predicting-heart-disease

January 30, 2024

[59]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
import sklearn.neighbors
from sklearn.neighbors import KNeighborsClassifier
```

[60]:
```
df = pd.read_csv('Heart_Disease_Prediction.csv')
```

[61]:
```
df.head()
```

[61]:

	Age	Sex	Chest pain type	BP	Cholesterol	FBS over 120	EKG results	\
0	70	1	4	130	322	0	2	
1	67	0	3	115	564	0	2	
2	57	1	2	124	261	0	0	
3	64	1	4	128	263	0	0	
4	74	0	2	120	269	0	2	

	Max HR	Exercise angina	ST depression	Slope of ST	\
0	109	0	2.4	2	
1	160	0	1.6	2	
2	141	0	0.3	1	
3	105	1	0.2	2	
4	121	1	0.2	1	

	Number of vessels fluro	Thallium	Heart Disease
0	3	3	Presence
1	0	7	Absence
2	0	7	Presence
3	1	7	Absence

```
4                        1         3       Absence
```

[62]:
```
df.describe()
```

[62]:
```
              Age         Sex  Chest pain type          BP  Cholesterol  \
count  270.000000  270.000000       270.000000  270.000000   270.000000
mean    54.433333    0.677778         3.174074  131.344444   249.659259
std      9.109067    0.468195         0.950090   17.861608    51.686237
min     29.000000    0.000000         1.000000   94.000000   126.000000
25%     48.000000    0.000000         3.000000  120.000000   213.000000
50%     55.000000    1.000000         3.000000  130.000000   245.000000
75%     61.000000    1.000000         4.000000  140.000000   280.000000
max     77.000000    1.000000         4.000000  200.000000   564.000000

       FBS over 120  EKG results      Max HR  Exercise angina  ST depression  \
count    270.000000   270.000000  270.000000       270.000000      270.00000
mean       0.148148     1.022222  149.677778         0.329630        1.05000
std        0.355906     0.997891   23.165717         0.470952        1.14521
min        0.000000     0.000000   71.000000         0.000000        0.00000
25%        0.000000     0.000000  133.000000         0.000000        0.00000
50%        0.000000     2.000000  153.500000         0.000000        0.80000
75%        0.000000     2.000000  166.000000         1.000000        1.60000
max        1.000000     2.000000  202.000000         1.000000        6.20000

       Slope of ST  Number of vessels fluro    Thallium
count   270.000000               270.000000  270.000000
mean      1.585185                 0.670370    4.696296
std       0.614390                 0.943896    1.940659
min       1.000000                 0.000000    3.000000
25%       1.000000                 0.000000    3.000000
50%       2.000000                 0.000000    3.000000
75%       2.000000                 1.000000    7.000000
max       3.000000                 3.000000    7.000000
```

[63]:
```
df.shape
```

[63]:
```
(270, 14)
```

[64]:
```
df.isnull().values.any()
```

[64]:
```
False
```

[65]:
```
df['Sex'].value_counts()
```

[65]:
```
Sex
1    183
0     87
```

