Titanic Survival Prediction

```
# importing essential libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing and Understanding Dataset

```
# loading dataset
titanic=pd.read_csv('/content/drive/MyDrive/Datasets/CodSoft/tested.xls')
titanic
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticke
0	892	0	3	Kelly, Mr. James	male	34.5	0	0	33091
1	893	1	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	36327
2	894	0	2	Myles, Mr. Thomas Francis	male	62.0	0	0	24027
3	895	0	3	Wirz, Mr. Albert	male	27.0	0	0	31515
4	896	1	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	310129

```
titanic.shape
```

```
(418, 12)
```

```
titanic.size
```

```
5016
```

```
titanic.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket

0	892	0	3	Kelly, Mr. James	male	34.5	0	0	330911	7.
1	893	1	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.
				Myles						

```
titanic.tail()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticke
413	1305	0	3	Spector, Mr. Woolf	male	NaN	0	0	A.5. 323
414	1306	1	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 1775
				Saether					

```
titanic.columns
```

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

```
titanic.dtypes
```

```
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
```

```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Survived     418 non-null    int64  
 2   Pclass       418 non-null    int64  
 3   Name         418 non-null    object 
 4   Sex          418 non-null    object 
 4   Sex          418 non-null    object 
 5   Age          332 non-null    float64
 6   SibSp        418 non-null    int64  
 7   Parch        418 non-null    int64  
 8   Ticket       418 non-null    object 
 9   Fare         417 non-null    float64
 10  Cabin        91 non-null     object 
 11  Embarked     418 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 39.3+ KB
```

```
titanic.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	
count	418.000000	418.000000	418.000000	332.000000	418.000000	418.000000	417.0
mean	1100.500000	0.363636	2.265550	30.272590	0.447368	0.392344	35.6
std	120.810458	0.481622	0.841838	14.181209	0.896760	0.981429	55.9
min	892.000000	0.000000	1.000000	0.170000	0.000000	0.000000	0.0
25%	996.250000	0.000000	1.000000	21.000000	0.000000	0.000000	7.8
50%	1100.500000	0.000000	3.000000	27.000000	0.000000	0.000000	14.4
75%	1204.750000	1.000000	3.000000	39.000000	1.000000	0.000000	31.5
max	1309.000000	1.000000	3.000000	76.000000	8.000000	9.000000	512.3

```
titanic.isna().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

```
titanic=titanic.drop(['Cabin'],axis=1)
```

```
titanic['Age']=titanic['Age'].fillna(titanic['Age'].mean())
titanic['Fare']=titanic['Fare'].fillna(titanic['Fare'].mean())
titanic.isna().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
```

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

```
0    266
1    152
Name: Survived, dtype: int64
```

```
titanic['Pclass'].value_counts()
```

```
3    218
1    107
2     93
Name: Pclass, dtype: int64
```

```
titanic['Sex'].value_counts()
```

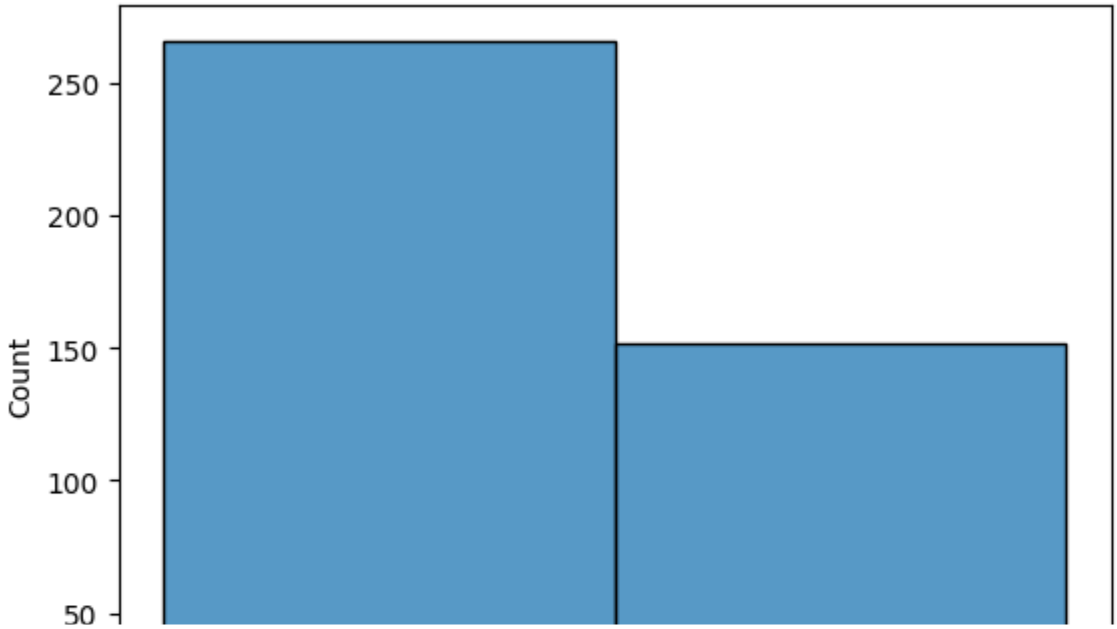
```
male      266
female    152
Name: Sex, dtype: int64
```

```
titanic['Embarked'].value_counts()
```

```
S    270
C    102
Q     46
Name: Embarked, dtype: int64
```

