The sinking of the Titanic is one of the most tragic tragedies in history. The tragedy took place on April 15th, 1912. The Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers. The numbers of survivors were low due to lack of lifeboats for all passengers. Some passengers were more likely to survive than others, such as women, children, and upper-class. This case study analyzes what sorts of people were likely to survive this tragedy. The dataset includes the following:

- Pclass: Ticket class (1 = 1st, 2 = 2nd, 3 = 3rd)
- Sex: Sex
- Age: Age in years
- Sibsp: # of siblings / spouses aboard the Titanic
- Parch: # of parents / children aboard the Titanic
- Ticket: Ticket number
- Fare: Passenger fare
- Cabin: Cabin number
- Embarked: Port of Embarkation C = Cherbourg, Q = Queenstown, S = Southampton

- Target class: Survived: (0 = No, 1 = Yes)

Source: https://commons.wikimedia.org/wiki/File:Titanic_II.jpg
Data Source: https://www.kaggle.com/c/titanic

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

```
# Read the data using pandas dataframe
titanic_df = pd.read_csv('/content/titanic.csv')
```

```
# Show the data head!
titanic_df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S

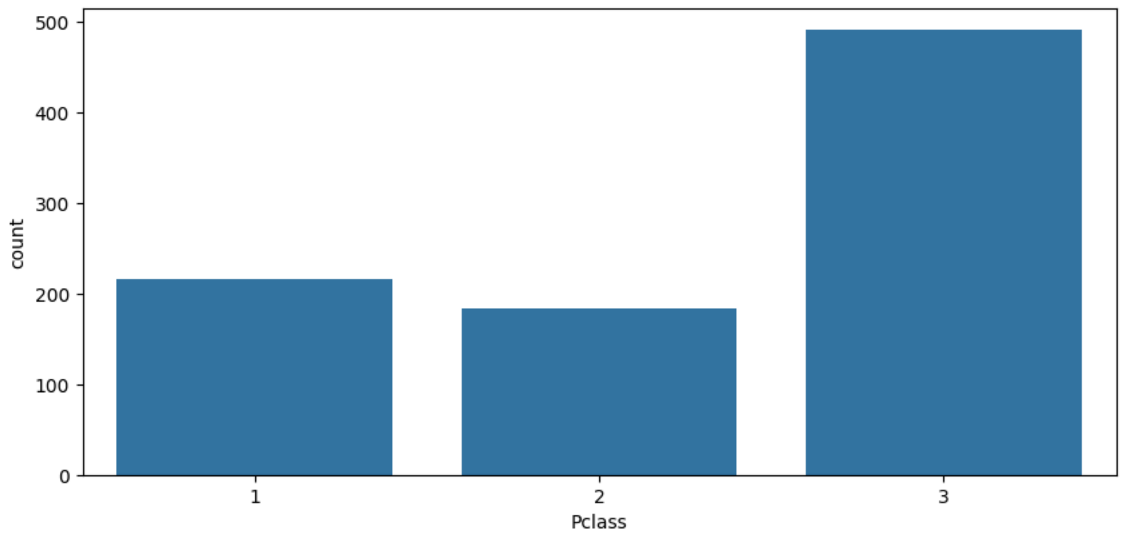
```
# Let's count the number of survivors and non-survivors
survived_df = titanic_df[titanic_df['Survived'] == 1]
no_survived_df = titanic_df[titanic_df['Survived'] == 0]
```

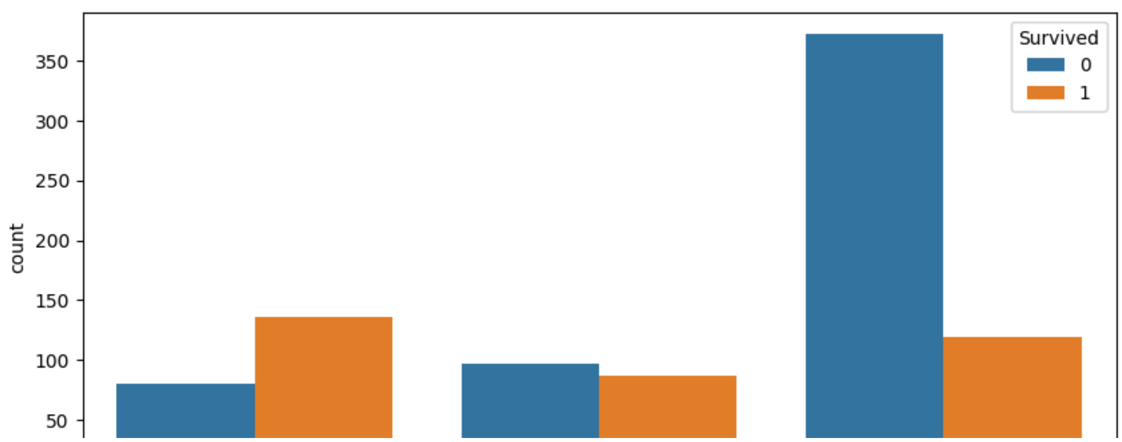
```
# Count the survived and deceased
print("Total =", len(titanic_df))