```
Name: count, dtype: int64
```

[66]:
```
df['Heart Disease'].value_counts()
```

[66]:
```
Heart Disease
Absence     150
Presence    120
Name: count, dtype: int64
```

[68]:
```
df.isnull().sum()
```

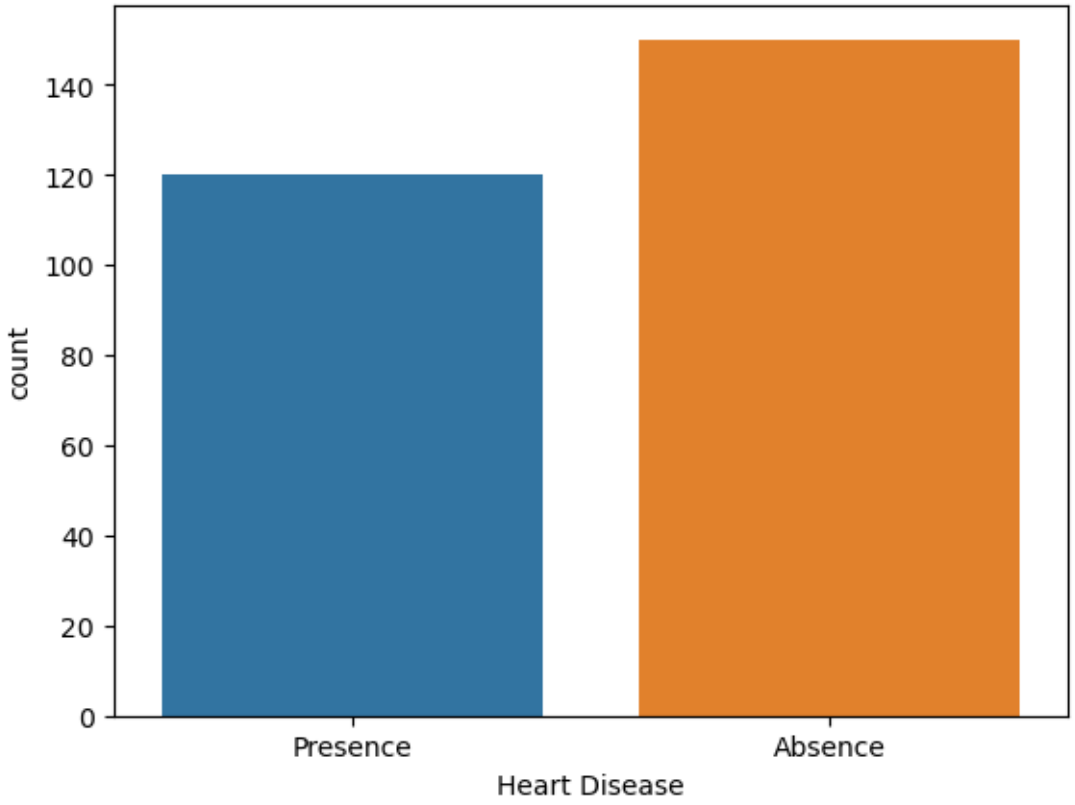
[68]:
```
Age                       0
Sex                       0
Chest pain type           0
BP                        0
Cholesterol               0
FBS over 120              0
EKG results               0
Max HR                    0
Exercise angina           0
ST depression             0
Slope of ST               0
Number of vessels fluro   0
Thallium                  0
Heart Disease             0
dtype: int64
```

[70]:
```
df.nunique()
```

[70]:
```
Age                        41
Sex                         2
Chest pain type             4
BP                         47
Cholesterol               144
FBS over 120                2
EKG results                 3
Max HR                     90
Exercise angina             2
ST depression              39
Slope of ST                 3
Number of vessels fluro     4
Thallium                    3
Heart Disease               2
dtype: int64
```

[69]:
```
sns.countplot(x='Heart Disease', data=df)
```

[69]: <Axes: xlabel='Heart Disease', ylabel='count'>



[71]:
```
df.head()
```

[71]:

	Age	Sex	Chest pain type	BP	Cholesterol	FBS over 120	EKG results	Max HR	Exercise angina	ST depression	Slope of ST	Number of vessels fluro	Thallium	Heart Disease
0	70	1	4	130	322	0	2	109	0	2.4	2	3	3	Presence
1	67	0	3	115	564	0	2	160	0	1.6	2	0	7	Absence
2	57	1	2	124	261	0	0	141	0	0.3	1			
3	64	1	4	128	263	0	0	105	1	0.2	2			
4	74	0	2	120	269	0	2	121	1	0.2	1			

```
2                       0         7      Presence
3                       1         7       Absence
4                       1         3       Absence
```

[ ]:

1 K-Nearest Neighbours

[72]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state = 0,␣
↪test_size = 0.35)
```

[73]:
```
sc_x = StandardScaler()
x_train = sc_x.fit_transform(x_train)
x_test = sc_x.transform(x_test)
```

