Logistic Regression

Import Packages

In [1]:
```
# Import Library for Working with Tabular Data
import pandas as pd
# Import Library for Numerical Computing
import numpy as np
# Import Library for Data Visualization
import matplotlib.pyplot as plt
# Import Another Library for Data Visualizations
# This makes it easier to create beautiful data visualizations using matplotlib.
import seaborn as sns
```

In [2]:
```
# matplotlib visualizations will embed themselves
# directly in our Jupyter Notebook. This will make them easier to
# access and interpret.
%matplotlib inline
```

Import Titanic Dataset

In [3]:
```
titanic_data = pd.read_csv('titanic_train.csv')
```

In [4]:
```
display(titanic_data.head())
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabi
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C8
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C12
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN

In [5]:
```
display(titanic_data.columns)
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

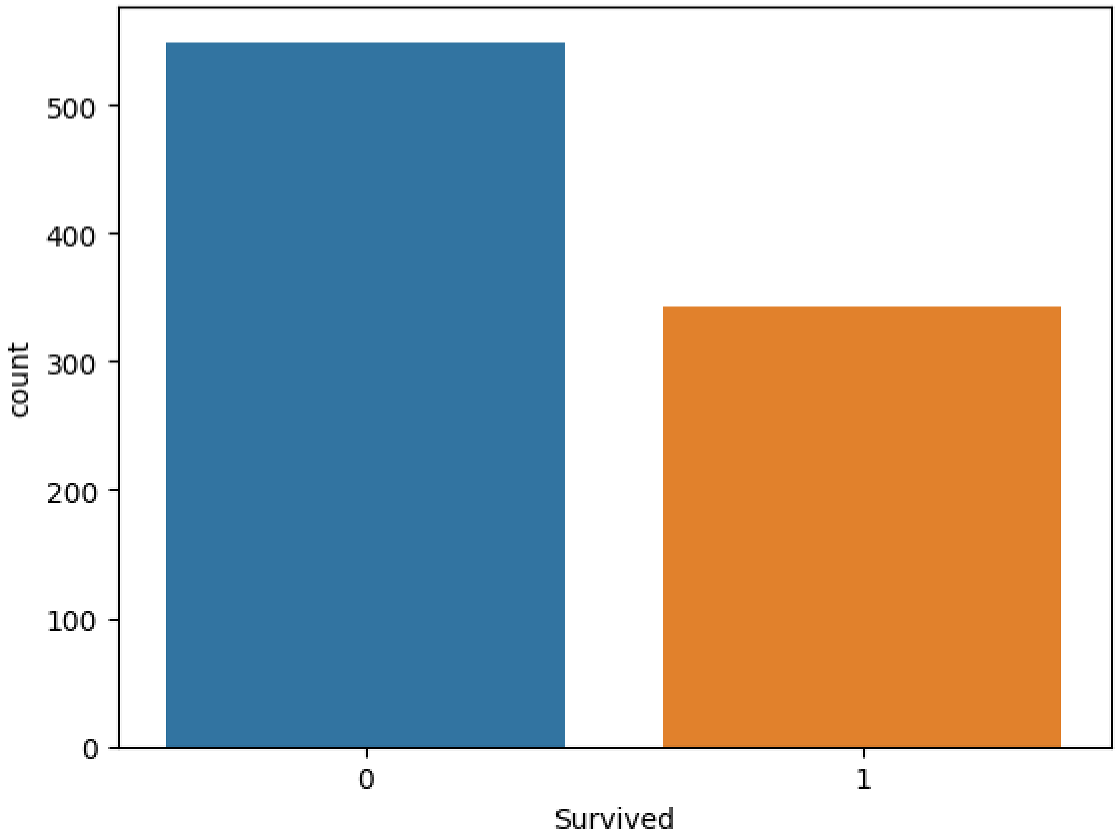
For the `Survived` feature, the variable will hold a value of *1* if the passenger survived and *0* if the passenger didn't survive.

Exploratory Data Analysis

It's useful to have a sense of the ratio between classification categories (i.e. how many survivors vs non-survivors in our training data).