print("Number of passengers who survived =", len(survived_df))
print("Percentage Survived =", 1. * len(survived_df) / len(titanic_df) * 100.0, "%")

print("Number of passengers who did not Survive =", len(no_survived_df))
print("Percentage who did not survive =", 1. * len(no_survived_df) / len(titanic_df) * 100.0, "%")
```

```
Total = 891
Number of passengers who survived = 342
Percentage Survived = 38.38383838383838 %
Number of passengers who did not Survive = 549
Percentage who did not survive = 61.61616161616161 %
```

```
# Bar Chart to indicate the number of people who survived based on their class
# If you are a first class, you have a higher chance of survival
plt.figure(figsize = [10, 10])
plt.subplot(211)
sns.countplot(x = 'Pclass', data = titanic_df)
plt.subplot(212)
sns.countplot(x = 'Pclass', hue = 'Survived', data = titanic_df)
```

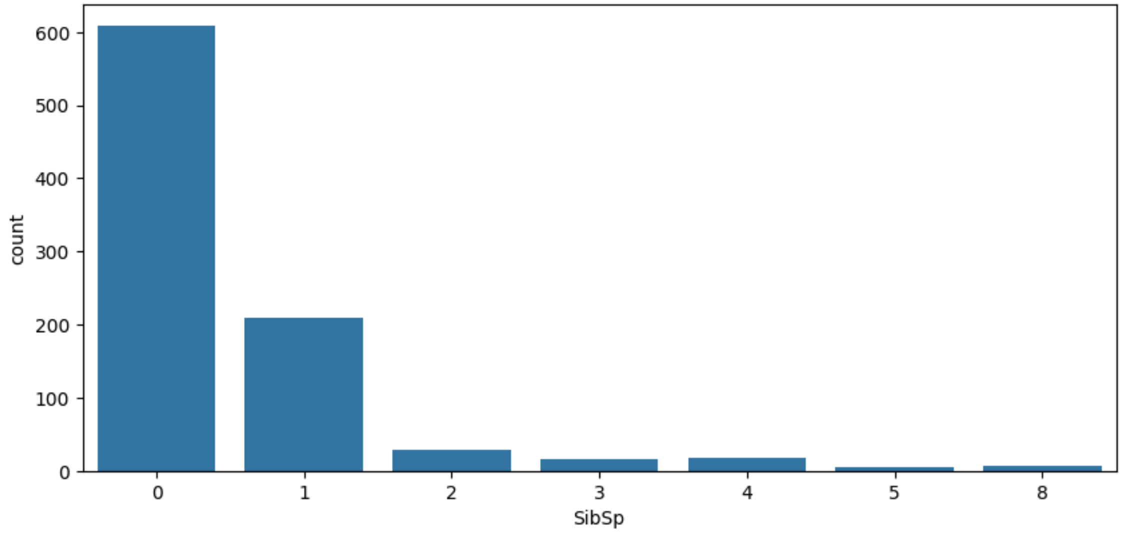
```
<Axes: xlabel='Pclass', ylabel='count'>
```



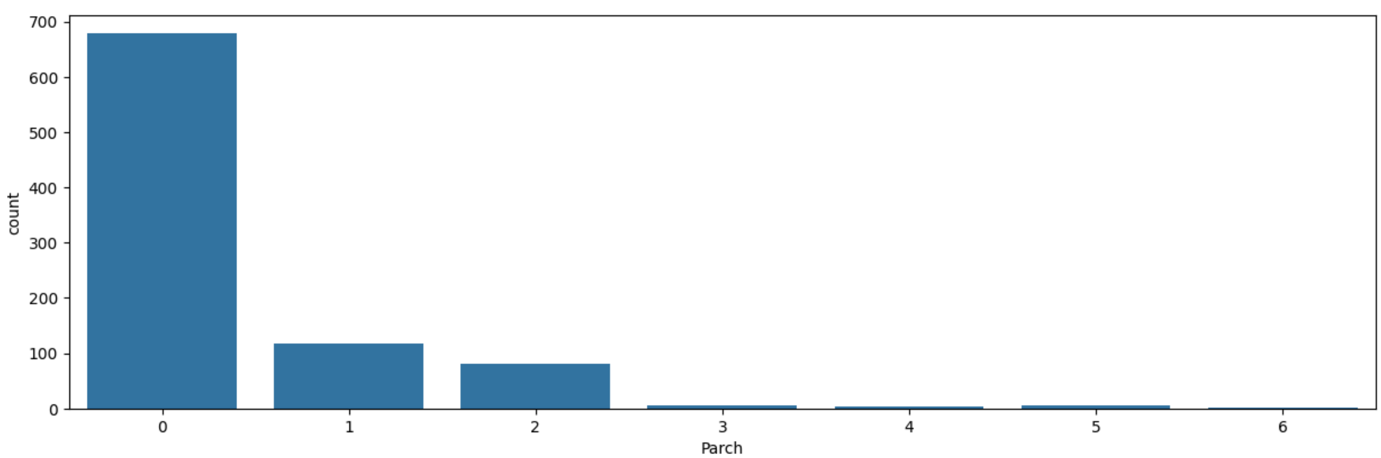