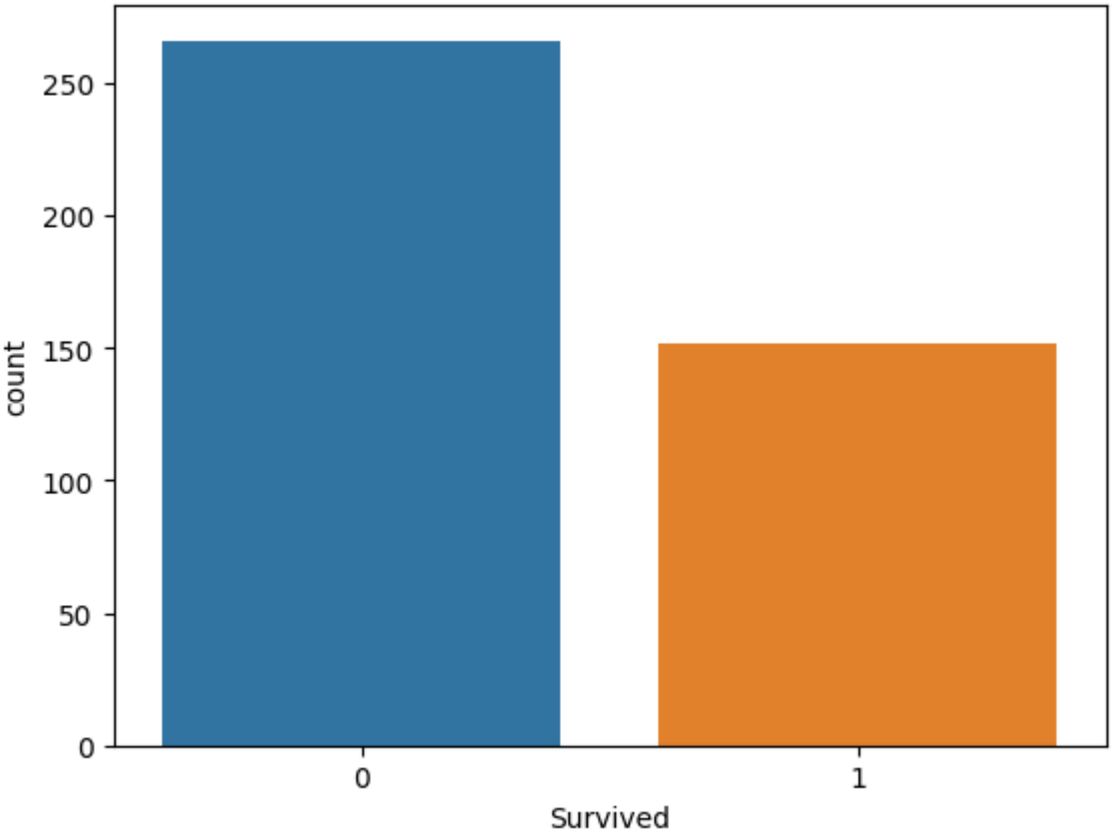
```
# histplot
sns.histplot(titanic,x='Sex')
```

