d3wjxxhy5

August 28, 2024

Machine Learning Project

Titanic Survival Prediction Using Machine Learning

WorkFlow–>

1.Collecting Data

2.Preprocessing The Data

3.Analysing The Data

4.Splitting the data into Test and Train

5.Applying a Machine Learning Model(Logistic Regression)

6.Evaluation/Cross-Checking our Model using Test Data

Step1 : Import The Libraries/Dependencies

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

Step2 : Data Collection and Processing

[ ]:
```
#Load the data from a csv file to a Pandas Dataframe

titanic_data=pd.read_csv('/content/train.csv')
```

[ ]:
```
#Print first 5 rows of Dataframe

titanic_data.head()
```

[ ]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
```

```
2            3          1       3
3            4          1       1
4            5          0       3

                                                   Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

[ ]:
```
#Check The number of Rows and Columns

titanic_data.shape
```

[ ]:
```
(891, 12)
```

[ ]:
```
# Getting some more info about the data(titanic_data)

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
```

[ ]:
```python
#To Check number of Missing values in each Column

titanic_data.isnull().sum()
```

[ ]:
```
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
```

Step3 : Handling Missing Values

[ ]:
```python
#Drop Cabin Column as it lacks much data
#axis=0 represent row and 1 represent column

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

[ ]:
```python
#Replace Missing values in 'Age' Column with Mean Value
#Inplace=True will permantly replace the dataframe with mean values

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

[ ]:
```python
#Finding The mode Value of 'Embarked' Column

print(titanic_data['Embarked'].mode())
```

```
0    S
Name: Embarked, dtype: object
```

[ ]:
```python
print(titanic_data['Embarked'].mode()[0])
```

```
S
```

[ ]:
```python
#Replace Missing values in 'Embarked' Column with Mode

titanic_data['Embarked'].fillna(titanic_data['Embarked'].mode()[0],␣
 ↪inplace=True)
```

[ ]:
```
#Check Missing Values Now

titanic_data.isnull().sum()
```

[ ]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

Step4 : Data Analysis

[ ]:
```
#Get Some Statistical Measures about the data

titanic_data.describe()
```

[ ]:
```
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  891.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   13.002015    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   22.000000    0.000000
50%     446.000000    0.000000    3.000000   29.699118    0.000000
75%     668.500000    1.000000    3.000000   35.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

[ ]:
```
#Finding the number of People Survived and Not Survived