```
# Bar Chart to indicate the number of people survived based on their siblings status
# If you have 1 siblings (SibSp = 1), you have a higher chance of survival compared to being alone (SibSp = 0)
plt.figure(figsize = [10, 10])
plt.subplot(211)
sns.countplot(x = 'SibSp', data = titanic_df)
plt.subplot(212)
sns.countplot(x = 'SibSp', hue = 'Survived', data = titanic_df)
```

```
<Axes: xlabel='SibSp', ylabel='count'>
```

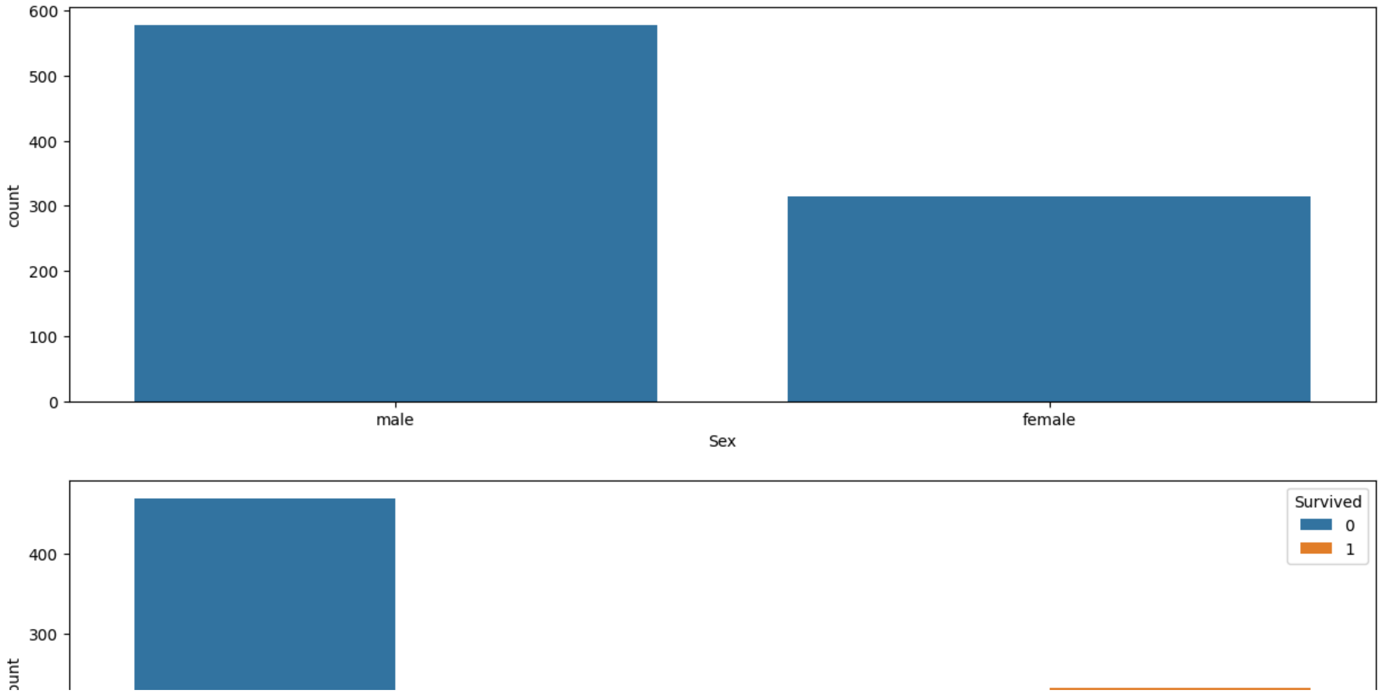



