Problem

The problem is to predict whether a passenger survived or not based on various features such as age, sex, class, etc.

Importing Libraries

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Data Collection and Preprocessing

In [3]:
```
data = pd.read_csv('titanic survival data.csv')
data.head()
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	892	0	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292
1	893	1	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000
2	894	0	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875
3	895	0	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625
4	896	1	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Survived     418 non-null    int64  
 2   Pclass       418 non-null    int64  
 3   Name         418 non-null    object 
 4   Sex          418 non-null    object 
 5   Age          332 non-null    float64
 6   SibSp        418 non-null    int64  
 7   Parch        418 non-null    int64  
 8   Ticket       418 non-null    object 
 9   Fare         417 non-null    float64
 10  Cabin        91 non-null     object 
 11  Embarked     418 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 39.3+ KB
```

In [5]:
```
data_description = data.describe()
data_description
```

Out[5]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	418.000000	418.000000	418.000000	332.000000	418.000000	418.000000	417.00000
mean	1100.500000	0.363636	2.265550	30.272590	0.447368	0.392344	35.62718
std	120.810458	0.481622	0.841838	14.181209	0.896760	0.981429	55.90757
min	892.000000	0.000000	1.000000	0.170000	0.000000	0.000000	0.00000
25%	996.250000	0.000000	1.000000	21.000000	0.000000	0.000000	7.89580
50%	1100.500000	0.000000	3.000000	27.000000	0.000000	0.000000	14.45420
75%	1204.750000	1.000000	3.000000	39.000000	1.000000	0.000000	31.50000
max	1309.000000	1.000000	3.000000	76.000000	8.000000	9.000000	512.32920

In [6]:
```
data.isna()
```

Out[6]:

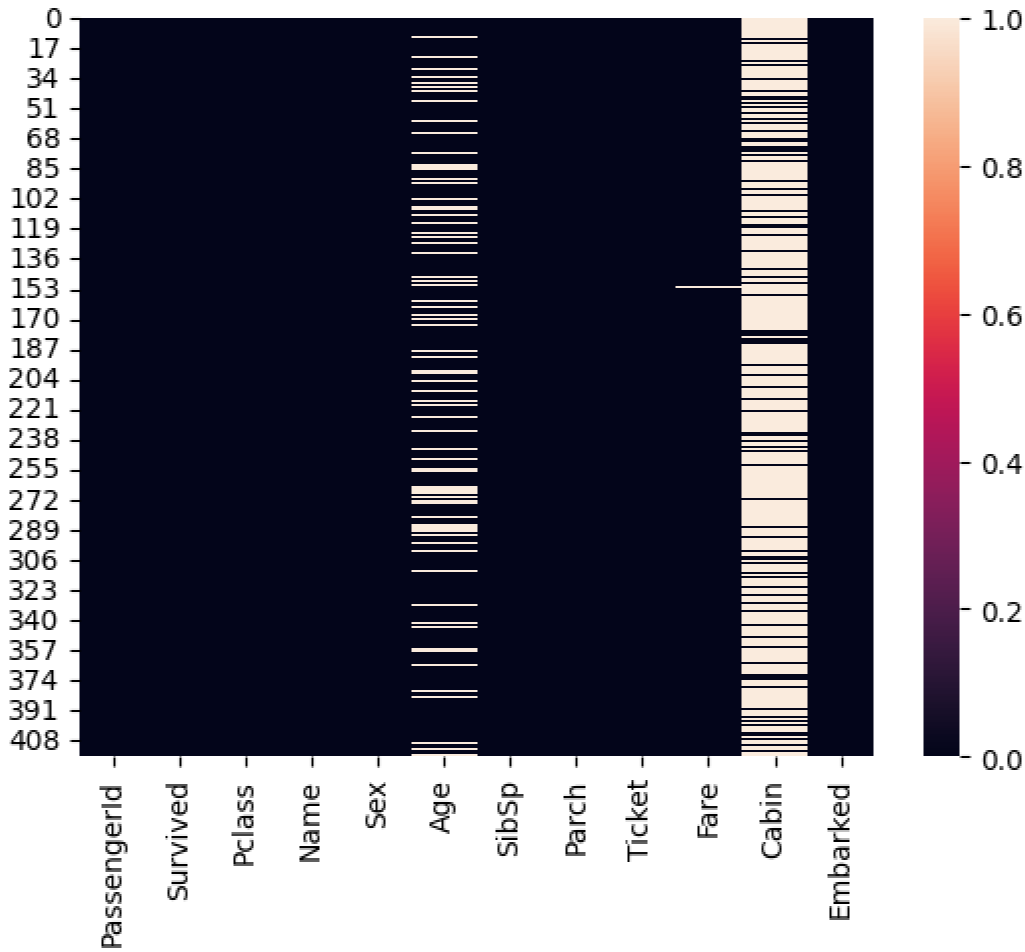
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	False	False	False	False	False	False	False	False	False	False	True
1	False	False	False	False	False	False	False	False	False	False	True
2	False	False	False	False	False	False	False	False	False	False	True
3	False	False	False	False	False	False	False	False	False	False	True
4	False	False	False	False	False	False	False	False	False	False	True
...	...	...	...	...	...	...	...	...	...	...	...
413	False	False	False	False	False	True	False	False	False	False	True
414	False	False	False	False	False	False	False	False	False	False	False
415	False	False	False	False	False	False	False	False	False	False	True
416	False	False	False	False	False	True	False	False	False	False	True
417	False	False	False	False	False	True	False	False	False	False	True