In [6]:
```
sns.countplot(x = 'Survived', data = titanic_data)
```

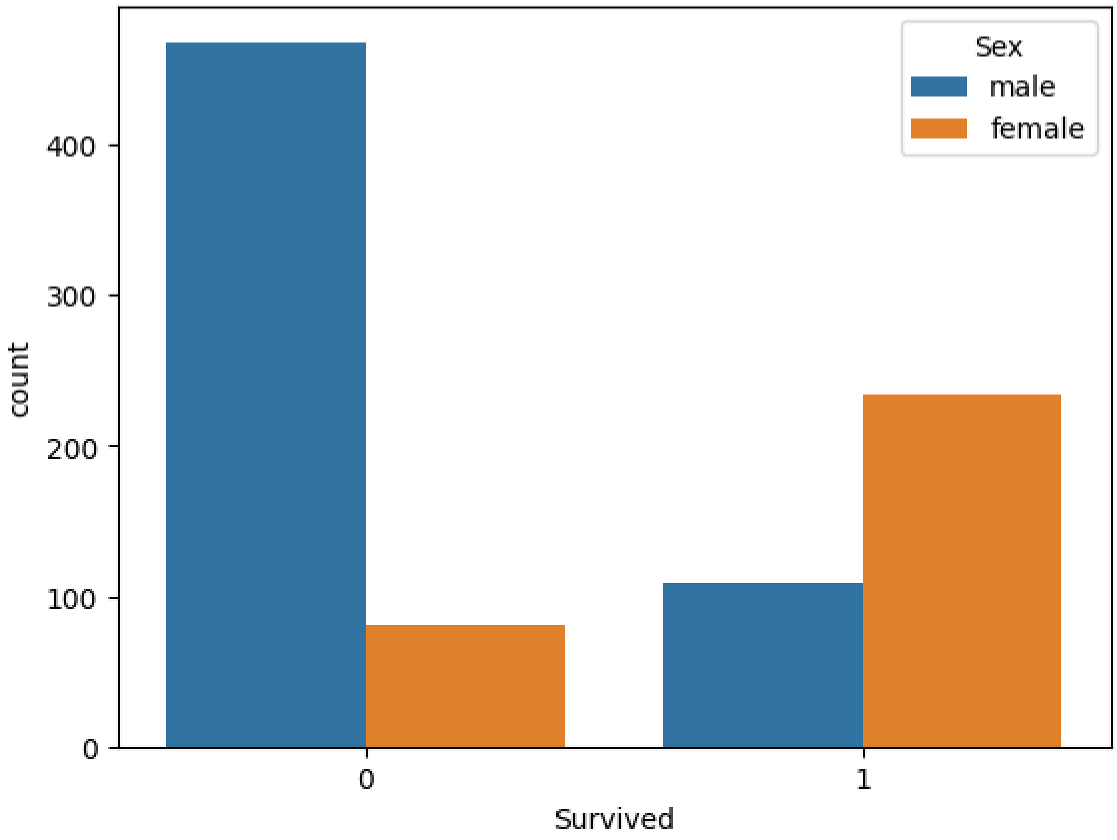
Out[6]:
```
<AxesSubplot: xlabel='Survived', ylabel='count'>
```



Lets compare the survival rates relative to some feature like *Male* and *Female* values for `Sex` Variable. We will notice that passengers with `Sex` *Male* were more likely to be non-survivors than passengers with `Sex` *Female*.

```
In [7]: sns.countplot(x = 'Survived', hue = 'Sex', data = titanic_data)
```

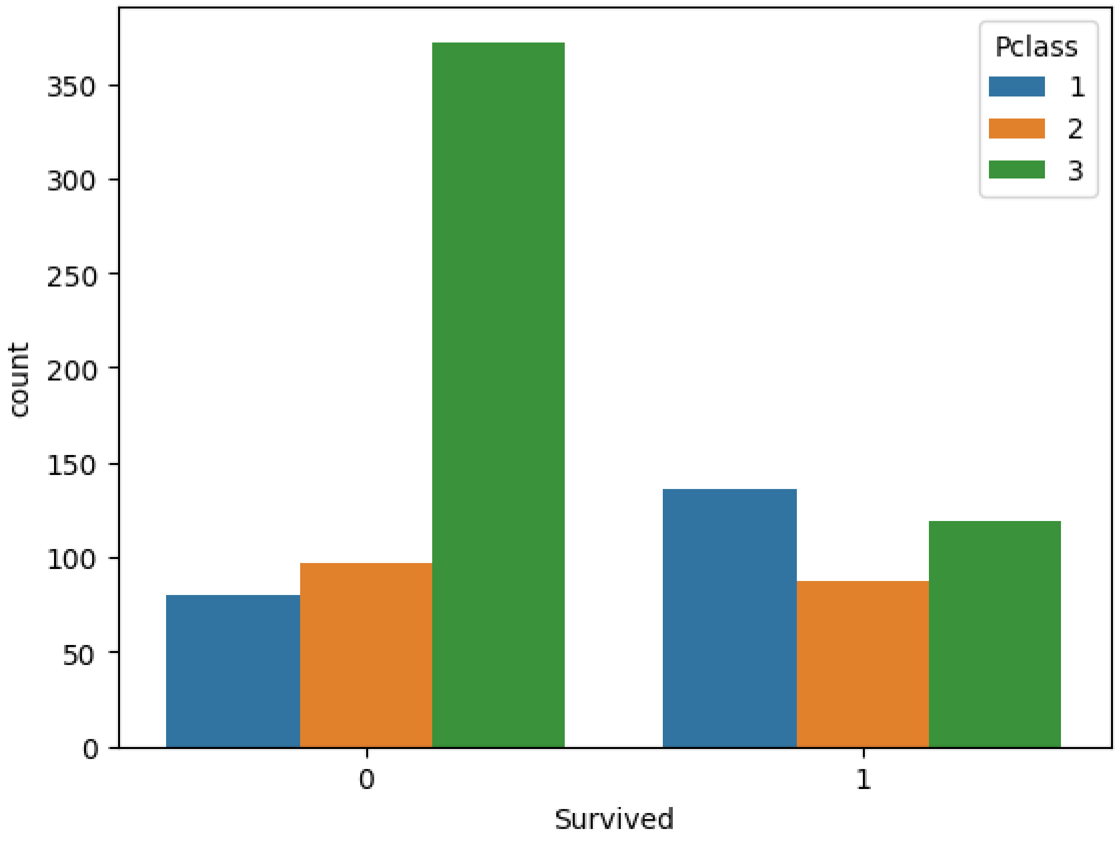
```
Out[7]: <AxesSubplot: xlabel='Survived', ylabel='count'>
```



