Titanic Disaster Survival Using Logistic Regression

In [1]:
```
#import Libraries
```

In [4]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Load the Data

In [8]:
```
titanic_data = pd.read_csv("titanic_train.csv")
```

In [10]:
```
titanic_data
```

Out[10]:

	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
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

In [11]:
```
len(titanic_data)
```

Out[11]:
```
891
```

View the data using head function which return top rows

In [12]:
```
titanic_data.head()
```

Out[12]:

	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
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

In [13]:
```
titanic_data.index
```

Out[13]:
```
RangeIndex(start=0, stop=891, step=1)
```

In [14]:
```
titanic_data.columns
```

Out[14]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [16]:
```
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 [18]:
```
titanic_data.dtypes
```

Out[18]:
```
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 [21]:
```
titanic_data.describe()
```

Out[21]:

	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 Datasets

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

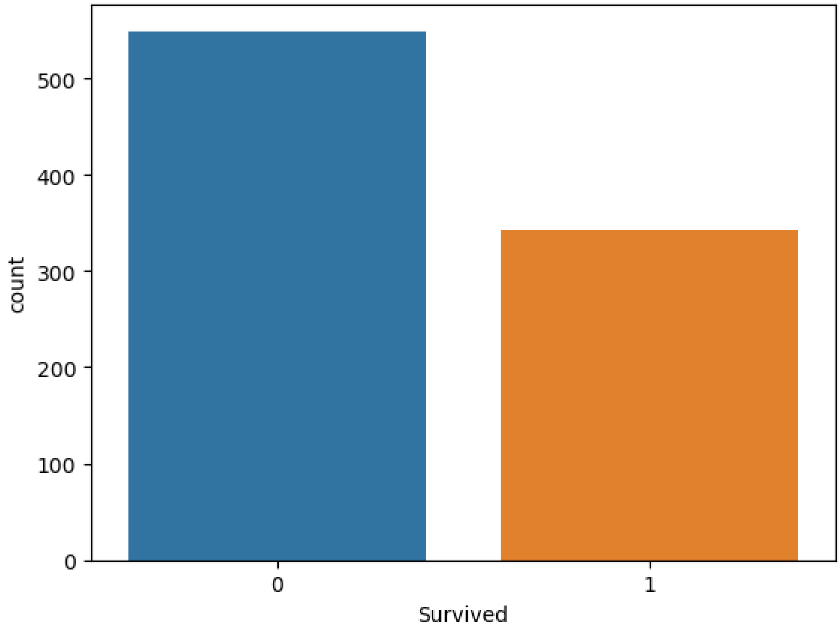
import Seaborn for visually analysing the data

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

In [22]:
```
#countplot of survived vs not survived
```

In [23]:
```
sns.countplot(x='Survived', data=titanic_data)
```

Out[23]: <Axes: xlabel='Survived', ylabel='count'>

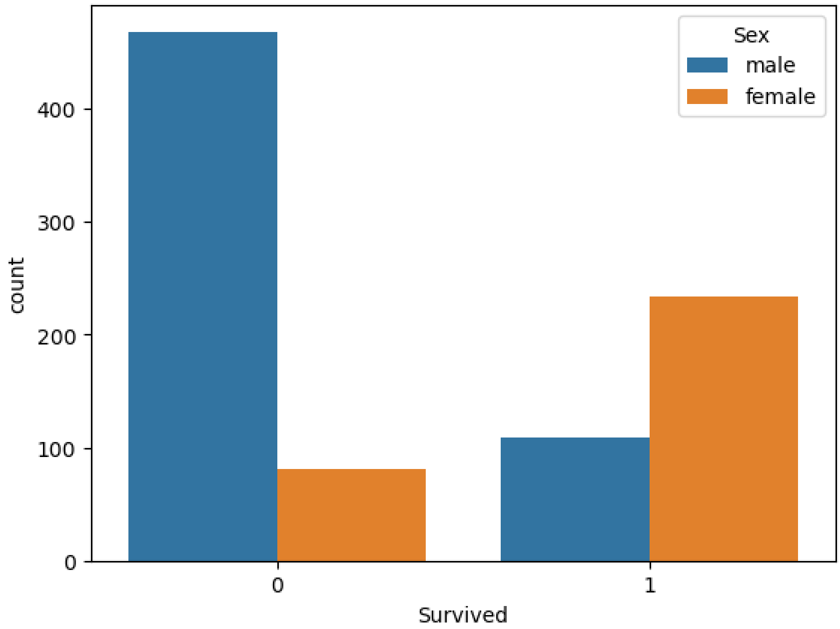


Male vs Female Survival

In [24]:
```
#Male vs Female Survival
```

In [25]:
```
sns.countplot(x='Survived', data=titanic_data, hue='Sex')
```

Out[25]: <Axes: xlabel='Survived', ylabel='count'>



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 [26]:
```
#Check for null
```

In [28]:
```
titanic_data.isna()
```

Out[28]:

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

891 rows × 12 columns

In [29]:
```
#Check how many values are null
```

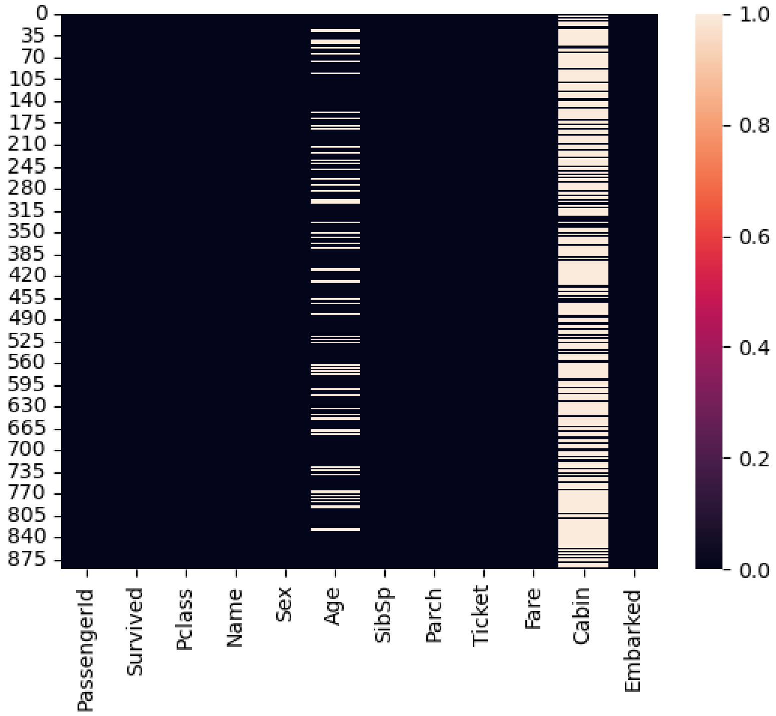
In [30]:
```
titanic_data.isna().sum()
```

Out[30]:
```
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 [31]:
```
#Visualize null values
```

In [32]:
```
sns.heatmap(titanic_data.isna())
```

Out[32]:
```
<Axes: >
```



In [33]:
```
#find the % of null values in age column
```

In [39]:
```
(titanic_data['Age'].isna().sum()/len(titanic_data['Age']))*100
```

Out[39]:
```
19.865319865319865
```

In [37]:
```
#find the % of null values in cabin column
```

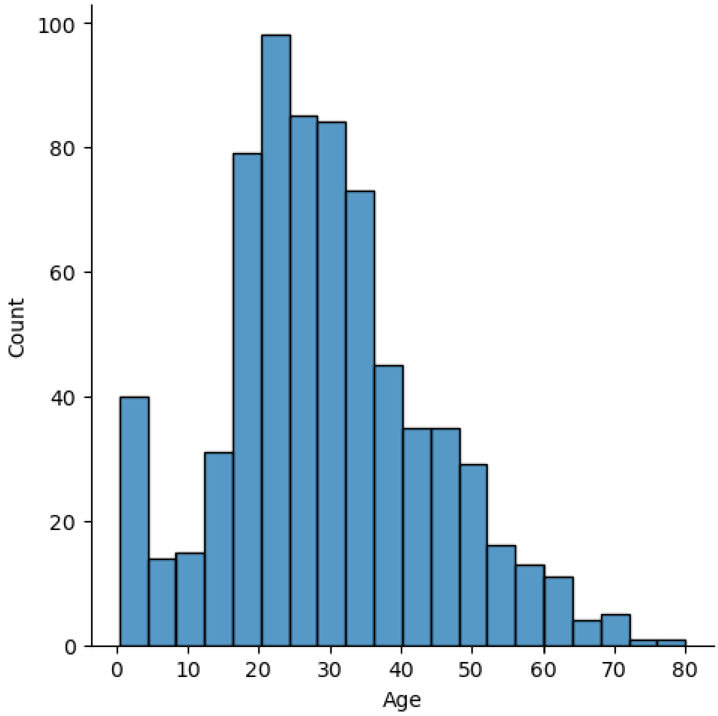
In [40]:
```
(titanic_data['Cabin'].isna().sum()/len(titanic_data['Cabin']))*100
```

Out[40]:
```
77.10437710437711
```

In [41]:
```
#find the distribution for the age column
```

In [42]:
```
sns.displot(x='Age', data=titanic_data)
```

Out[42]:
```
<seaborn.axisgrid.FacetGrid at 0x1bb41bcbbe0>
```



Data Cleaning

Fill the missing values</BR> we will find the missing values for age. In order to fill missing values we use fillna method.</BR> For now we will fill the missing age by taking average of all age.

In [43]:
```
#fill age column
```

In [52]:
```
titanic_data['Age'].fillna(titanic_data['Age'].mean(), inplace=True)
```

In [ ]:
```
#verify null values
```

In [51]:
```
titanic_data['Age'].isna().sum()
```

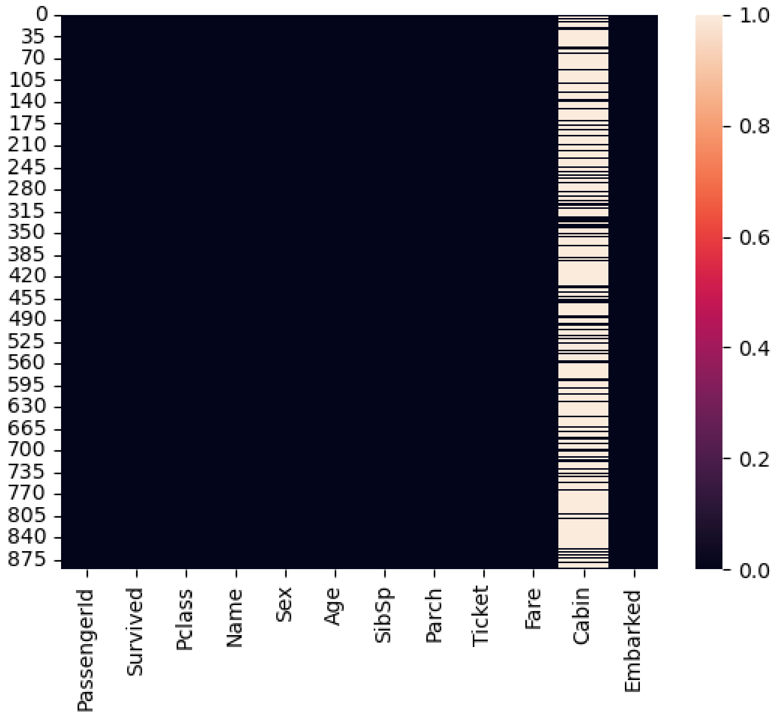
Out[51]:
```
0
```

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

In [53]:
```
#visualize null values
```

In [54]:
```
sns.heatmap(titanic_data.isna())
```

Out[54]:
```
<Axes: >
```



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

In [55]:
```
#drop cabin column
```

In [56]:
```
titanic_data.drop('Cabin', axis=1, inplace=True)
```

In [57]:
```
#see the content of data
```

In [58]:
```
titanic_data.head()
```

Out[58]:

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

Preaparing Data for Model </BR> 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 [59]:
```
#Check for the non-numeric column
```

In [60]:
```
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 [61]:
```
titanic_data.dtypes
```