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



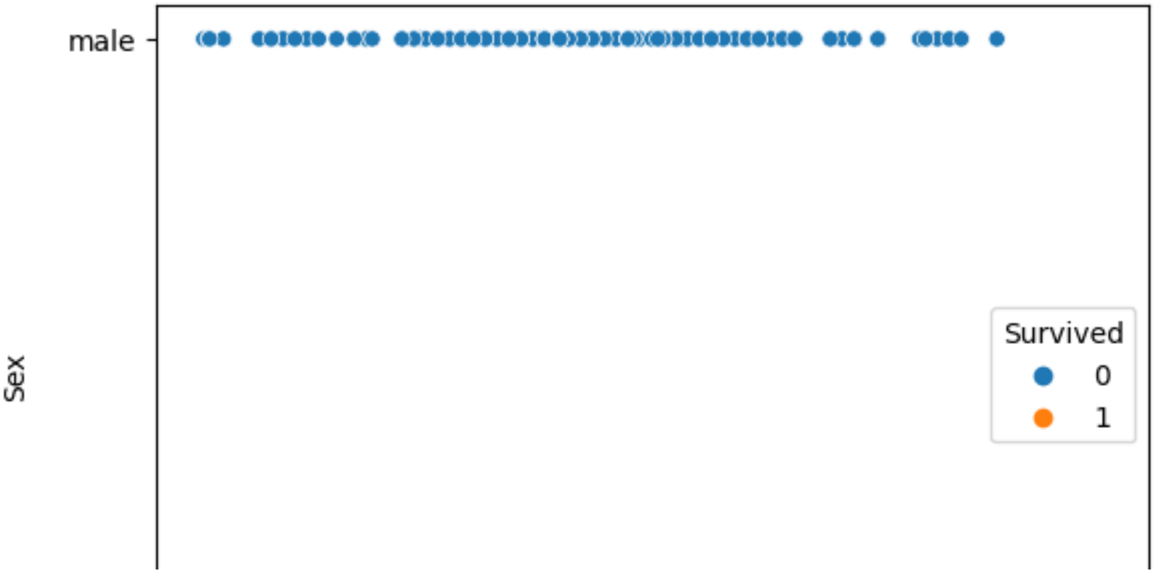
```
#countplot
sns.countplot(data=titanic,x='Survived')
```

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



```
# scatterplot
sns.scatterplot(data=titanic,x='Age',y='Sex',hue='Survived')
```