Similarly, compare survival rates relative to `PClass` Variable.
We will notice that passengers with `PClass` value of *3* (cheapest and least luxurious class) were most likely to die.

```
In [8]: sns.countplot(x = 'Survived', hue = 'Pclass', data = titanic_data)
```

```
Out[8]: <AxesSubplot: xlabel='Survived', ylabel='count'>
```

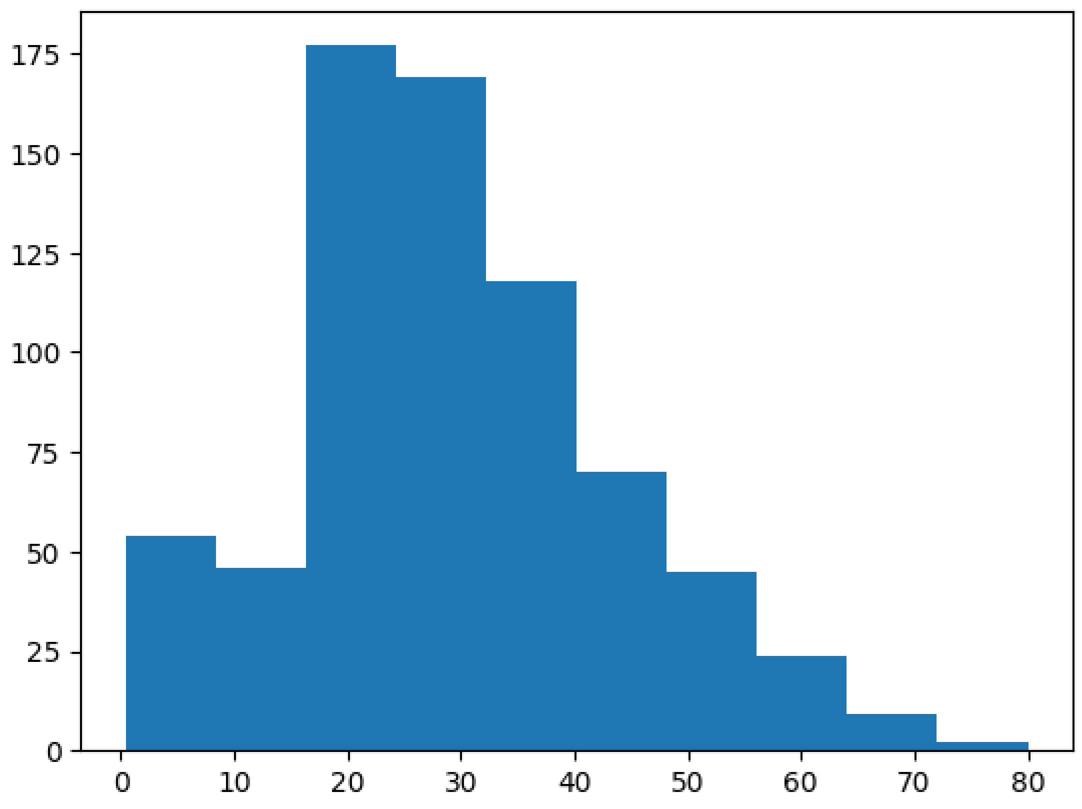


Generate a histogram to see `Age` distribution of passengers.