```
# Bar Chart to indicate the number of people survived based on their Parch status (how many parents onboard)
# If you have 1, 2, or 3 family members (Parch = 1,2), you have a higher chance of survival compared to being alone (Parch = 0)
plt.figure(figsize = [15, 10])
plt.subplot(211)
sns.countplot(x = 'Parch', data = titanic_df)
plt.subplot(212)
sns.countplot(x = 'Parch', hue = 'Survived', data = titanic_df);
```



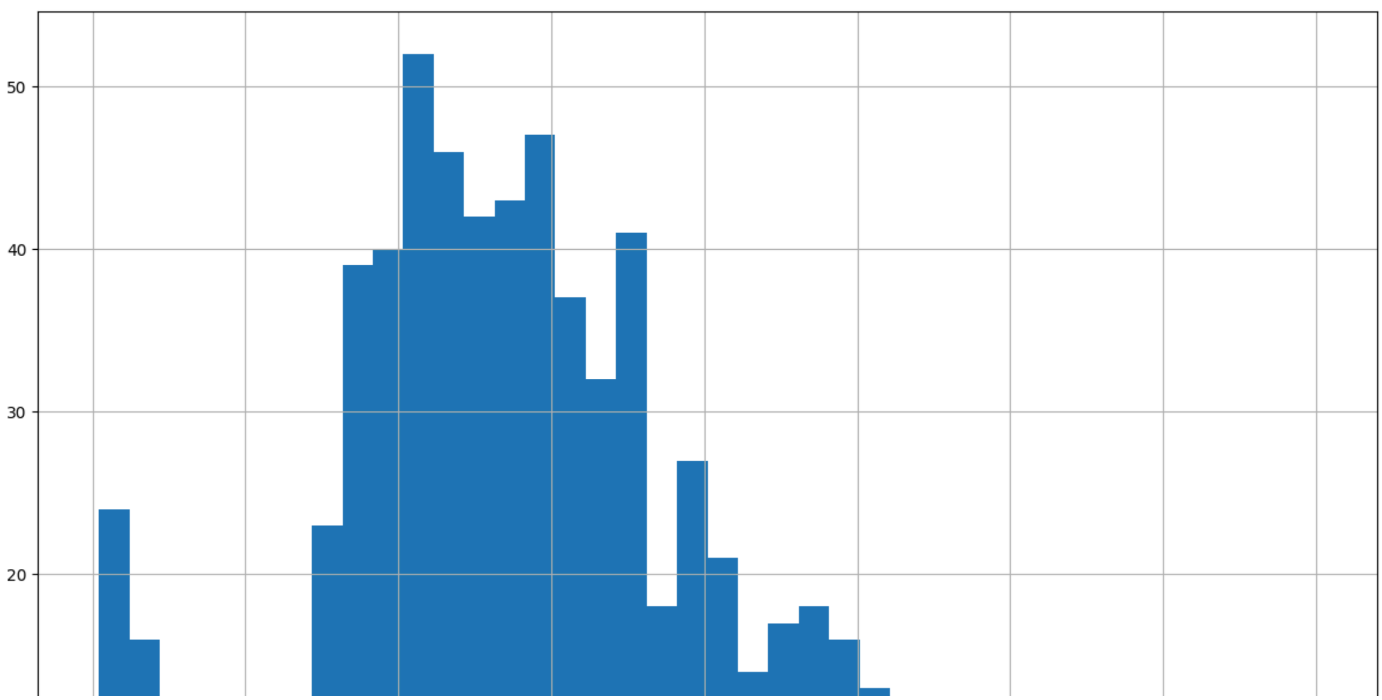
```
# Bar Chart to indicate the number of people survived based on their sex
# If you are a female, you have a higher chance of survival compared to other ports!
plt.figure(figsize = [15, 10])
plt.subplot(211)
sns.countplot(x = 'Sex', data = titanic_df)
plt.subplot(212)
sns.countplot(x = 'Sex', hue = 'Survived', data = titanic_df)
```