[91]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=0.95)
x_train_pca = pca.fit_transform(x_train)
x_test_pca = pca.transform(x_test)
```

[92]:
```
import math
math.sqrt(len(y_test))
```

[92]:
```
9.746794344808963
```

[93]:
```
# Creating KNN Model.
classifier = KNeighborsClassifier()
classifier.fit(x_train,y_train)
```

[93]:
```
KNeighborsClassifier()
```

[94]:
```
y_pred = classifier.predict(x_test)
y_pred
```

[94]:
```
array(['Absence', 'Absence', 'Presence', 'Presence', 'Absence', 'Absence',
       'Absence', 'Absence', 'Presence', 'Absence', 'Absence', 'Presence',
       'Presence', 'Presence', 'Presence', 'Absence', 'Absence',
       'Presence', 'Presence', 'Absence', 'Presence', 'Absence',
       'Presence', 'Presence', 'Absence', 'Presence', 'Absence',
       'Absence', 'Absence', 'Absence', 'Presence', 'Presence',
       'Presence', 'Absence', 'Presence', 'Absence', 'Presence',
       'Presence', 'Absence', 'Absence', 'Absence', 'Absence', 'Absence',
       'Absence', 'Absence', 'Absence', 'Presence', 'Absence', 'Presence',
       'Presence', 'Presence', 'Presence', 'Absence', 'Absence',
       'Absence', 'Absence', 'Absence', 'Presence', 'Absence', 'Presence',
       'Absence', 'Absence', 'Absence', 'Absence', 'Absence', 'Absence',
```

```
       'Absence', 'Presence', 'Presence', 'Absence', 'Presence',
       'Absence', 'Presence', 'Absence', 'Presence', 'Presence',
       'Presence', 'Absence', 'Absence', 'Presence', 'Presence',
       'Presence', 'Presence', 'Absence', 'Presence', 'Absence',
       'Presence', 'Absence', 'Presence', 'Absence', 'Presence',
       'Absence', 'Absence', 'Presence', 'Absence'], dtype=object)
```

[95]:
```
cm = confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[40 15]
 [13 27]]
```

[96]:
```
print(accuracy_score(y_test,y_pred))
```

```
0.7052631578947368
```

2 SVM Model

[79]:
```
# Creating SVM model.
from sklearn import svm
clf = svm.SVC(kernel='rbf')
clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

[80]:
```
y_pred = clf.predict(x_test)
y_pred
```

[80]:
```
array(['Absence', 'Absence', 'Presence', 'Presence', 'Absence', 'Absence',
       'Absence', 'Absence', 'Presence', 'Absence', 'Absence', 'Absence',
       'Presence', 'Presence', 'Presence', 'Absence', 'Absence',
       'Presence', 'Presence', 'Absence', 'Presence', 'Absence',
       'Presence', 'Presence', 'Presence', 'Absence', 'Absence',
       'Absence', 'Absence', 'Absence', 'Presence', 'Absence', 'Presence',
       'Absence', 'Presence', 'Absence', 'Presence', 'Presence',
       'Absence', 'Absence', 'Absence', 'Absence', 'Absence', 'Absence',
       'Absence', 'Absence', 'Presence', 'Absence', 'Presence',
       'Presence', 'Presence', 'Presence', 'Absence', 'Absence',
       'Absence', 'Absence', 'Absence', 'Presence', 'Absence', 'Presence',
       'Absence', 'Absence', 'Absence', 'Absence', 'Absence', 'Presence',
       'Presence', 'Presence', 'Absence', 'Absence', 'Presence',
       'Absence', 'Presence', 'Absence', 'Presence', 'Absence',
       'Presence', 'Absence', 'Absence', 'Presence', 'Presence',
       'Absence', 'Presence', 'Absence', 'Presence', 'Absence',
       'Presence', 'Presence', 'Presence', 'Absence', 'Presence',
       'Absence', 'Absence', 'Presence', 'Absence'], dtype=object)
```

[81]:
```
cm = confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[44 11]
 [11 29]]
```