Use `dropna()` method to account for Dataset containing several*NULL* values. We will notice that there is a concentration of passengers with `Age` value between *20* and *40*.

```
In [9]: plt.hist(titanic_data['Age'].dropna())
```

```
Out[9]: (array([ 54.,  46., 177., 169., 118.,  70.,  45.,  24.,   9.,   2.]),
         array([ 0.42 ,  8.378, 16.336, 24.294, 32.252, 40.21 , 48.168, 56.126,
                64.084, 72.042, 80.   ]),
         <BarContainer object of 10 artists>)
```

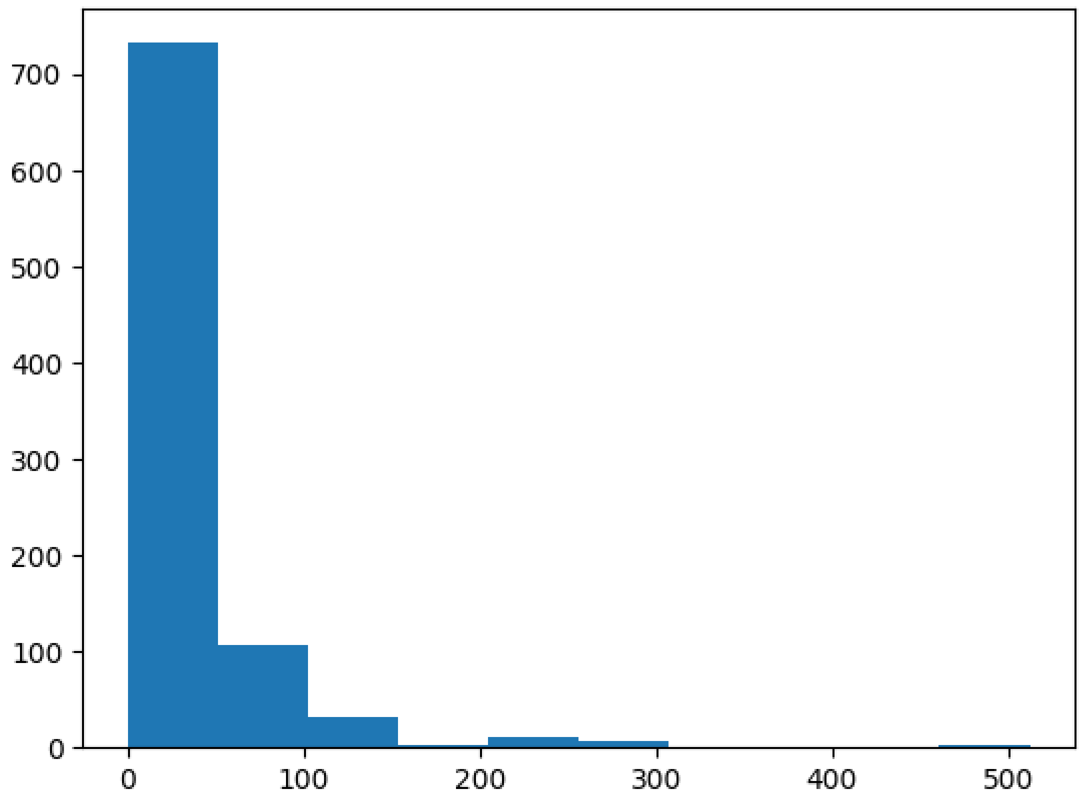


Investigate the distribution of `Fare` within Titanic Dataset.

We will notice that there are three distinct groups of `Fare` within the Dataset. This probably exhibits correlation with the different `PClass` categories.

```
In [10]: plt.hist(titanic_data['Fare'])
```

```
Out[10]: (array([732., 106.,  31.,   2.,  11.,   6.,   0.,   0.,   0.,   3.]),
          array([  0.     ,  51.23292, 102.46584, 153.69876, 204.93168, 256.1646 ,
                 307.39752, 358.63044, 409.86336, 461.09628, 512.3292 ]),
          <BarContainer object of 10 artists>)
```



Visualize Missing Data

The function `isnull()` will generate a DataFrame of *bool* values where cell contains:

- *True* if *NULL* value
- *False* otherwise

In [11]:
```
display(titanic_data.isnull())
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emba
0	False	False	False	False	False	False	False	False	False	False	True	
1	False	False	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	False	True	
3	False	False	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	False	True	
...	...	...	...	...	...	...	...	...	...	...	...	
886	False	False	False	False	False	False	False	False	False	False	True	
887	False	False	False	False	False	False	False	False	False	False	False	
888	False	False	False	False	False	True	False	False	False	False	True	
889	False	False	False	False	False	False	False	False	False	False	False	
890	False	False	False	False	False	False	False	False	False	False	True	

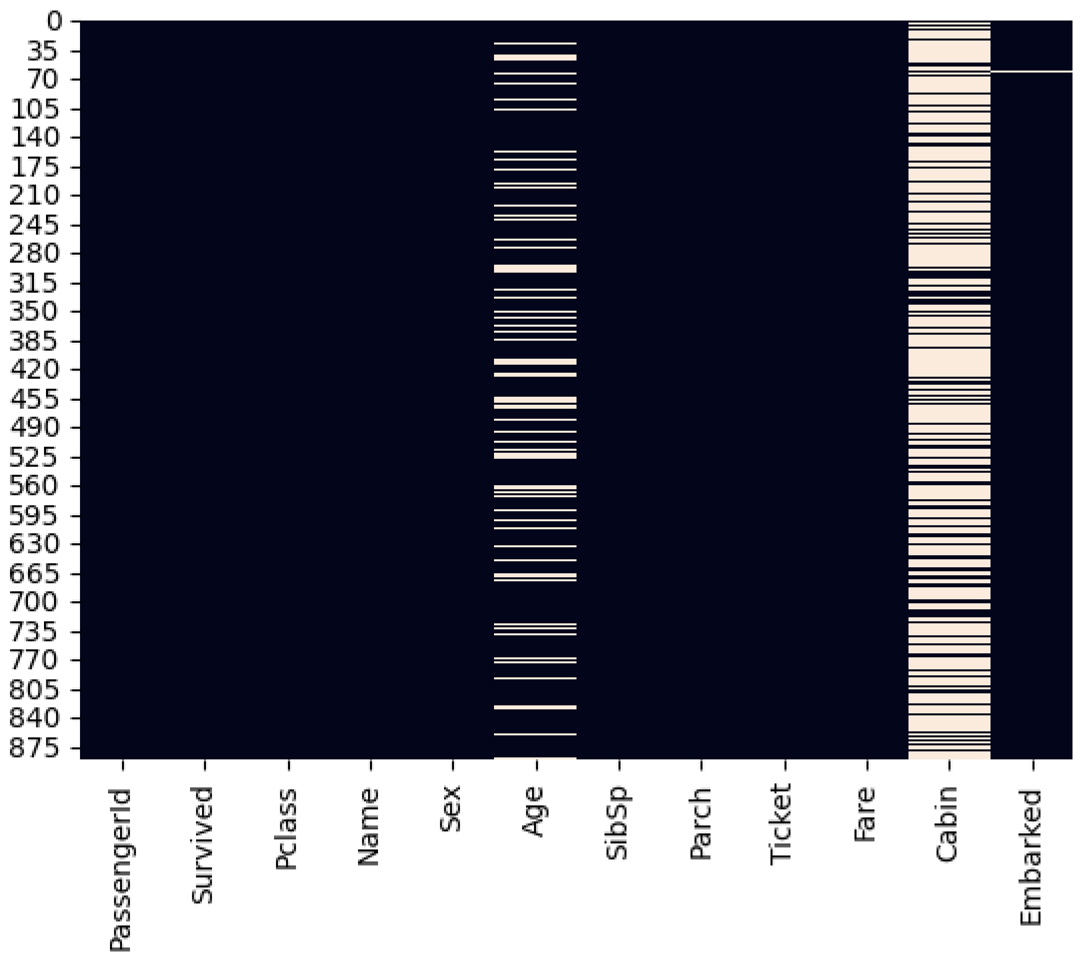
891 rows × 12 columns

A quicker visualization for assessing missing data is by using `seaborn` visualization library to create a heatmap.

- White lines indicate missing values in Dataset
  - Will notice that the majority is in the `Age` and `Cabin` columns

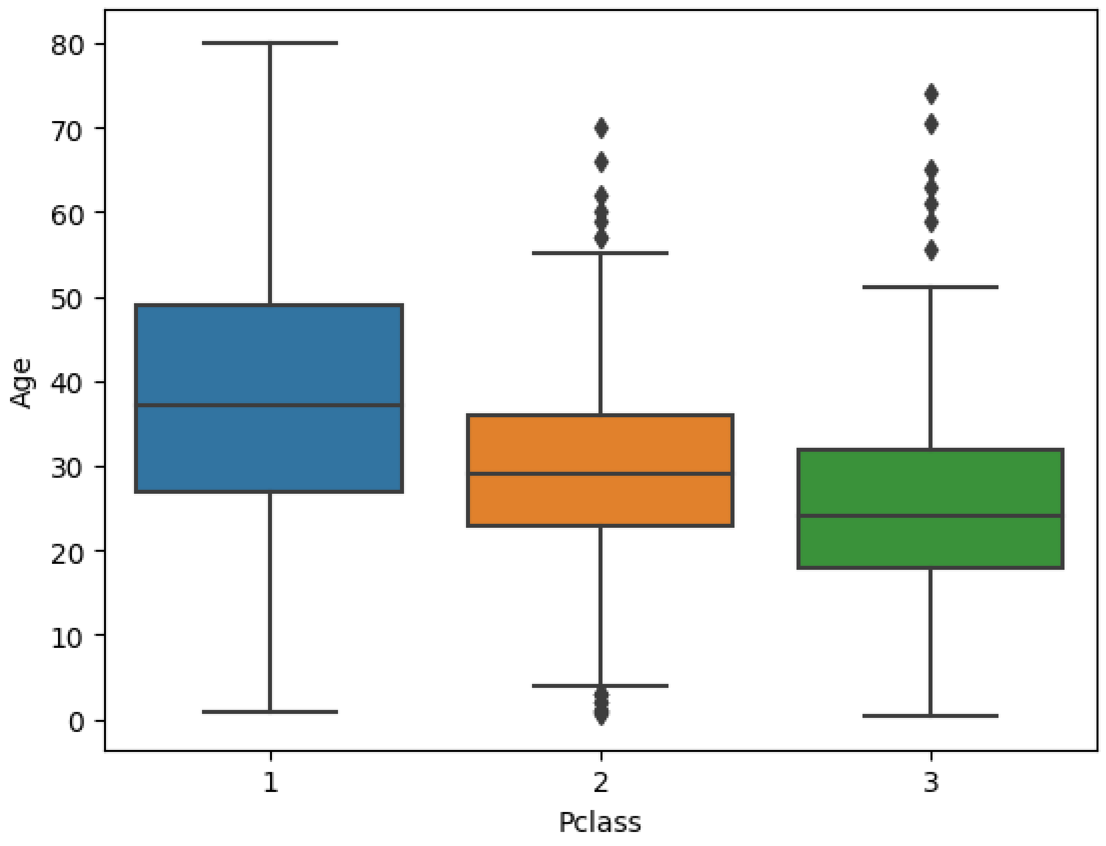
```
In [12]: sns.heatmap(titanic_data.isnull(), cbar = False)
```