```
<Axes: xlabel='Sex', ylabel='count'>
```



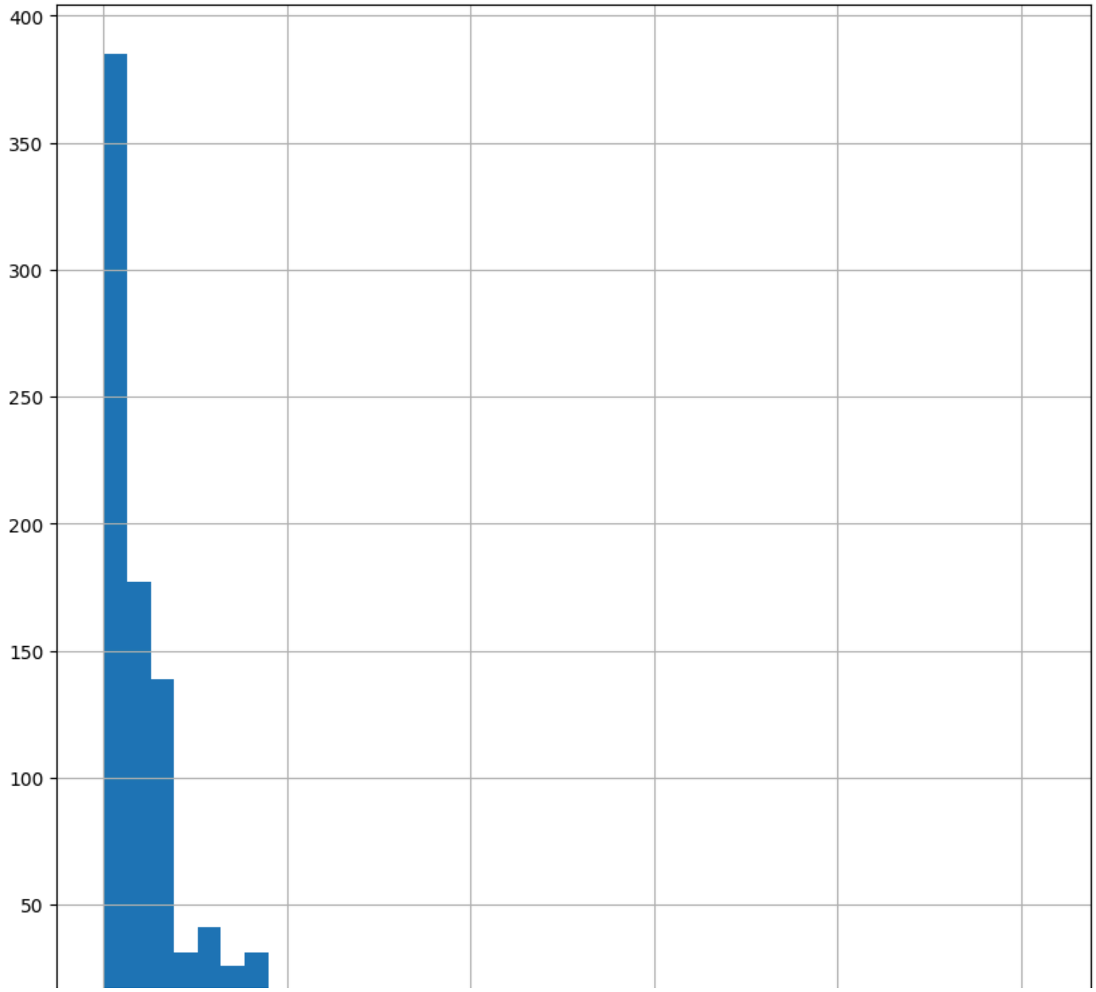