titanic_data['Survived'].value_counts()
```

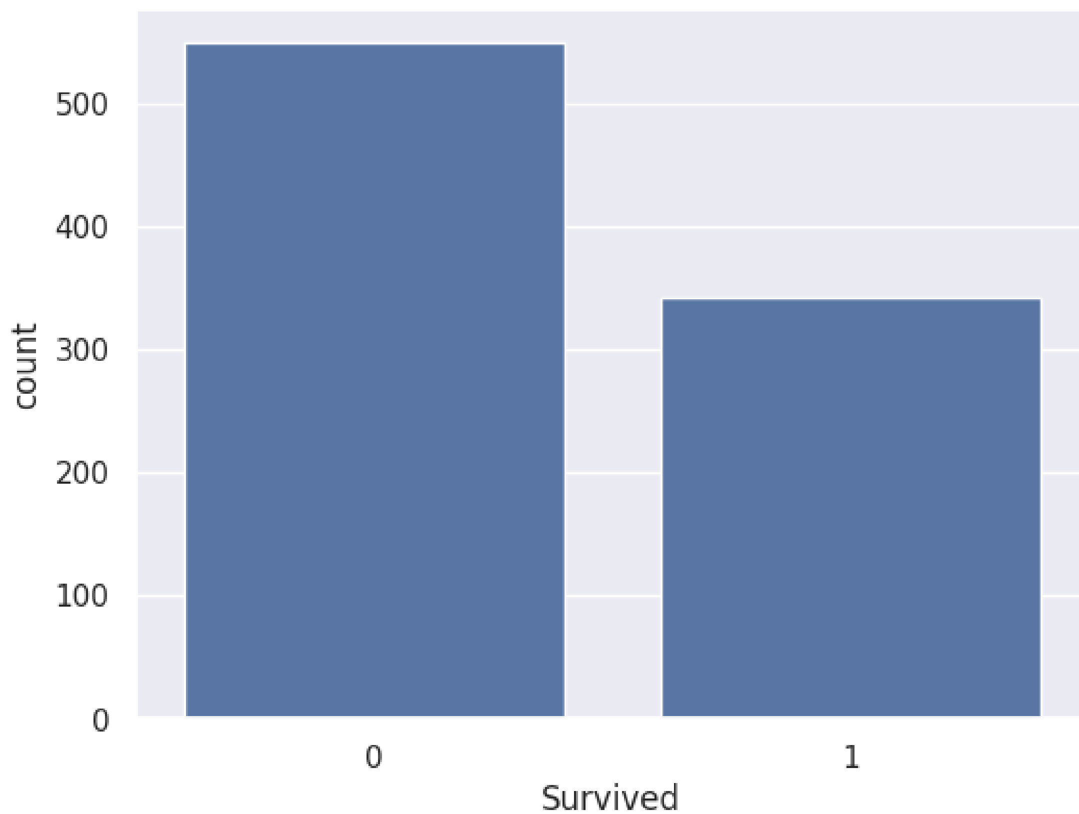
[ ]: Survived
0   549
1   342
Name: count, dtype: int64

Step5 : Data Visualization

[ ]:
```
sns.set()
```

[ ]:
```
#Making a Count-Plot for Survived Column