```
Out[12]: <AxesSubplot: >
```



In [13]:
```
sns.boxplot(
    data = titanic_data,
    x = 'Pclass',
    y = 'Age'
)
```

Out[13]:
```
<AxesSubplot: xlabel='Pclass', ylabel='Age'>
```



Impute Missing Data

Imputation Method fills in missing `Age` values with average `Age` value for the specific `Pclass` the passenger belongs to.

Generate a boxplot of `Age` distributions in each Pclass.

- We will notice that passengers with `Pclass` value of *1* tend to be the oldest.
  - Similarly, passengers with `Pclass` value of *3* tend to be the youngest. *It is also assumed this probably correlates with* `Fare` *as well.*

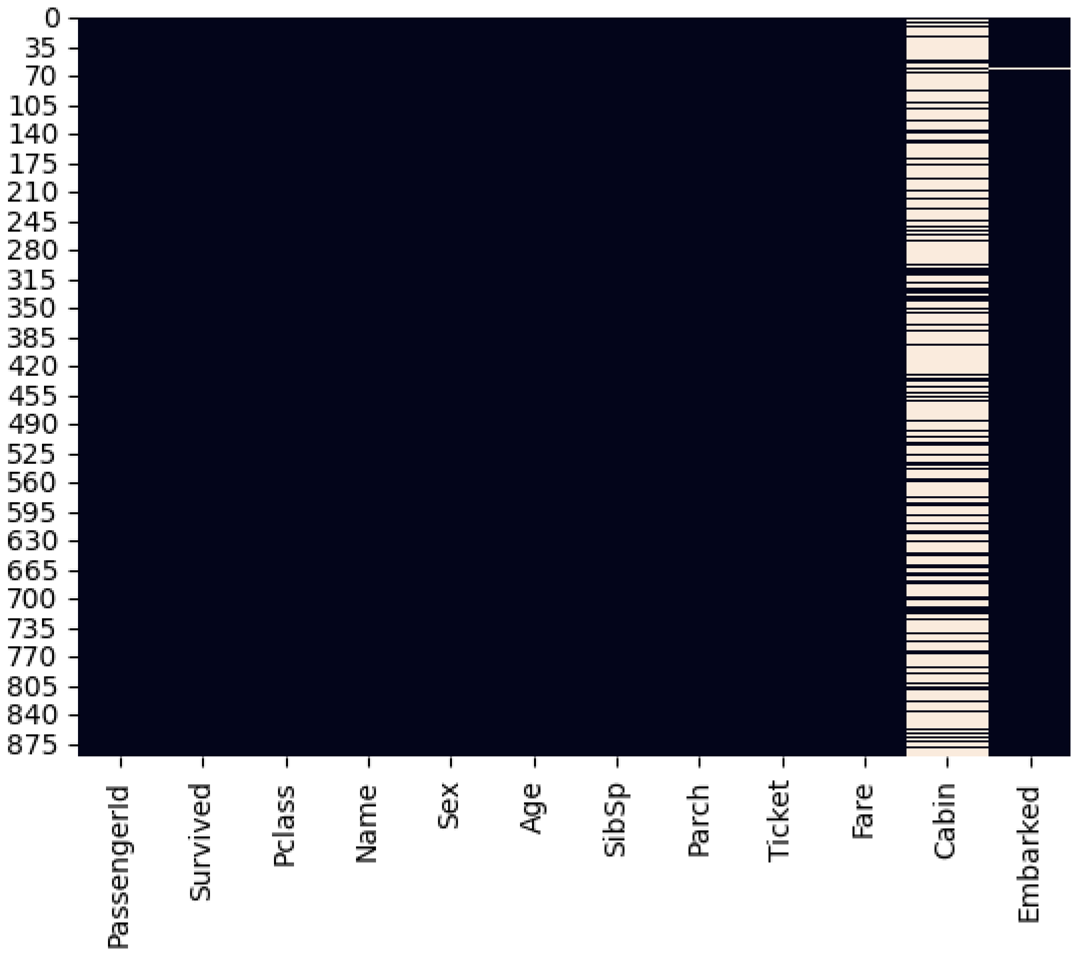
In [14]:
```python
def impute_missing_age(columns) :
    age = columns[0]
    passenger_class = columns[1]

