Titanic Disaster Survival Using Logistic Regression

In [126]:

```
#import libraries
```

In [109]:

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

Load the Data

In [110]:

```
titanic_data = pd.read_csv('titanic_train.csv')
```

In [111]:

```
len(titanic_data)
```

Out[111]:

```
891
```

View the data using head function which returns top rows

In [112]:

```
titanic_data.head()
```

Out[112]:

	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

In [114]:

```
titanic_data.index
```

Out[114]:

```
RangeIndex(start=0, stop=891, step=1)
```

In [115]:

```
titanic_data.columns
```

Out[115]:

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

In [116]:

```
titanic_data.info()
```

```
<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
```

In [117]:

```
titanic_data.dtypes
```

Out[117]:

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

In [118]:

```
titanic_data.describe()
```

Out[118]:

	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

Explaining Dataset

survival : Survival 0 = No, 1 = Yes
pclass : Ticket class 1 = 1st, 2 = 2nd, 3 = 3rd
sex : Sex
Age : Age in years
sibsp : Number 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

Data Analysis

Import Seaborn for visually analysing the data

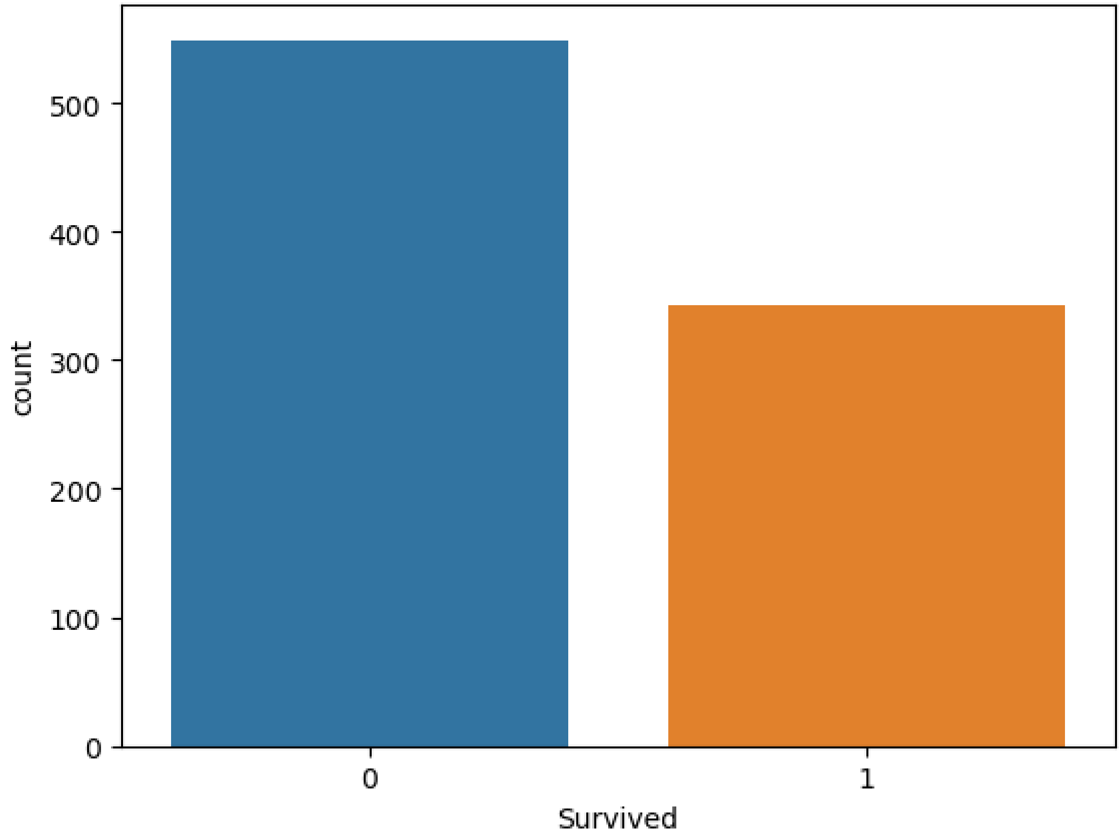
Find out how many survived vs Died using countplot method of seaboarn