sns.countplot(titanic_data , x="Survived")
```

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

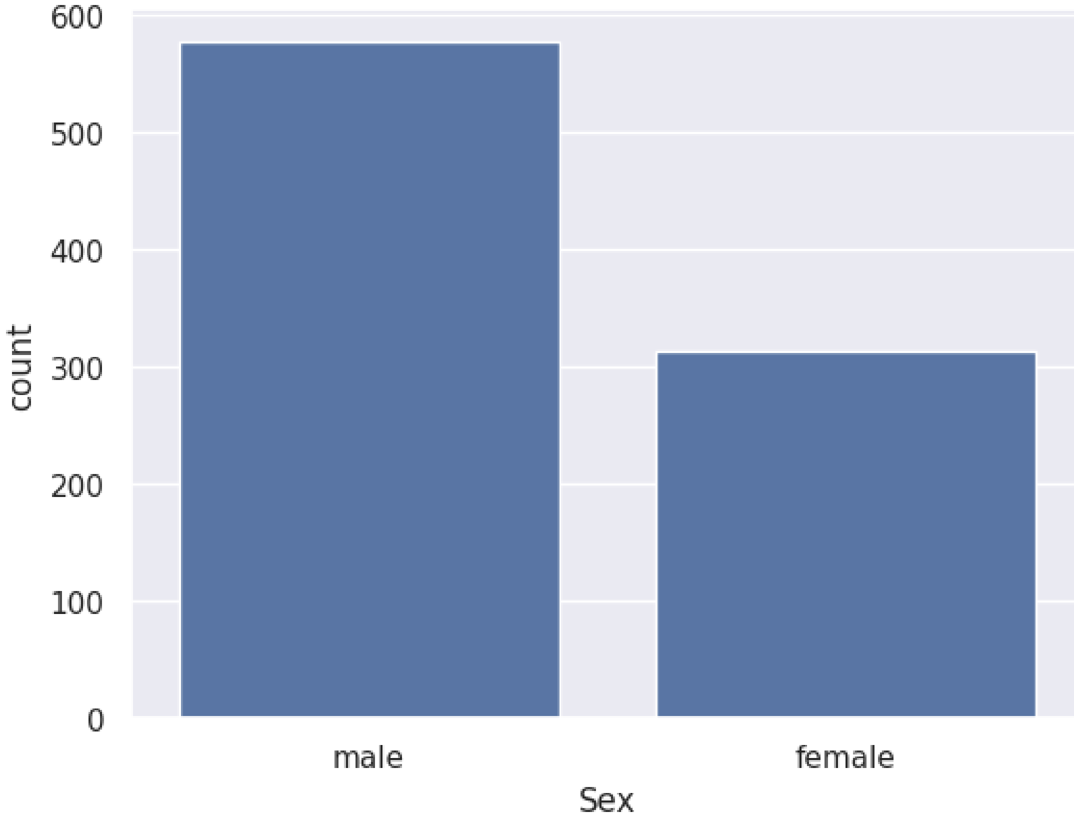


[ ]:
```
titanic_data['Sex'].value_counts()
```

[ ]: Sex
male   577
female   314
Name: count, dtype: int64

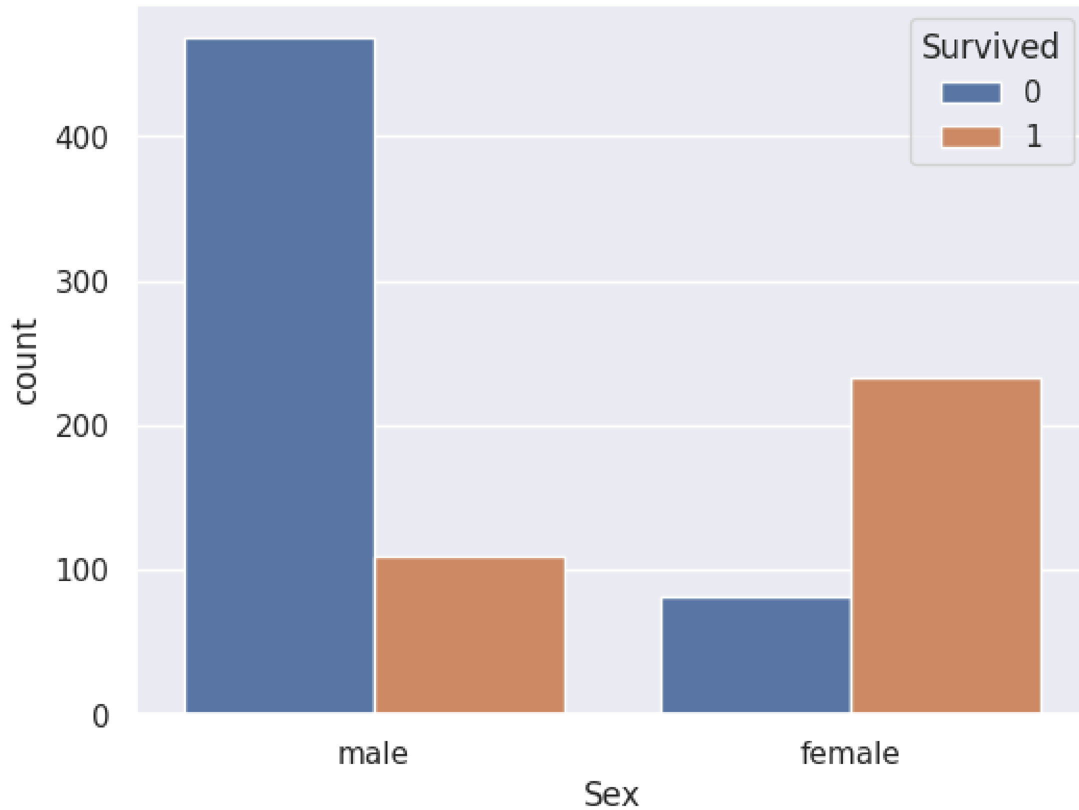
[ ]:
```python
#Making a Count-Plot for "Gender" Column