    # Check if Age value is missing.
    if pd.isnull(age) :
        if (passenger_class == 1) :
            # Return average value of Pclass 1.
            return titanic_data[titanic_data['Pclass'] == 1]['Age'].mean()
        elif (passenger_class == 2) :
            # Return average value of Pclass 2.
            return titanic_data[titanic_data['Pclass'] == 2]['Age'].mean()
        elif (passenger_class == 3) :
            # Return average value of Pclass 3.
            return titanic_data[titanic_data['Pclass'] == 3]['Age'].mean()
    else :
        return age
```

In [15]:
```
# Apply function to every row in titanic_data DataFrame
titanic_data['Age'] = titanic_data[['Age','Pclass']].apply(impute_missing_age, axis
```

Check original heatmap to notice that `Age` column is not *NULL*.

In [16]:
```
sns.heatmap(titanic_data.isnull(), cbar = False)
```

Out[16]:
```
<AxesSubplot: >
```



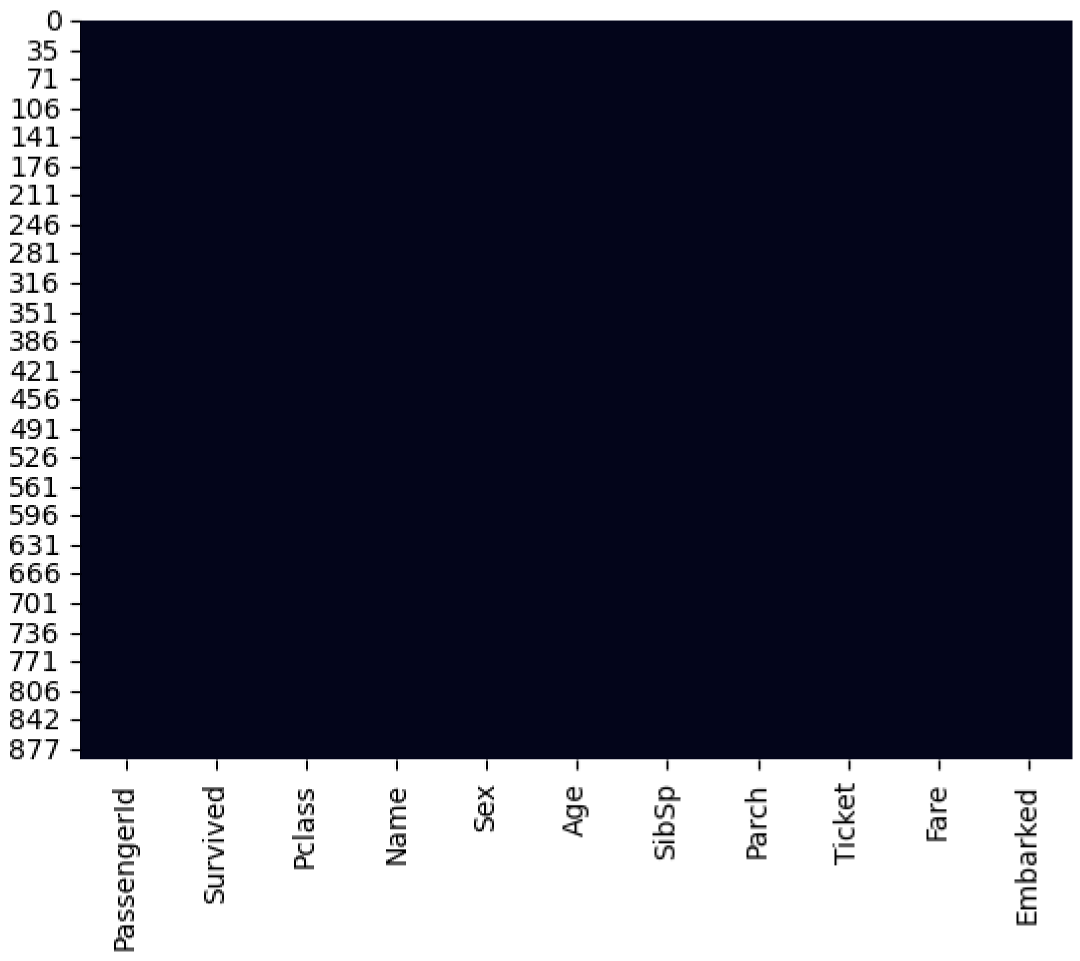
It was rather important we dealt with the missing `Age` values since this datapoint has an impact on survival for most disasters and diseases.

Remove the column with high prevalence of missing data (i.e. `Cabin`).

In [17]:
```
titanic_data.drop('Cabin', axis = 1, inplace = True)
```

In [18]:
```
# Drop all rows with missing values.
titanic_data.dropna(inplace = True)
sns.heatmap(titanic_data.isnull(), cbar = False)
```

Out[18]:
```
<AxesSubplot: >
```



Numerically Encode Categorical Variables

Create dummy variables to solve this issue by creating a new column for each value in DataFrame column.

The output will create two new columns, `Male` and `Female`.
These are perfect predictors of each other and significantly reduce the predictive power of our algorithm (i.e. Multicollinearity).
(e.g. *0* in `Female` column indicates a *1* in `Male` column)

In [19]:
```
pd.get_dummies(titanic_data['Sex'])
```

Out[19]:

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

889 rows × 2 columns

In [20]:
```
# Add argument drop_first to method get_dummies to remove Multicollinearity from ou
sex_data = pd.get_dummies(titanic_data['Sex'], drop_first = True)
```

For our `embarked_data` variable, we have *2* columns (i.e. `Q` and `S`). Have removed `C` column.
Note that `Q` and `S` columns are not perfect predictors of each other.

In [21]:
```
embarked_data = pd.get_dummies(titanic_data['Embarked'], drop_first = True)
```

In [22]:
```
# Concatenate sex_data and embarked_data data columns into existing pandas DataFram
titanic_data = pd.concat([titanic_data, sex_data, embarked_data], axis = 1)
```

In [23]:
```
print(titanic_data.columns)
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Embarked', 'male', 'Q', 'S'],