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



```
# boxplot
sns.boxplot(titanic,x='Survived',y='Age')
```

```
# pairplot
sns.pairplot(data=titanic,hue='Survived')
```

```
titanic.corr()
```

```
sns.heatmap(titanic.corr(),annot=True,fmt='.3f')
```

```
# Label encoding
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
titanic['Sex']=le.fit_transform(titanic['Sex'])
titanic['Embarked']=le.fit_transform(titanic['Embarked'])
titanic
```

```
titanic_surv=titanic.drop(columns=['PassengerId','Name','Ticket'],axis=1)
titanic_surv
```

```
# separating i/p and output
x=titanic_surv.iloc[:,:-1].values
y=titanic_surv.iloc[:,-1].values
print("Dimensions \nX :",x.ndim,"\nY :",y.ndim)
```

```
Dimensions
X : 2
Y : 1
```

```
x
```

```
array([[ 0.    ,  3.    ,  1.    , ...,  0.    ,  0.    ,  7.8292],
       [ 1.    ,  3.    ,  0.    , ...,  1.    ,  0.    ,  7.    ],
       [ 0.    ,  2.    ,  1.    , ...,  0.    ,  0.    ,  9.6875],
       ...,
       [ 0.    ,  3.    ,  1.    , ...,  0.    ,  0.    ,  7.25  ],
       [ 0.    ,  3.    ,  1.    , ...,  0.    ,  0.    ,  8.05  ],
       [ 0.    ,  3.    ,  1.    , ...,  1.    ,  1.    , 22.3583]])
```

```
y
```

```
array([1, 2, 1, 2, 2, 2, 1, 2, 0, 2, 2, 2, 2, 2, 2, 0, 1, 0, 2, 0, 0, 2,
       2, 0, 0, 2, 0, 0, 2, 0, 2, 2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 2, 2,
       2, 2, 0, 1, 0, 2, 2, 0, 2, 2, 0, 1, 2, 2, 2, 0, 2, 2, 2, 1, 0, 2,
       1, 2, 0, 2, 1, 2, 2, 0, 0, 0, 2, 2, 2, 1, 0, 2, 2, 2, 1, 0, 1, 2,
       1, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 2, 1, 2, 0, 2, 1, 1, 2, 2,
       0, 1, 0, 1, 2, 0, 0, 2, 0, 2, 2, 1, 0, 2, 1, 2, 2, 1, 2, 2, 2, 0,
       2, 0, 2, 2, 0, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 0, 0, 2, 2,
       2, 2, 2, 2, 2, 2, 1, 0, 2, 2, 2, 2, 0, 2, 0, 2, 2, 0, 2, 0, 2, 2,
       2, 0, 2, 0, 2, 0, 2, 1, 0, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 0, 2,
       2, 2, 1, 2, 0, 2, 2, 0, 1, 2, 0, 2, 2, 2, 2, 2, 2, 2, 1, 2, 0, 2,
       0, 2, 2, 2, 0, 0, 2, 1, 2, 2, 2, 2, 2, 1, 0, 2, 0, 0, 2, 0, 0, 2,
       0, 2, 2, 2, 2, 2, 2, 0, 2, 2, 0, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2,
       0, 2, 2, 2, 2, 2, 0, 1, 0, 1, 0, 2, 2, 2, 2, 2, 2, 2, 1, 0, 2, 2,
       2, 2, 0, 2, 2, 1, 0, 2, 2, 2, 0, 0, 2, 2, 2, 0, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 0, 2, 1, 0, 1, 0, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 2, 2,
       2, 0, 0, 0, 2, 2, 2, 0, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0, 2,
       2, 2, 2, 2, 2, 2, 1, 2, 2, 0, 2, 2, 0, 2, 0, 2, 0, 0, 2, 0, 2, 2,
       2, 0, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2,
       1, 0, 2, 1, 2, 2, 0, 2, 0, 0, 2, 0, 1, 2, 1, 1, 2, 2, 0, 2, 2, 0])
```

```
# splitting dataset into training and testing data
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=42)
```

```
x_train
```

```
array([[  0.    ,   1.    ,   1.    , ...,   0.    ,   0.    ,  75.2417],
       [  0.    ,   3.    ,   1.    , ...,   0.    ,   0.    ,   7.75  ],
       [  1.    ,   1.    ,   0.    , ...,   1.    ,   0.    , 221.7792],
       ...,
       [  0.    ,   1.    ,   1.    , ...,   0.    ,   0.    ,  75.2417],
       [  0.    ,   2.    ,   1.    , ...,   0.    ,   0.    ,  13.5   ],
       [  0.    ,   3.    ,   1.    , ...,   0.    ,   0.    ,   7.75  ]])
```

```
y_train
```

```
array([0, 1, 2, 0, 0, 2, 2, 0, 1, 0, 2, 1, 2, 2, 0, 0, 2, 2, 0, 2, 0, 0,