sns.countplot(titanic_data , x="Sex")
```

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



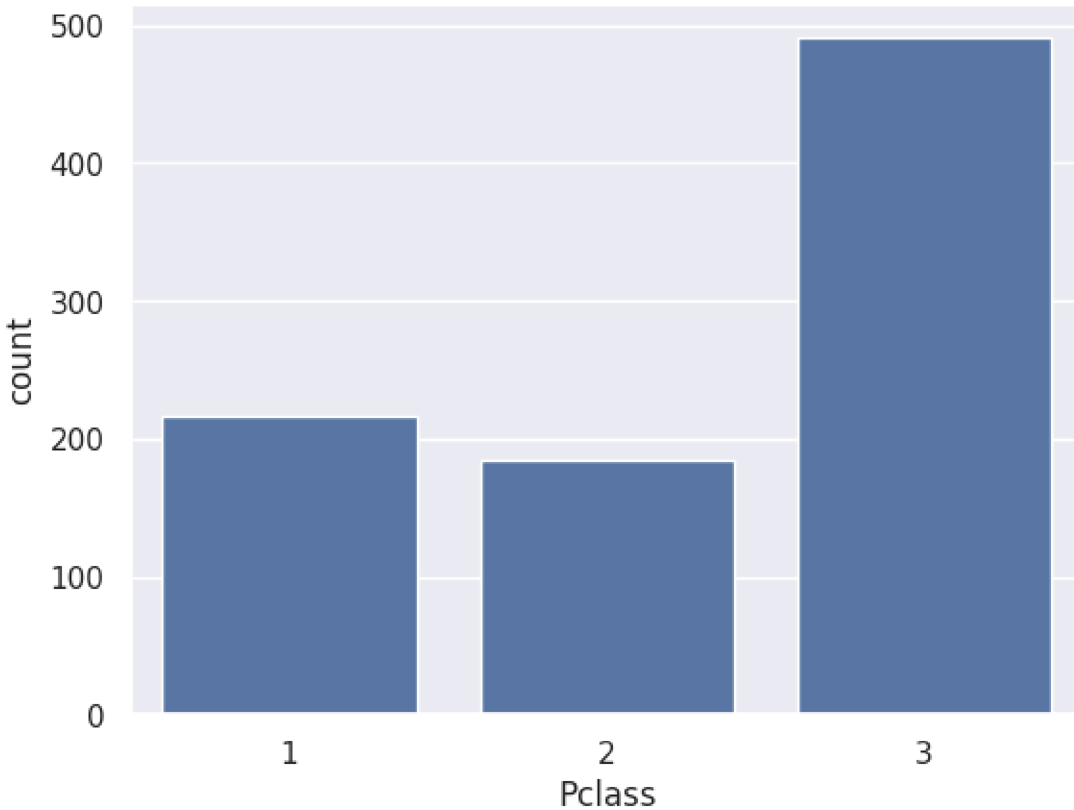
[ ]:
```python
# Number Of Survivors based on Gender