[82]:
```
print(accuracy_score(y_test,y_pred))
```

```
0.7684210526315789
```

[105]:
```
for i in range(len(y_test)):
    if y_pred[i] != y_test.iloc[i]:
        print("Index:", i, "Prediction:", y_pred[i], "Actual:", y_test.iloc[i])
```

```
Index: 2 Prediction: Presence Actual: Absence
Index: 6 Prediction: Absence Actual: Presence
Index: 8 Prediction: Presence Actual: Absence
Index: 11 Prediction: Presence Actual: Absence
Index: 12 Prediction: Presence Actual: Absence
Index: 19 Prediction: Absence Actual: Presence
Index: 23 Prediction: Presence Actual: Absence
Index: 24 Prediction: Absence Actual: Presence
Index: 25 Prediction: Presence Actual: Absence
Index: 29 Prediction: Absence Actual: Presence
Index: 30 Prediction: Presence Actual: Absence
Index: 31 Prediction: Presence Actual: Absence
Index: 35 Prediction: Absence Actual: Presence
Index: 39 Prediction: Absence Actual: Presence
Index: 40 Prediction: Absence Actual: Presence
Index: 43 Prediction: Absence Actual: Presence
Index: 45 Prediction: Absence Actual: Presence
Index: 46 Prediction: Presence Actual: Absence
Index: 51 Prediction: Presence Actual: Absence
Index: 53 Prediction: Absence Actual: Presence
Index: 65 Prediction: Absence Actual: Presence
Index: 68 Prediction: Presence Actual: Absence
Index: 70 Prediction: Presence Actual: Absence
Index: 75 Prediction: Presence Actual: Absence
Index: 82 Prediction: Presence Actual: Absence
Index: 86 Prediction: Presence Actual: Absence
Index: 87 Prediction: Absence Actual: Presence
Index: 92 Prediction: Absence Actual: Presence
```

[99]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

for i in range(1, 26):
    neigh = KNeighborsClassifier(n_neighbors=i, weights='uniform')
```

```
    neigh.fit(x_train, y_train)

    # Use the previously trained model to make predictions on the training set
    y_pred_train = neigh.predict(x_train)
    y_pred_test = neigh.predict(x_test)

    score_test = accuracy_score(y_pred_test, y_test)
    print('for', i, 'test accuracy is ', score_test)

    score_train = accuracy_score(y_pred_train, y_train)
    print('train accuracy is ', score_train)
```

```
for 1 test accuracy is  0.7157894736842105
train accuracy is  1.0
for 2 test accuracy is  0.7684210526315789
train accuracy is  0.8742857142857143
for 3 test accuracy is  0.6631578947368421
train accuracy is  0.88
for 4 test accuracy is  0.7157894736842105
train accuracy is  0.8742857142857143
for 5 test accuracy is  0.7052631578947368
train accuracy is  0.84
for 6 test accuracy is  0.7263157894736842
train accuracy is  0.8342857142857143
for 7 test accuracy is  0.7368421052631579
train accuracy is  0.8457142857142858
for 8 test accuracy is  0.7473684210526316
train accuracy is  0.8342857142857143
for 9 test accuracy is  0.7578947368421053
train accuracy is  0.8342857142857143
for 10 test accuracy is  0.7368421052631579
train accuracy is  0.84
for 11 test accuracy is  0.7368421052631579
train accuracy is  0.84
for 12 test accuracy is  0.7578947368421053
train accuracy is  0.8285714285714286
for 13 test accuracy is  0.7578947368421053
train accuracy is  0.8457142857142858
for 14 test accuracy is  0.7578947368421053
train accuracy is  0.8457142857142858
for 15 test accuracy is  0.7368421052631579
train accuracy is  0.8571428571428571
for 16 test accuracy is  0.7473684210526316
train accuracy is  0.8342857142857143
for 17 test accuracy is  0.7368421052631579
train accuracy is  0.8457142857142858
for 18 test accuracy is  0.7578947368421053
```