       2, 2, 0, 1, 0, 2, 0, 2, 2, 2, 2, 2, 2, 0, 0, 1, 2, 2, 1, 2, 2, 2,
       0, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 1,
       2, 2, 2, 0, 2, 2, 0, 0, 2, 2, 0, 2, 0, 2, 2, 2, 2, 0, 2, 0, 0, 2,
       2, 2, 1, 2, 2, 2, 2, 0, 2, 1, 2, 2, 0, 0, 2, 2, 2, 2, 2, 0, 0, 0,
       0, 2, 0, 0, 2, 2, 1, 0, 2, 1, 2, 0, 2, 2, 0, 2, 2, 2, 1, 1, 2, 0,
       2, 0, 2, 2, 1, 2, 1, 2, 0, 2, 2, 2, 2, 0, 0, 2, 2, 2, 0, 2, 2, 0,
       2, 2, 2, 2, 0, 0, 0, 2, 2, 2, 2, 1, 1, 2, 0, 2, 1, 2, 0, 2, 2, 2,
       0, 2, 0, 2, 2, 1, 2, 2, 1, 0, 0, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 0,
       0, 2, 2, 2, 0, 2, 2, 1, 2, 0, 0, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 0,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 2, 2, 0, 2, 2, 0, 2, 2, 1, 1,
       1, 0, 2, 1, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 0,
       2, 1, 1, 2, 2, 2, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 1, 0, 0,
       2, 2, 1, 0, 2, 1])
```

```
x_test
```

```
array([[0.00000000e+00, 3.00000000e+00, 1.00000000e+00, 2.50000000e+01,
        0.00000000e+00, 0.00000000e+00, 7.22920000e+00],
       [1.00000000e+00, 1.00000000e+00, 0.00000000e+00, 3.90000000e+01,
        0.00000000e+00, 0.00000000e+00, 2.11337500e+02],
       [0.00000000e+00, 3.00000000e+00, 1.00000000e+00, 2.10000000e+01,
        0.00000000e+00, 0.00000000e+00, 7.75000000e+00],
       [0.00000000e+00, 3.00000000e+00, 1.00000000e+00, 3.50000000e+01,
        0.00000000e+00, 0.00000000e+00, 7.89580000e+00],
       [1.00000000e+00, 3.00000000e+00, 0.00000000e+00, 3.60000000e+01,
        0.00000000e+00, 2.00000000e+00, 1.21833000e+01],
       [0.00000000e+00, 2.00000000e+00, 1.00000000e+00, 5.00000000e+01,
        1.00000000e+00, 0.00000000e+00, 2.60000000e+01],
       [1.00000000e+00, 3.00000000e+00, 0.00000000e+00, 2.90000000e+01,
        0.00000000e+00, 0.00000000e+00, 7.92500000e+00],
       [0.00000000e+00, 1.00000000e+00, 1.00000000e+00, 4.90000000e+01,
        0.00000000e+00, 0.00000000e+00, 2.60000000e+01],
       [1.00000000e+00, 2.00000000e+00, 0.00000000e+00, 1.90000000e+01,
        0.00000000e+00, 0.00000000e+00, 1.30000000e+01],
       [0.00000000e+00, 3.00000000e+00, 1.00000000e+00, 3.02725904e+01,
        0.00000000e+00, 0.00000000e+00, 8.05000000e+00],
       [0.00000000e+00, 3.00000000e+00, 1.00000000e+00, 2.10000000e+01,
        2.00000000e+00, 0.00000000e+00, 2.41500000e+01],
       [1.00000000e+00, 1.00000000e+00, 0.00000000e+00, 5.10000000e+01,
        0.00000000e+00, 1.00000000e+00, 3.94000000e+01],
       [1.00000000e+00, 3.00000000e+00, 0.00000000e+00, 1.60000000e+01,
       1.00000000e+00, 1.00000000e+00, 8.51670000e+00],
       [1.00000000e+00, 1.00000000e+00, 0.00000000e+00, 3.90000000e+01,
        0.00000000e+00, 0.00000000e+00, 1.08900000e+02],
       [0.00000000e+00, 3.00000000e+00, 1.00000000e+00, 3.02725904e+01,
        0.00000000e+00, 0.00000000e+00, 8.05000000e+00],
       [0.00000000e+00, 3.00000000e+00, 1.00000000e+00, 3.02725904e+01,
        0.00000000e+00, 0.00000000e+00, 5.64958000e+01],
       [1.00000000e+00, 3.00000000e+00, 0.00000000e+00, 2.80000000e+01,
        0.00000000e+00, 0.00000000e+00, 7.77500000e+00],
       [0.00000000e+00, 1.00000000e+00, 1.00000000e+00, 5.50000000e+01,
        0.00000000e+00, 0.00000000e+00, 5.00000000e+01],
       [0.00000000e+00, 3.00000000e+00, 1.00000000e+00, 1.00000000e+01,