418 rows × 12 columns

In [7]:

```
sns.heatmap(data.isna())
#visualising missing values, cabin has max null values.
```

Out[7]: <Axes: >



In [8]:
```
data.isna().sum()
```

Out[8]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

In [9]:
```
# filling mean values of fare and age
fare_mean = data_description.loc['mean', 'Fare']
data['Fare'].fillna(fare_mean, inplace = True)
```

In [10]:
```
age = data['Age']
age_mean = data_description.loc['mean', 'Age']
age.fillna(age_mean, inplace = True)
```

In [11]:
```
age.isna().sum()
```

Out[11]: 0

In [12]:
```
# dropping irrelevant features
data.drop(['Cabin','PassengerId','Name','Ticket'], axis=1, inplace=True)
```

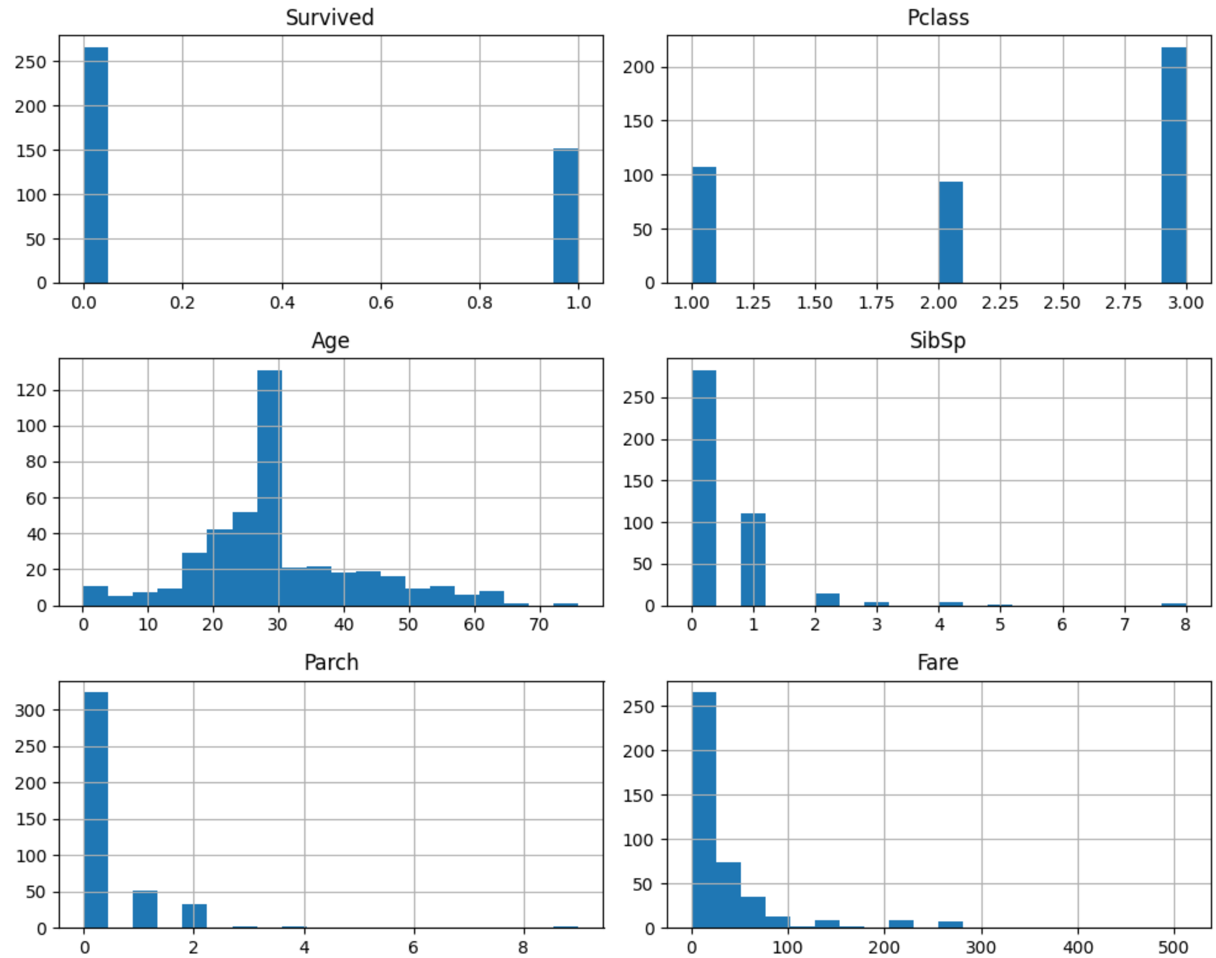
In [13]:
```
data.head()
```

Out[13]:

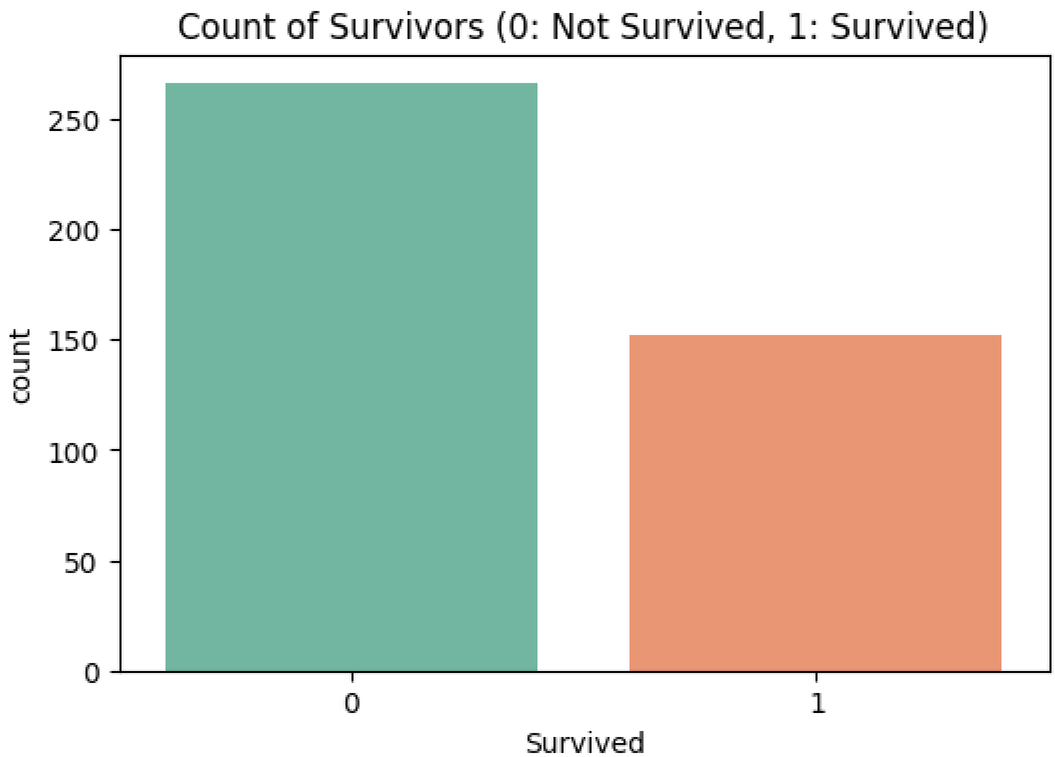
	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	34.5	0	0	7.8292	Q
1	1	3	female	47.0	1	0	7.0000	S
2	0	2	male	62.0	0	0	9.6875	Q
3	0	3	male	27.0	0	0	8.6625	S
4	1	3	female	22.0	1	1	12.2875	S

Data visualization

In [14]:
```
data.hist(bins=20, figsize=(10, 8))
plt.tight_layout()
plt.show()
```