```
# Age Histogram
plt.figure(figsize=[15, 10])
titanic_df['Age'].hist(bins = 40)
```

```
<Axes: >
```



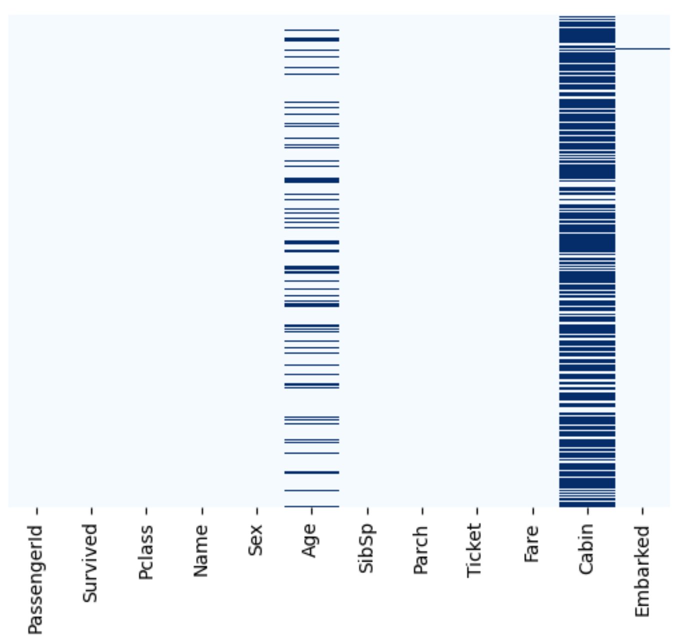

```
# Fare Histogram
plt.figure(figsize=[10, 10])
titanic_df['Fare'].hist(bins = 40)
```

<Axes: >




```
# Let's explore which dataset is missing
sns.heatmap(titanic_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")
```

<Axes: >



```
# Let's drop the cabin coloumn and test with inplace = true and false
titanic_df.drop('Cabin', axis = 1, inplace = True)
```

```
# Let's drop the embarked, Ticket, passengerID, and Name as well
titanic_df.drop(['Name', 'Ticket', 'Embarked', 'PassengerId'], axis = 1, inplace = True)
```

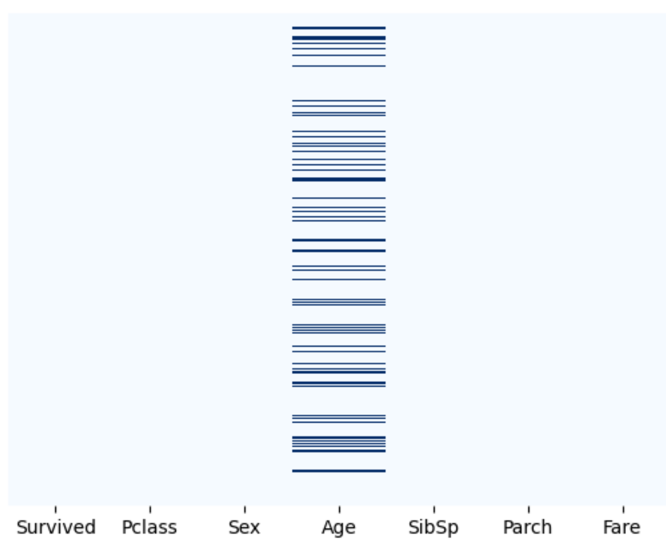
```
titanic_df
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare
0	0	3	male	22.0	1	0	7.2500
1	1	1	female	38.0	1	0	71.2833
2	1	3	female	26.0	0	0	7.9250
3	1	1	female	35.0	1	0	53.1000
4	0	3	male	35.0	0	0	8.0500
...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000
887	1	1	female	19.0	0	0	30.0000
888	0	3	female	NaN	1	2	23.4500
889	1	1	male	26.0	0	0	30.0000
890	0	3	male	32.0	0	0	7.7500