        4.00000000e+00, 1.00000000e+00, 2.91250000e+01],
       [0.00000000e+00, 2.00000000e+00, 1.00000000e+00, 2.30000000e+01,
        1.00000000e+00, 0.00000000e+00, 1.05000000e+01],
       [0.00000000e+00, 2.00000000e+00, 1.00000000e+00, 5.70000000e+01,
        0.00000000e+00, 0.00000000e+00, 1.30000000e+01],
       [0.00000000e+00, 1.00000000e+00, 1.00000000e+00, 4.10000000e+01,
        1.00000000e+00, 0.00000000e+00, 5.18625000e+01],
       [1.00000000e+00, 3.00000000e+00, 0.00000000e+00, 3.00000000e+00,
        1.00000000e+00, 1.00000000e+00, 1.37750000e+01],
       [0.00000000e+00, 2.00000000e+00, 1.00000000e+00, 3.00000000e+01,
        0.00000000e+00, 0.00000000e+00, 1.30000000e+01],
       [1.00000000e+00, 3.00000000e+00, 0.00000000e+00, 3.02725904e+01,
        0.00000000e+00, 2.00000000e+00, 1.52458000e+01],
       [1.00000000e+00, 3.00000000e+00, 0.00000000e+00, 1.85000000e+01,
        0.00000000e+00, 0.00000000e+00, 7.28330000e+00],
       [1.00000000e+00, 1.00000000e+00, 0.00000000e+00, 2.50000000e+01,
        1.00000000e+00, 0.00000000e+00, 5.54417000e+01],
       [0.00000000e+00, 1.00000000e+00, 1.00000000e+00, 3.02725904e+01,
        0.00000000e+00, 0.00000000e+00, 2.65500000e+01],
       [0.00000000e+00, 3.00000000e+00, 1.00000000e+00, 3.90000000e+01,
        0.00000000e+00, 2.00000000e+00, 7.22920000e+00],
```

```
y_test
```

```
    array([0, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 2, 2, 2, 2, 1, 2, 2, 2,
           2, 2, 0, 1, 0, 2, 0, 2, 2, 2, 2, 2, 2, 2, 1, 0, 2, 0, 1, 2, 0, 2,
           0, 2, 2, 0, 2, 2, 1, 0, 2, 2, 0, 2, 2, 2, 2, 1, 2, 2, 0, 2, 2, 2,
           2, 2, 0, 0, 0, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 0, 0, 1, 2, 2, 1, 2,
           2, 2, 0, 2, 0, 2, 2, 2, 2, 2, 1, 1, 2, 1, 2, 2, 2, 2, 2, 2, 0, 1,
           2, 1, 2, 2, 0, 0, 0, 2, 2, 2, 2, 2, 2, 0, 2, 2])
```

```
# standardization technique used for normalization
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
ss.fit(x_train)
x_train=ss.transform(x_train)
x_test=ss.transform(x_test)
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix,accuracy_score
knn=KNeighborsClassifier(n_neighbors=7)
nb=GaussianNB()
svm=SVC()
dtc=DecisionTreeClassifier(criterion='entropy')
rfc=RandomForestClassifier()
lst=[knn,nb,svm,dtc,rfc]
```

```
for i in lst:
  print(i)
  i.fit(x_train,y_train)
  y_pred=i.predict(x_test)
  print("-"*45)
  print("Confusion Matrix :\n",confusion_matrix(y_test,y_pred))
  print("Accuracy Score :",(accuracy_score(y_test,y_pred)*100),"%\n")
```

```
KNeighborsClassifier(n_neighbors=7)
---------------------------------------------
Confusion Matrix :
 [[ 7  1 19]
 [ 0  2 12]
 [ 6  4 75]]
Accuracy Score : 66.66666666666666 %

GaussianNB()
---------------------------------------------
Confusion Matrix :
 [[ 8 11  8]
 [ 0 13  1]
 [ 8 57 20]]
Accuracy Score : 32.53968253968254 %

SVC()
---------------------------------------------
Confusion Matrix :
 [[ 3  1 23]
 [ 0  0 14]
 [ 3  0 82]]
Accuracy Score : 67.46031746031747 %

DecisionTreeClassifier(criterion='entropy')
---------------------------------------------
Confusion Matrix :
 [[18  1  8]
 [ 0  6  8]
 [ 9  3 73]]
Accuracy Score : 76.98412698412699 %

RandomForestClassifier()
---------------------------------------------
Confusion Matrix :
 [[14  0 13]
 [ 1  5  8]
 [ 9  1 75]]
Accuracy Score : 74.60317460317461 %
```