      dtype='object')
```

Drop Redundant Columns

Drop original `Sex` and `Embarked` columns from DataFrame for better readability. Also, drop columns that aren't predictive of Titanic crash survival rates for same reason (i.e. `Name`, `PassengerId`, `Ticket`).

In [24]:
```
titanic_data.drop(['Name', 'Ticket', 'Sex', 'Embarked'], axis = 1, inplace = True)
```

In [25]:
```
print(titanic_data.columns)
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare',
       'male', 'Q', 'S'],
      dtype='object')
```

Split Training and Test Data

At this point, every field in the Dataset is numeric, making it an excellent candidate for a **Logistic Regression** Machine Learning Algorithm.

In [26]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

In [27]:
```
# y-values -> data we're trying to predict.
y_data = titanic_data['Survived']
# x-values -> data used to make predictions.
x_data = titanic_data.drop('Survived', axis = 1)
```

In [28]:
```
x_training_data, x_test_data, y_training_data, y_test_data = train_test_split(
    x_data, y_data,
    test_size = 0.3
)
```

Logistic Regression Model

Import the appropriate model from `scikit-learn` (i.e. **Logistic Regression**)

In [29]:
```
# Increase max_iter variable from default value of 100. Otherwise,
# Jupyter Notebook gives ConvergenceWarning.
model = LogisticRegression(max_iter = 1000).fit(x_training_data, y_training_data)
```

Make Predictions and Measure Performance

In [30]:
```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

In [31]:
```
predictions = model.predict(x_test_data)
```

In [32]:
```
metrics_1 = classification_report(y_test_data, predictions)
print(metrics_1)
```

```
              precision    recall  f1-score   support

           0       0.79      0.86      0.83       154
           1       0.79      0.69      0.74       113

    accuracy                           0.79       267
   macro avg       0.79      0.78      0.78       267
weighted avg       0.79      0.79      0.79       267
```

We can see the raw **Confusion Matrix** and calculate the performance metrics manually as well. This is a tool used to compare *True Positives*, *True Negatives*, *False Positives*, *False Negatives*.

This allows you to assess whether your model is particularly weak in a specific quadrant. You may wish to ensure that the model performs especially well in a dangerous zone of the **Confusion Matrix**. (e.g. High rate of *False Negatives* for cancer diagnosis indicates that you incorrectly predict malignant tumors to be non-malignant.)

In [33]:
```
metrics_2 = confusion_matrix(y_test_data, predictions)
print(metrics_2)
```

```
[[133  21]
 [ 35  78]]
```