In [119]:

```
#countplot of survived vs not  survived
```

In [120]:

```
sns.countplot(x='Survived',data=titanic_data)
plt.show()
```



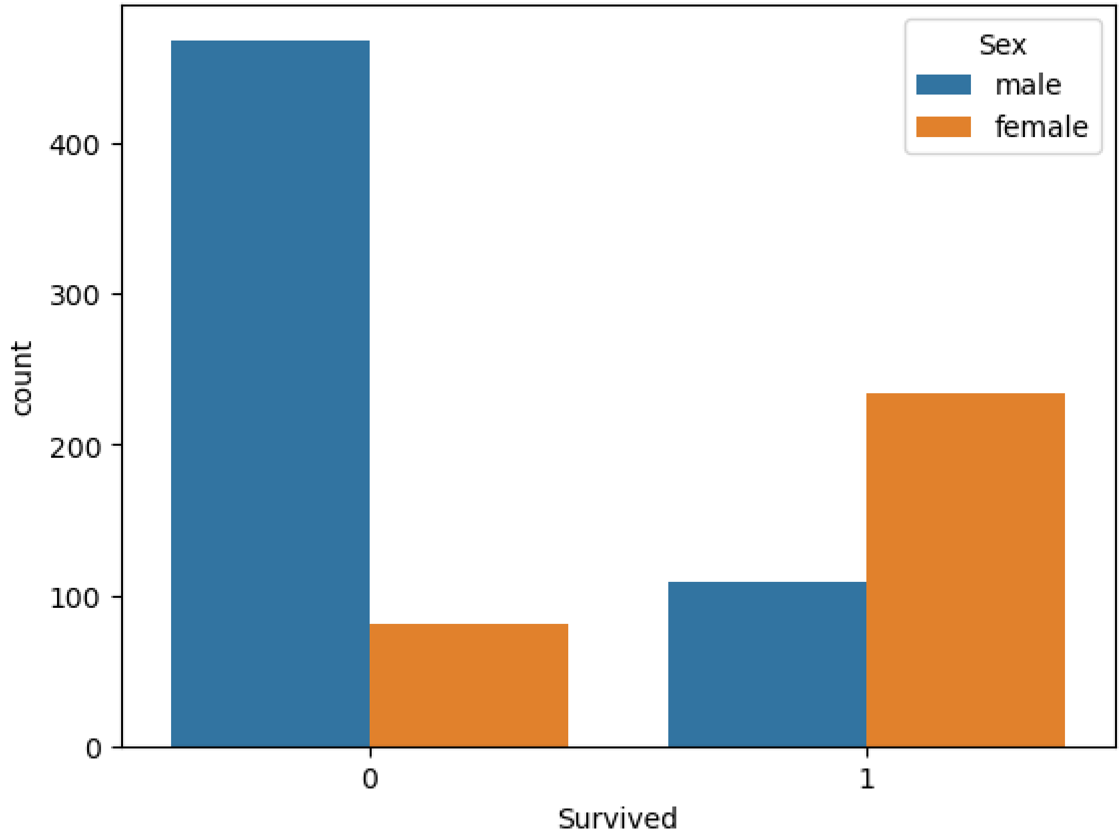
Male vs Female Survival

In [121]:

```
#Male vs Female Survived
```

In [122]:

```
sns.countplot(x = 'Survived', data = titanic_data, hue = 'Sex')
plt.show()
```



**See age group of passengeres travelled **
Note: We will use displot method to see the histogram. However some records does not have age hence the method will throw an error. In order to avoid that we will use dropna method to eliminate null values from graph

In [123]:

```
#Check for null
```

In [124]:

```
titanic_data.isna()
```

Out[124]:

	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	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

In [27]:

```
#Check how many values are null
```

In [125]:

```
titanic_data.isna().sum()
```

Out[125]:

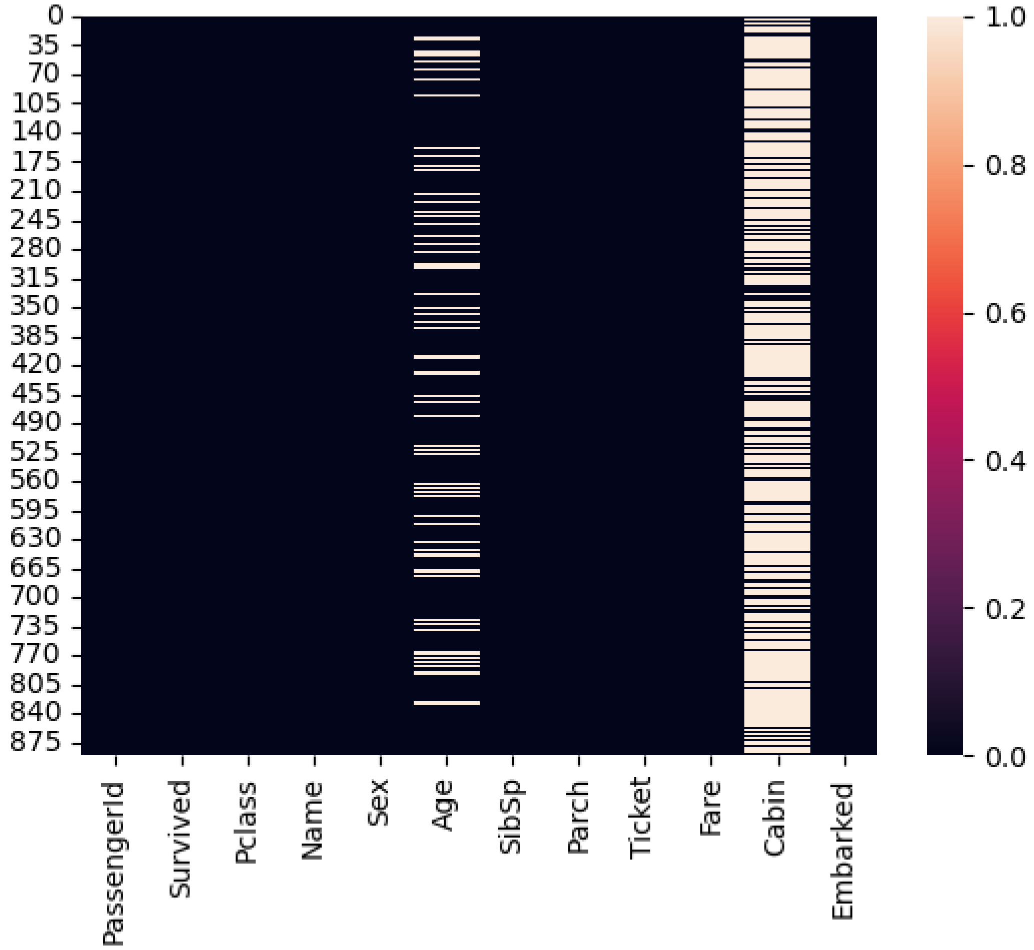
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [30]:

```
#Visualize null values
```

In [35]:

```
sns.heatmap(titanic_data.isna())
plt.show()
```



In [36]:

```
#find the % of null values in age column
```

In [44]:

```
(titanic_data['Age'].isna().sum()/len(titanic_data['Age']))*100
```

Out[44]:

```
19.865319865319865
```

In [50]:

```
#find the % of null values in cabin column
```

In [51]:

```
(titanic_data['Cabin'].isna().sum()/len(titanic_data['Cabin']))*100
```

Out[51]:

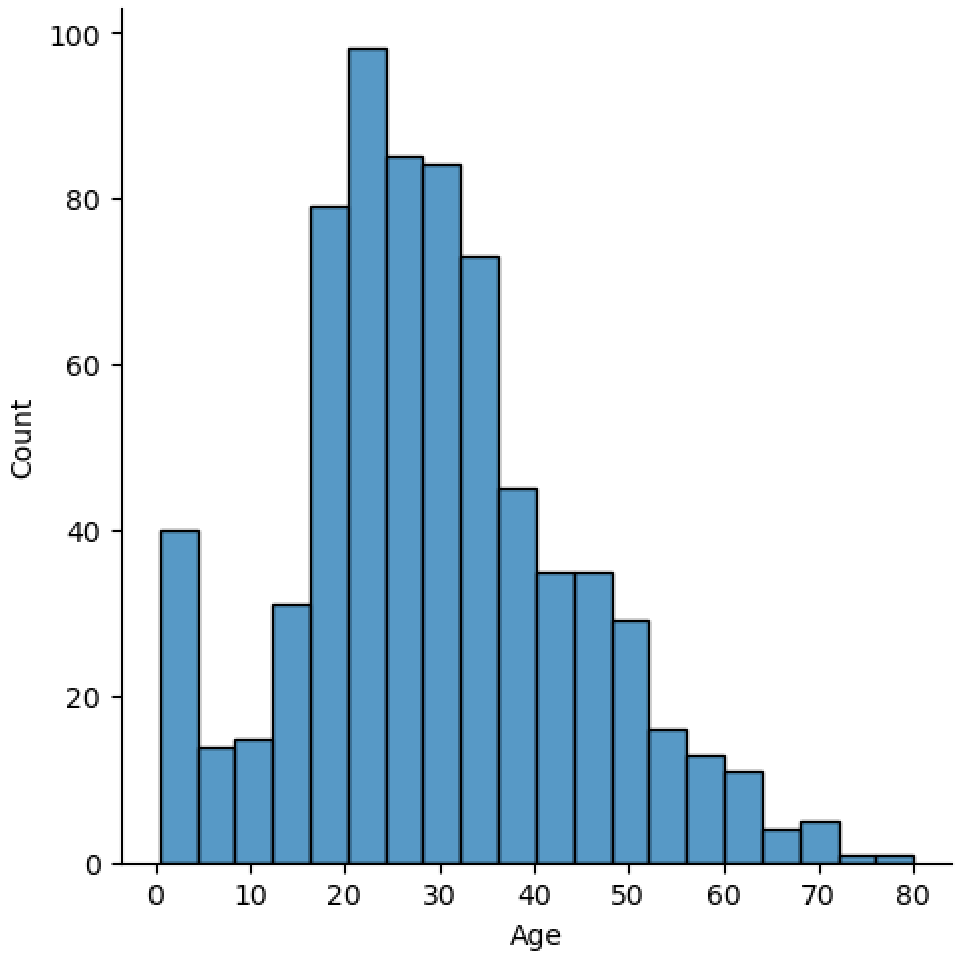
```
77.10437710437711
```

In [54]:

```
sns.displot(x='Age',data=titanic_data)
```

Out[54]:

```
<seaborn.axisgrid.FacetGrid at 0x182ec58eec0>
```



Data Cleaning

Fill the missing values

we will fill the missing values for age. In order to fill missing values we use fillna method.
For now we will fill the missing age by taking average of all age

In [55]:

```
#fill age column
```

In [58]:

```
titanic_data['Age'].fillna(titanic_data['Age'].mean(),inplace=True)
```

We can verify that no more null data exist

we will examine data by isnull mehtod which will return nothing

In [59]:

```
#verify null value
```

In [60]:

```
titanic_data['Age'].isna().sum()
```

Out[60]:

```
0
```

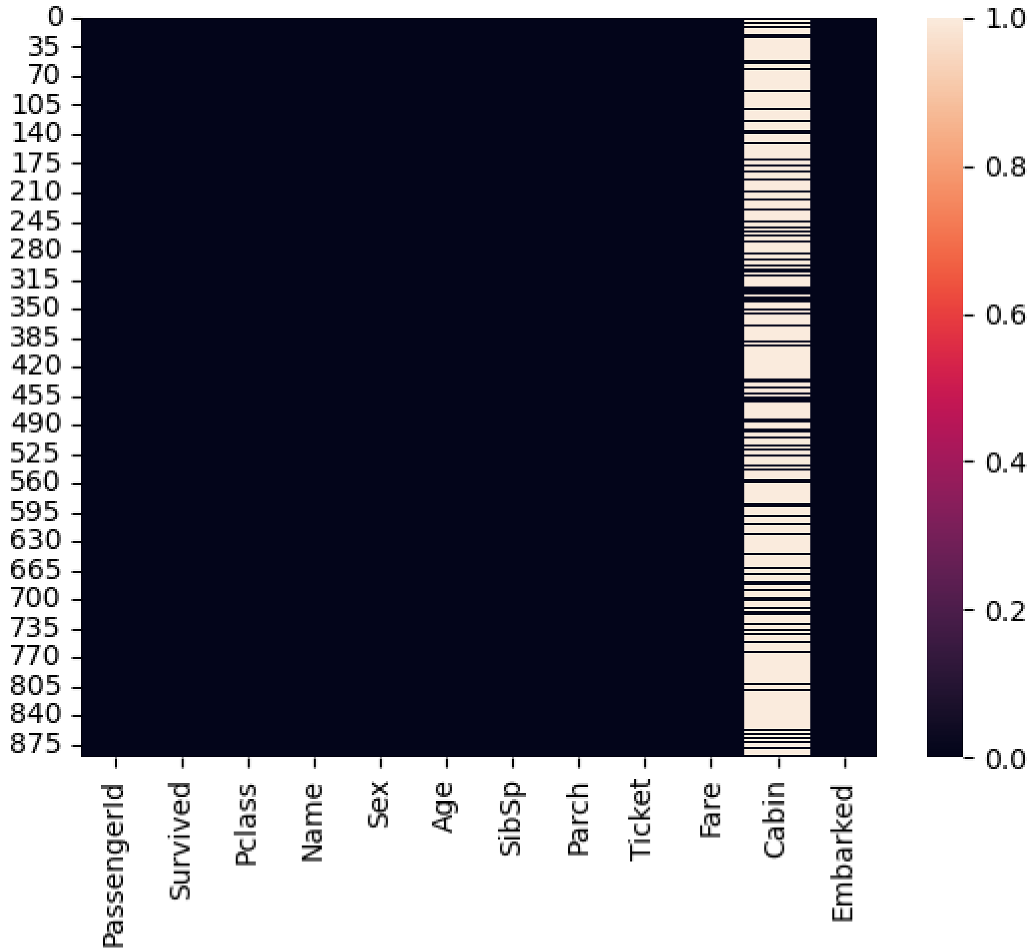
Alternatively we will visualise the null value using heatmap
we will use heatmap method by passing only records which are null.

