Cherbourg	no	False
380	woman	False	NaN	Cherbourg	yes	True
438	man	True	C	Southampton	no	False
527	man	True	C	Southampton	no	True
557	man	True	NaN	Cherbourg	no	True
679	man	True	B	Cherbourg	yes	False
689	child	False	B	Southampton	yes	False
700	woman	False	C	Cherbourg	yes	False
716	woman	False	C	Cherbourg	yes	True
730	woman	False	B	Southampton	yes	True
737	man	True	B	Cherbourg	yes	True
742	woman	False	B	Cherbourg	yes	False
779	woman	False	B	Southampton	yes	False

7. Check for Categorical columns and perform encoding

```
In [35]: import pandas as pd

# Load the Titanic dataset
data = pd.read_csv('titanic.csv')

# Check for categorical columns
categorical_columns = data.select_dtypes(include=['object']).columns

# Perform encoding for categorical columns
data_encoded = pd.get_dummies(data, columns=categorical_columns)

# Print the encoded dataset
print("Encoded dataset:")
print(data_encoded.head())
```

```
Encoded dataset:
   survived  pclass    age  sibsp  parch      fare  adult_male  alone \
0          0      3  22.0      1      0    7.2500      True  False
1          1      1  38.0      1      0   71.2833     False  False
2          1      3  26.0      0      0    7.9250     False  True
3          1      1  35.0      1      0   53.1000     False  False
4          0      3  35.0      0      0    8.0500      True  True

   sex_female  sex_male ...  deck_C  deck_D  deck_E  deck_F  deck_G \
0          0        1 ...      0      0      0      0      0
1          1        0 ...      1      0      0      0      0
2          1        0 ...      0      0      0      0      0
3          1        0 ...      1      0      0      0      0
4          0        1 ...      0      0      0      0      0

   embark_town_Cherbourg  embark_town_Queenstown  embark_town_Southampton \
0                      0                      0                      1
1                      1                      0                      0
2                      0                      0                      1
3                      0                      0                      1
4                      0                      0                      1

   alive_no  alive_yes
0         1         0
1         0         1
2         0         1
3         0         1
4         1         0

[5 rows x 31 columns]
```

8. Split the data into dependent and independent variables.

```
In [36]: import pandas as pd

# Load the Titanic dataset
data = pd.read_csv('titanic.csv')

# Split into dependent and independent variables
X = data.drop('survived', axis=1) # Independent variables (features)
y = data['survived'] # Dependent variable (target)

# Print the shapes of the variables
print("Independent variables shape:", X.shape)
print("Dependent variable shape:", y.shape)
```

Independent variables shape: (891, 14)
Dependent variable shape: (891,)

```
In [37]: print(X)

   pclass      sex    age  sibsp  parch     fare embarked  class who \
0       3    male  22.0      1      0    7.2500        S  Third  man
1       1  female  38.0      1      0   71.2833        C First woman
2       3  female  26.0      0      0    7.9250        S  Third woman
3       1  female  35.0      1      0   53.1000        S First woman
4       3    male  35.0      0      0    8.0500        S  Third  man
..     ...
886      2    male  27.0      0      0   13.0000        S Second  man
887      1  female  19.0      0      0   30.0000        S First woman
888      3  female   NaN      1      2   23.4500        S  Third woman
889      1    male  26.0      0      0   30.0000        C First  man
890      3    male  32.0      0      0    7.7500        Q  Third  man

   adult_male  deck embark_town  alive  alone
0      True    NaN  Southampton  no  False
1     False     C  Cherbourg  yes  False
2     False    NaN  Southampton  yes  True
3     False     C  Southampton  yes  False
4      True    NaN  Southampton  no  True
..   ...
886     True    NaN  Southampton  no  True
887     False    B  Southampton  yes  True
888     False    NaN  Southampton  no  False
889     True     C  Cherbourg  yes  True
890     True    NaN  Queenstown  no  True

[891 rows x 14 columns]
```

```
In [39]: print(y)

0    0
1    1
2    1
3    1
4    0
..
886   0
887   1
888   0
889   1
890   0
Name: survived, Length: 891, dtype: int64
```

9. Scale the independent variables

```
In [43]: from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Load the Titanic dataset
data = pd.read_csv('titanic.csv')

# Split into dependent and independent variables
X = data.drop('survived', axis=1) # Independent variables (features)
y = data['survived'] # Dependent variable (target)

# Identify the categorical columns
categorical_cols = X.select_dtypes(include=['object']).columns

# Perform one-hot encoding on categorical columns
encoder = OneHotEncoder(drop='first')
X_encoded = encoder.fit_transform(X[categorical_cols]).toarray()
encoded_cols = encoder.get_feature_names_out(categorical_cols)
X_encoded = pd.DataFrame(X_encoded, columns=encoded_cols)

# Concatenate encoded columns with remaining columns
X_encoded = pd.concat([X_encoded, X.drop(categorical_cols, axis=1)], axis=1)

# Scale the independent variables
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_encoded)

# Print the scaled independent variables
print(X_scaled)
```

```
[[ 0.73769513 -0.30756234  0.61930636 ... -0.50244517  0.81192233
-1.2316449 ]
[-1.35557354 -0.30756234 -1.61470971 ...  0.78684529 -1.2316449
-1.2316449 ]
[-1.35557354 -0.30756234  0.61930636 ... -0.48885426 -1.2316449
0.81192233]
...
[-1.35557354 -0.30756234  0.61930636 ... -0.17626324 -1.2316449
-1.2316449 ]
[ 0.73769513 -0.30756234 -1.61470971 ... -0.04438104  0.81192233
0.81192233]
[ 0.73769513  3.25137334 -1.61470971 ... -0.49237783  0.81192233
0.81192233]]
```

10. Split the data into training and testing

```
In [44]: from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Print the shapes of the training and testing sets
print("Training set shape:", X_train.shape, y_train.shape)
print("Testing set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (712, 26) (712,)
Testing set shape: (179, 26) (179,)
```

```
In [45]: print("Training set:")
print(X_train)
print(y_train)

print("Testing set:")
print(X_test)
print(y_test)|
```

Training set:

[[0.73769513 -0.30756234 0.61930636 ... -0.07458307 0.81192233
 0.81192233]
 [0.73769513 -0.30756234 0.61930636 ... -0.38667072 0.81192233
 0.81192233]
 [0.73769513 -0.30756234 0.61930636 ... -0.48885426 0.81192233
 0.81192233]
 ...
 [0.73769513 -0.30756234 0.61930636 ... -0.36435545 0.81192233
 -1.2316449]
 [-1.35557354 -0.30756234 0.61930636 ... 1.76774081 -1.2316449
 -1.2316449]
 [0.73769513 -0.30756234 0.61930636 ... 0.90773798 0.81192233
 -1.2316449]]
331 0
733 0
382 0
704 0
813 0
...
106 1
270 0
860 0
435 1
102 0
Name: survived, Length: 712, dtype: int64

Testing set:

[[0.73769513 -0.30756234 -1.61470971 ... -0.34145224 0.81192233
 -1.2316449]
 [0.73769513 -0.30756234 0.61930636 ... -0.43700744 0.81192233
 0.81192233]
 [0.73769513 -0.30756234 0.61930636 ... -0.48885426 0.81192233]