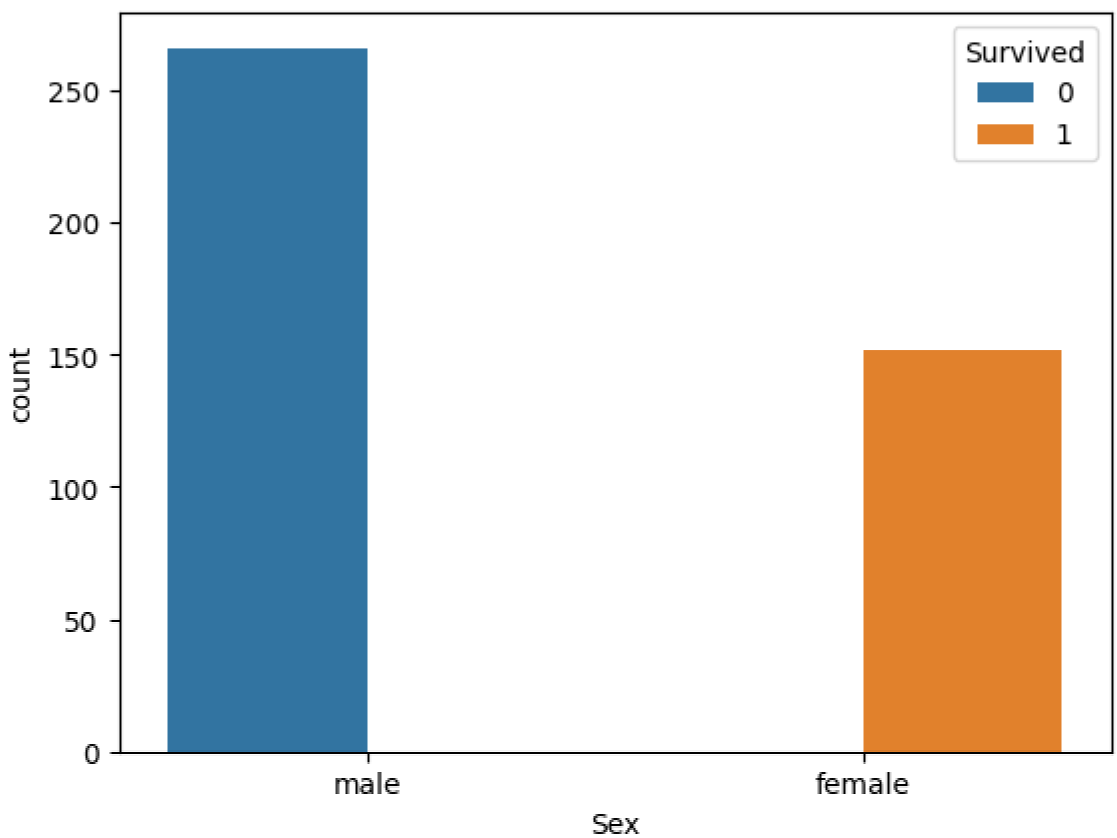


In [15]:
```
# visualizing count of survivors and non-survivors using a bar plot
plt.figure(figsize=(6, 4))
sns.countplot(x='Survived', data=data, palette='Set2')
plt.title('Count of Survivors (0: Not Survived, 1: Survived)')
plt.show()
```