sns.countplot(titanic_data, x="Sex", hue="Survived")
```

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



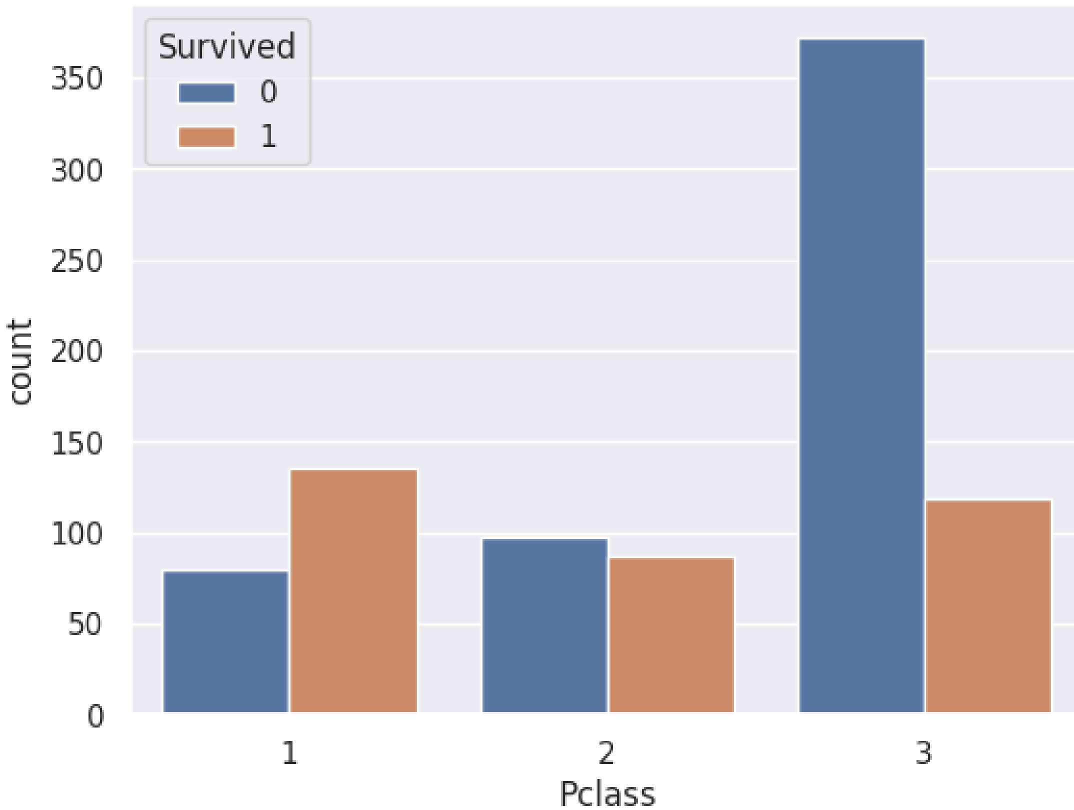
[ ]:
```
#Making a Count-Plot for P-Class
sns.countplot(titanic_data,x='Pclass')
```

[ ]: <Axes: xlabel='Pclass', ylabel='count'>



[ ]:
```
#Classify dataset based on Pclass
sns.countplot(titanic_data, x="Pclass", hue="Survived")
```

[ ]: <Axes: xlabel='Pclass', ylabel='count'>



Step 6: Encoding the Categorical Columns

[ ]:
```
titanic_data['Sex'].value_counts()
```

[ ]:
```
Sex
male      577
female    314
Name: count, dtype: int64
```

[ ]:
```
titanic_data['Embarked'].value_counts()
```

[ ]:
```
Embarked
S    646
C    168
Q     77
Name: count, dtype: int64
```

[ ]:
```
#Replace the above categories with numbers as 0,1,2

titanic_data.replace({'Sex':{'male':0,'female':1}, 'Embarked':{'S':0, 'C':1,␣
↪'Q':2}}, inplace=True)
```

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

[ ]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name  Sex   Age  SibSp  Parch  \
0                            Braund, Mr. Owen Harris    0  22.0      1      0
1  Cumings, Mrs. John Bradley (Florence Briggs Th…    1  38.0      1      0
2                             Heikkinen, Miss. Laina    1  26.0      0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)    1  35.0      1      0
4                           Allen, Mr. William Henry    0  35.0      0      0

             Ticket     Fare  Embarked
0         A/5 21171   7.2500         0
1          PC 17599  71.2833         1
2  STON/O2. 3101282   7.9250         0
3            113803  53.1000         0
4            373450   8.0500         0
```

Step 7: Separating Features and Target

[ ]:
```
#x is data for prediction of y and y is target that we need to predict
```

[ ]:
```
x= titanic_data.drop(columns=['PassengerId','Name','Ticket','Survived'],axis=1)
y= titanic_data['Survived']
```