In [61]:

```
#visualize null values
```

In [63]:

```
sns.heatmap(titanic_data.isna())
plt.show()
```



We can see cabin column has a number of null values, as such we can not use it for prediction. Hence we will drop it

In [65]:

```
#Drop cabin column
```

In [66]:

```
titanic_data.drop('Cabin',axis=1,inplace=True)
```

In [67]:

```
#see the contents of the data
```

In [69]:

```
titanic_data.head()
```

Out[69]:

	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

Preaparing Data for Model No we will require to convert all non-numerical columns to numeric. Please note this is required for feeding data into model. Lets see which columns are non numeric info describe method

In [70]:

```
#Check for the non-numeric column
```

In [71]:

```
titanic_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 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          891 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  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(4)
memory usage: 76.7+ KB
```

In [72]:

```
titanic_data.dtypes
```

Out[72]:

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Embarked        object
dtype: object
```

We can see, Name, Sex, Ticket and Embarked are non-numerical.It seems Name,Embarked and Ticket number are not useful for Machine Learning Prediction hence we will eventually drop it. For Now we would convert Sex Column to dummies numerical values**

In [73]:

```
#convert sex column to numerical values
```

In [74]:

```
gender = pd.get_dummies(titanic_data['Sex'], drop_first = True)
```

In [75]:

```
titanic_data ['Gender'] = gender
```

In [78]:

```
titanic_data.head()
```

Out[78]:

	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

In [79]:

```
#drop the columns which are not required
```

In [80]:

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

In [81]:

```
titanic_data.head()
```

Out[81]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Gender
0	1	0	3	22.0	1	0	7.2500	1
1	2	1	1	38.0	1	0	71.2833	0
2	3	1	3	26.0	0	0	7.9250	0
3	4	1	1	35.0	1	0	53.1000	0
4	5	0	3	35.0	0	0	8.0500	1

In [82]:

```
#Seperate Dependent and Independent variables
```

In [84]:

```
x = titanic_data[['PassengerId','Pclass','Age','SibSp','Parch','Fare','Gender']]
y = titanic_data['Survived']
```

In [85]:

```
y
```

Out[85]:

```
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
```

Data Modelling

Building Model using Logestic Regression

Build the model

In [86]:

```
#import train test split method
```

In [87]:

```
from sklearn.model_selection import train_test_split
```

In [88]:

```
#train test split
```

In [89]:

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.33,random_state=42)
```

In [90]:

```
#import Logistic  Regression
```

In [91]:

```
from sklearn.linear_model import LogisticRegression
```

In [92]:

```
#Fit  Logistic Regression
```

In [93]:

```
lr = LogisticRegression()
```

In [95]:

```
lr.fit(x_train,y_train)
```

```
C:\Users\baps\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

Out[95]:

```
▸ LogisticRegression
```

In [96]:

```
#predict
```

In [97]:

```
predict = lr.predict(x_test)
```

Testing

See how our model is performing

In [98]:

```
#print confusion matrix
```

In [99]:

```
from sklearn.metrics import confusion_matrix
```

In [102]:

```
pd.DataFrame(confusion_matrix(y_test, predict), columns = ['Predicted No', 'Predicted Yes'
```

Out[102]:

	Predicted No	Predicted Yes
Actual No	151	24
Actual Yes	38	82

Type Markdown and LaTeX: $\alpha 2$

In [104]:

```
#import classification report
```

In [105]:

```
from sklearn.metrics import classification_report
```

In [106]:

```
print(classification_report(y_test,predict))
```

```
              precision    recall  f1-score   support

           0       0.80      0.86      0.83       175
           1       0.77      0.68      0.73       120

    accuracy                           0.79       295
   macro avg       0.79      0.77      0.78       295
weighted avg       0.79      0.79      0.79       295
```

Precision is fine considering Model Selected and Available Data. Accuracy can be increased by further using more features (which we dropped earlier) and/or by using other model

Note:
Precision : Precision is the ratio of correctly predicted positive observations to the total predicted positive observations
Recall : Recall is the ratio of correctly predicted positive observations to the all observations in actual class
F1 score - F1 Score is the weighted average of Precision and Recall.

In [ ]:

```

```