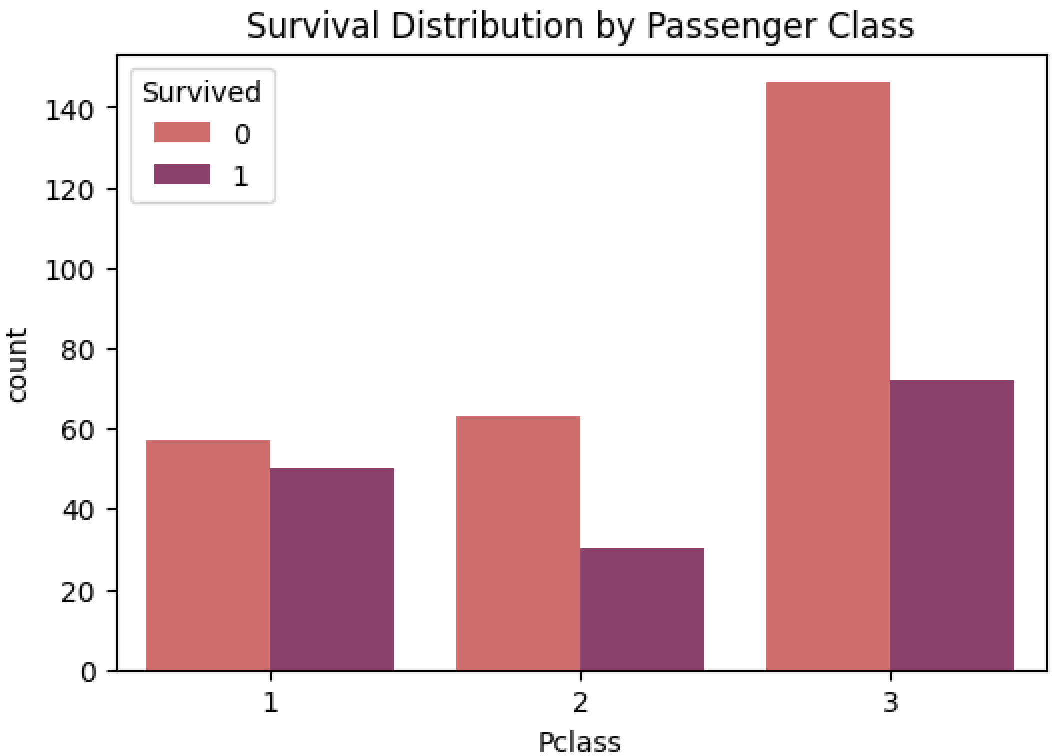


In [16]:
```
# analyzing count of male and female survivors
sns.countplot(x = 'Sex', hue='Survived', data=data)
```

Out[16]: <Axes: xlabel='Sex', ylabel='count'>



In [17]:
```
plt.figure(figsize=(6, 4))
sns.countplot(x='Pclass', hue='Survived', data=data, palette='flare')
plt.title('Survival Distribution by Passenger Class')
plt.show()
```



In [18]:
```
data['Embarked'].value_counts()
# 2-S, 0-C, 1-Q
```

Out[18]:
```
Embarked
S    270
C    102
Q     46
Name: count, dtype: int64
```

In [19]:
```
data['Sex'].value_counts()
# male - 1, female - 0
```

Out[19]:
```
Sex
male      266
female    152
Name: count, dtype: int64
```

In [20]:
```
# convert categorical data to numerical data
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
data['Embarked'] = label_encoder.fit_transform(data['Embarked'])
data['Sex'] = label_encoder.fit_transform(data['Sex'])
```

In [21]:
```
data
```

Out[21]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	1	34.50000	0	0	7.8292	1
1	1	3	0	47.00000	1	0	7.0000	2
2	0	2	1	62.00000	0	0	9.6875	1
3	0	3	1	27.00000	0	0	8.6625	2
4	1	3	0	22.00000	1	1	12.2875	2
...	...	...	...	...	...	...	...	...
413	0	3	1	30.27259	0	0	8.0500	2
414	1	1	0	39.00000	0	0	108.9000	0
415	0	3	1	38.50000	0	0	7.2500	2
416	0	3	1	30.27259	0	0	8.0500	2
417	0	3	1	30.27259	1	1	22.3583	0

418 rows × 8 columns

Separating data

In [22]:
```
x = data[['Pclass','Sex','Age','SibSp','Parch','Fare','Embarked']]
y = data['Survived']
```

Data Splitting

In [23]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_st
```

Data Modelling

In [24]:
```
#applying logistic regression
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(x_train,y_train)
```

Out[24]:
```
▾ LogisticRegression
LogisticRegression()
```

In [25]:
```
predict = lr.predict(x_test)
```

In [26]:
```
#testing
```

In [27]:
```
from sklearn.metrics import confusion_matrix
pd.DataFrame(confusion_matrix(y_test,predict),columns=['Predicted No','Predicted Ye
```

Out[27]:

	Predicted No	Predicted Yes
Actual No	92	0
Actual Yes	0	46

In [35]:
```
from sklearn.metrics import classification_report,accuracy_score,accuracy_score
print(classification_report(y_test,predict))
accuracy = accuracy_score(y_test,predict)
print('Accuracy is: ',accuracy)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        92
           1       1.00      1.00      1.00        46

    accuracy                           1.00       138
   macro avg       1.00      1.00      1.00       138
weighted avg       1.00      1.00      1.00       138

Accuracy is:  1.0
```