[ ]:
```
print(x)
```

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

[891 rows x 7 columns]
```

```
[ ]: print(y)
```

```
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
```

Step 8: Splitting Data as Training and Test Data

```
[ ]: X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.2,␣
      ↪random_state=2)
```

```
[ ]: print(x.shape, X_train.shape, X_test.shape)
```

```
(891, 7) (712, 7) (179, 7)
```

Step 9: Training Model using Logistic Regression

```
[ ]: #we create a model instance to train our features

     model=LogisticRegression()
```

```
[ ]: #training The Logistic Regression model with the training data

     model.fit(X_train, Y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:460:
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(
```

```
[ ]: LogisticRegression()
```

Step 10: Model Evaluation using Accuracy_Score

[ ]:
```
#accuracy_score on training data

X_train_prediction=model.predict(X_train)
```

[ ]:
```
print(X_train_prediction)
```

```
[0 1 0 0 0 0 0 1 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 1 0 0 1 0 1
 0 0 0 0 0 0 1 1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 1 0 0 1 1 0 0 1 1 0 1 0 0 1
 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 1 0 0 0 1 1 1 0 1 0 0 0 0 0 1 0 0 0
 1 1 0 0 1 0 0 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 0 1 0 0
 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 0 1 1 1
 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 1 1 1 1 0 0 0 0 0 0 0
 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0
 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 1 0 0 1 0 0 0 1 0 0 0
 0 1 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 0 1 1 0 1 1
 0 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 1 0 0 0 0 1 1 0 0 0 1 0 1 1 1 0 0
 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 1 1 1 0 1 1 0 0 0
 0 1 0 1 0 0 1 1 0 0 0 0 1 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0
 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 1 1 0 1 0 0 1 0 0 0 1 1 0 1 0
 0 0 0 0 1 0 0 1 0 1 1 0 0 1 0 0 1 0 0 0 1 0 1 1 0 0 1 1 0 1 0 1 1 1 0 1 0
 0 1 0 0 1 0 0 1 0 0 0 0 1 1 0 0 1 0 1 0 0 0 0 0 0 1 1 1 0 0 1 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
 0 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 1 0 1 1 1 0 0 0 1 0 1 0 0 0 1 1 1 0 0 1 1
 0 0 0 1 0 1 0 0 0 0 0 1 1 0 1 1 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0
 1 0 0 1 0 1 0 0 0 1 1 1 1 1 0 0 1 1 0 1 1 1 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0
 0 0 0 1 1 0 0 1 0]
```

[ ]:
```
#Here the target is being compared with the predicted trained outcome
#It prints the probability of correct predictions
# of x_pred with y_train
training_data_accuracy=accuracy_score(Y_train, X_train_prediction)
```

[ ]:
```
print("Accuracy of Training Data:", training_data_accuracy)
```

```
Accuracy of Training Data: 0.8075842696629213
```

[ ]:
```
#accuracy_score on test data

X_test_prediction=model.predict(X_test)
```

[ ]:
```
print(X_test_prediction)
```

```
[0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 1 1 0 1 0 1 1 0 0 0 0 0 0 0 0 1 1
 0 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0
 1 0 0 0 1 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 0 1 1 0 1 1 0 0 0 0
 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1 1 0 1 0 0
 0 1 0 0 0 0 1 0 0 1 1 0 1 0 0 0 1 1 0 0 1 0 0 1 1 1 0 0 0 0 0]
```

[ ]:
```python
#Now predict on test data
test_data_accuracy=accuracy_score(Y_test, X_test_prediction)
```

[ ]:
```python
print("Accuracy of Test Data:", training_data_accuracy)
```

```
Accuracy of Test Data: 0.8075842696629213
```

The Above Model can predict correctly in almost 80 cases out of 100 cases. That's pretty good..

[ ]:
```python

```