891 rows × 7 columns

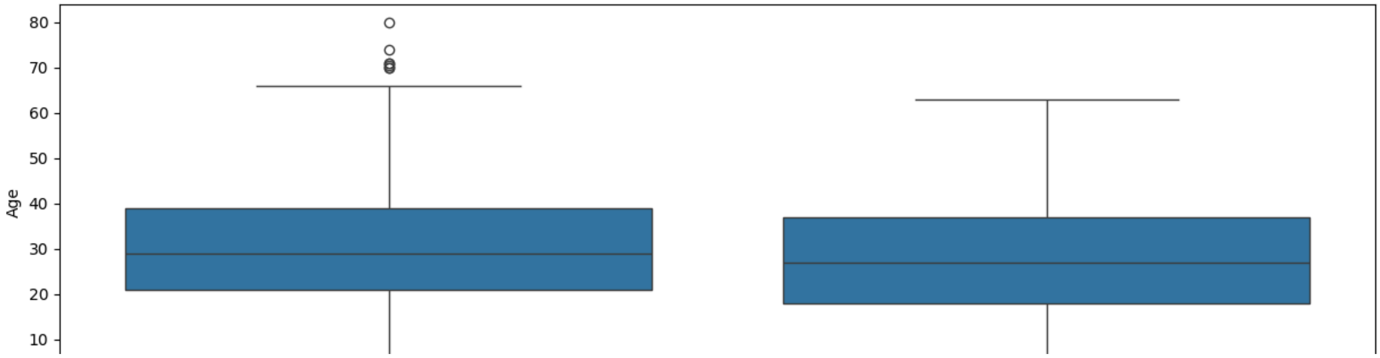
```
# Let's view the data one more time!
sns.heatmap(titanic_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")
```

<Axes: >



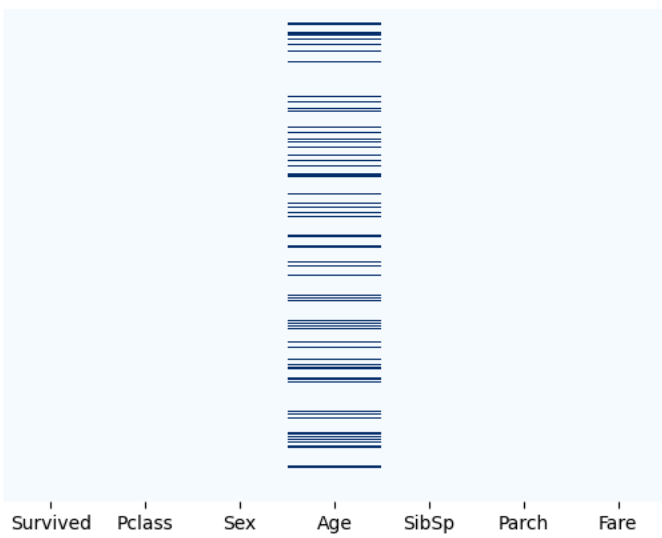

```
# Let's get the average age for male (~29) and female (~25)
plt.figure(figsize=(15, 10))
plt.subplot(211)
sns.boxplot(x = 'Sex', y = 'Age', data = titanic_df)
```

```
<Axes: xlabel='Sex', ylabel='Age'>
```




```
def fill_age(data):
    """Fills missing age values based on sex, Male=29, Female=25.

    Args:
        data: A pandas Series containing 'Age' and 'Sex' columns.

    Returns:
        A pandas Series with filled age values.
    """
    age = data['Age']
    sex = data['Sex']

    # Use loc to fill NaN values based on sex
    data.loc[pd.isnull(age) & (sex == 'male'), 'Age'] = 29
    data.loc[pd.isnull(age) & (sex == 'female'), 'Age'] = 25

    return data


# Let's view the data one more time!
sns.heatmap(titanic_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")
```

```
<Axes: >
```




```
# You just need one column only to represent male or female
pd.get_dummies(titanic_df['Sex'])
```

	female	male
0	False	True
1	True	False
2	True	False
3	True	False
4	False	True
...	...	...
886	False	True
887	True	False
888	True	False
889	False	True
890	False	True

891 rows × 2 columns

```
male = pd.get_dummies(titanic_df['Sex'], drop_first = True)
```

```
# first let's drop the embarked and sex
titanic_df.drop(['Sex'], axis = 1, inplace = True)
```

```
titanic_df
```