Out[61]:
```
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 [62]:
```
#convert sex column to numerical values
```

In [65]:
```
gender=pd.get_dummies(titanic_data['Sex'],drop_first=True)
```

In [66]:
```
titanic_data['Gender']=gender
```

In [69]:
```
titanic_data.head()
#Here, in Gender male=1 & female=0
```

Out[69]:

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

In [70]:
```
#drop the column which are not require
```

In [71]:
```
titanic_data.drop(['Name','Sex','Ticket','Embarked'], axis=1, inplace=True)
```

In [72]:
```
titanic_data.head()
```

Out[72]:

	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 [73]:
```
#Seperate Dependent and Independent variables
```

In [75]:
```
x=titanic_data[['PassengerId','Pclass','Age','SibSp','Parch','Fare','Gender']]
y=titanic_data['Survived']
```

In [76]:
```
x
```

Out[76]:

	PassengerId	Pclass	Age	SibSp	Parch	Fare	Gender
0	1	3	22.000000	1	0	7.2500	1
1	2	1	38.000000	1	0	71.2833	0
2	3	3	26.000000	0	0	7.9250	0
3	4	1	35.000000	1	0	53.1000	0
4	5	3	35.000000	0	0	8.0500	1
...	...	...	...	...	...	...	...
886	887	2	27.000000	0	0	13.0000	1
887	888	1	19.000000	0	0	30.0000	0
888	889	3	29.699118	1	2	23.4500	0
889	890	1	26.000000	0	0	30.0000	1
890	891	3	32.000000	0	0	7.7500	1

891 rows × 7 columns

In [77]:
```
y
```

Out[77]:
```
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 Modeling

Building Model using Logistic Regression</BR> </BR> **Build the model**

In [79]:
```
#import train test split method
```

In [80]:
```
from sklearn.model_selection import train_test_split
```

In [81]:
```
#train test split
```

In [83]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42)
```

In [84]:
```
#import logistic Regression
```

In [85]:
```
from sklearn.linear_model import LogisticRegression
```

In [86]:
```
#fit Logistic Regression
```

In [92]:
```
lr = LogisticRegression()
```

In [93]:
```
lr.fit(X_train,y_train)
```

```
C:\Users\RISHABH\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py: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 in:
    https://scikit-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-regression
  n_iter_i = _check_optimize_result(
```

Out[93]:
```
▾ LogisticRegression
LogisticRegression()
```

In [96]:
```
#predict
```

In [99]:
```
predict=lr.predict(X_test)
```

Testing

See how our model is performing

In [100…

```
#print confusion matrix
```

In [101…

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

In [104…

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

Out[104]:

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

In [105…

```
#import classification report
```

In [106…

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

In [108…

```
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:</BR> 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.

You can find this project on **GitHub**