	Survived	Pclass	Age	SibSp	Parch	Fare
0	0	3	22.0	1	0	7.2500
1	1	1	38.0	1	0	71.2833
2	1	3	26.0	0	0	7.9250
3	1	1	35.0	1	0	53.1000
4	0	3	35.0	0	0	8.0500
...	...	...	...	...	...	...
886	0	2	27.0	0	0	13.0000
887	1	1	19.0	0	0	30.0000
888	0	3	NaN	1	2	23.4500
889	1	1	26.0	0	0	30.0000
890	0	3	32.0	0	0	7.7500

891 rows × 6 columns

```
# Now let's add the encoded column male again
titanic_df = pd.concat([titanic_df, male], axis = 1)
```

```
titanic_df
```

	Survived	Pclass	Age	SibSp	Parch	Fare	male
0	0	3	22.0	1	0	7.2500	True
1	1	1	38.0	1	0	71.2833	False
2	1	3	26.0	0	0	7.9250	False
3	1	1	35.0	1	0	53.1000	False
4	0	3	35.0	0	0	8.0500	True
...	...	...	...	...	...	...	...
886	0	2	27.0	0	0	13.0000	True
887	1	1	19.0	0	0	30.0000	False
888	0	3	NaN	1	2	23.4500	False
889	1	1	26.0	0	0	30.0000	True
890	0	3	32.0	0	0	7.7500	True

891 rows × 7 columns

Train Logistic Regression Classifier Model

```
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
# 1. Create an imputer object with a strategy (e.g., mean, median)
imputer = SimpleImputer(strategy='mean') # or strategy='median'


#Let's drop the target coloumn before we do train test split
X = titanic_df.drop('Survived', axis = 1).values
y = titanic_df['Survived'].values


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10)


# 3. Fit the imputer on the training data and transform both train and test sets
X_train = imputer.fit_transform(X_train) # Fit on training data and transform
X_test = imputer.transform(X_test)       # Transform the test data



# 4. Now, fit your Logistic Regression model
classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(random_state=0)
```

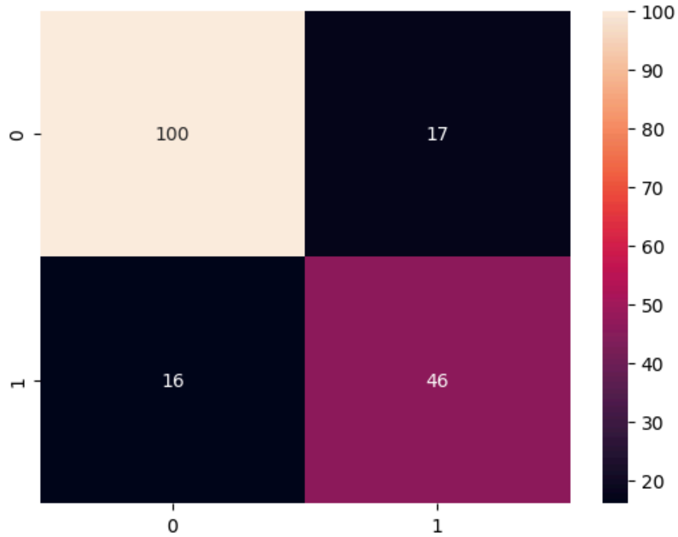
Assess Trained Model Performance

```
#Assess Trained Model Performance
y_predict_test = classifier.predict(X_test)
y_predict_test
```

```
array([0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0,
       1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 1])
```

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_predict_test)
sns.heatmap(cm, annot = True, fmt = "d")
```

```
<Axes: >
```




```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_predict_test))
```

```
              precision    recall  f1-score   support

           0       0.86      0.85      0.86       117
           1       0.73      0.74      0.74        62

    accuracy                           0.82       179
   macro avg       0.80      0.80      0.80       179
weighted avg       0.82      0.82      0.82       179
```

Fitting Naive Bayes Classifier Model

```
# Fitting Naive Bayes Classifier Model
from sklearn.naive_bayes import MultinomialNB
classifier = MultinomialNB()
classifier.fit(X_train, y_train)
```

```
▾ MultinomialNB ⓘ ?
MultinomialNB()
```

```
y_predict_test = classifier.predict(X_test)
y_predict_test

cm = confusion_matrix(y_test, y_predict_test)
sns.heatmap(cm, annot = True, fmt = "d")

print(classification_report(y_test, y_predict_test))
```

```
              precision    recall  f1-score   support

           0       0.79      0.89      0.84       117
           1       0.72      0.55      0.62        62
```