```
train accuracy is  0.8514285714285714
for 19 test accuracy is  0.7578947368421053
train accuracy is  0.8571428571428571
for 20 test accuracy is  0.7684210526315789
train accuracy is  0.8342857142857143
for 21 test accuracy is  0.7684210526315789
train accuracy is  0.84
for 22 test accuracy is  0.7684210526315789
train accuracy is  0.84
for 23 test accuracy is  0.7578947368421053
train accuracy is  0.84
for 24 test accuracy is  0.7473684210526316
train accuracy is  0.8342857142857143
for 25 test accuracy is  0.7684210526315789
train accuracy is  0.8342857142857143
```

[100]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
parameters = {'n_neighbors': [5,10,15,20,26], 'weights': ['uniform',␣
 ↪'distance'], 'p': [1, 2, 3, 4, 5]}

classifier = KNeighborsClassifier()
clf = GridSearchCV(classifier, parameters)

clf.fit(x_train, y_train)
```

[100]:
```
GridSearchCV(estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [5, 10, 15, 20, 26],
                         'p': [1, 2, 3, 4, 5],
                         'weights': ['uniform', 'distance']})
```

[101]:
```
clf.best_params_
```

[101]:
```
{'n_neighbors': 5, 'p': 1, 'weights': 'distance'}
```

[102]:
```
new=list(x_test)
new
```

[102]:
```
[array([ 1.01962288,  0.6950186 , -2.27563089,  0.83158612, -0.3035494 ,
         2.29128785,  0.96076892, -0.02262845, -0.70407847,  1.0261241 ,
         2.35351575, -0.68375373]),
 array([-0.827745  , -1.43881042, -0.16297622,  0.64853257, -1.35582535,
        -0.43643578,  0.96076892,  0.40780408,  1.42029623,  0.29050942,
         2.35351575, -0.68375373]),
 array([ 1.12829158, -1.43881042,  0.89335112, -0.08368162,  1.01179554,
        -0.43643578, -1.040833  , -1.22783955, -0.70407847,  0.78091921,
         0.71261832,  1.4722265 ]),
```

```
array([ 1.78030377,  0.6950186 , -0.16297622,  1.74685385,  0.37291371,
       -0.43643578, -1.040833  , -1.65827208,  1.42029623,  1.51653389,
        0.71261832,  0.39423638]),
array([-0.28440151,  0.6950186 , -0.16297622, -1.30403861, -1.39340664,
       -0.43643578, -1.040833  , -1.1847963 , -0.70407847, -0.3633703 ,
       -0.92827912, -0.68375373]),
array([ 1.67163507, -1.43881042, -2.27563089,  0.52649687, -0.19080555,
       -0.43643578, -1.040833  ,  0.0204148 , -0.70407847,  0.61744928,
       -0.92827912,  1.4722265 ]),
array([ 0.04160459,  0.6950186 , -1.21930355,  3.69942503,  0.6359827 ,
       -0.43643578,  0.96076892,  1.91431795, -0.70407847, -0.85378009,
       -0.92827912,  0.39423638]),
array([-1.2624198 ,  0.6950186 , -0.16297622, -0.69386011, -0.17201491,
        2.29128785, -1.040833  ,  1.8712747 , -0.70407847, -0.19990037,
        2.35351575, -0.68375373]),
array([ 0.80228548,  0.6950186 , -0.16297622,  1.13667536, -0.11564298,
        2.29128785, -1.040833  , -0.58219075,  1.42029623, -0.03643044,
        0.71261832, -0.68375373]),
array([ 1.88897247, -1.43881042,  0.89335112, -1.18200291, -1.88196333,
       -0.43643578, -1.040833  , -1.09870979, -0.70407847,  0.45397935,
        0.71261832, -0.68375373]),
array([-0.28440151, -1.43881042, -0.16297622,  0.52649687,  1.10574875,
       -0.43643578,  0.96076892, -0.36697448, -0.70407847,  0.37224438,
       -0.92827912,  0.39423638]),
array([ 0.36761068,  0.6950186 ,  0.89335112,  0.03835408, -0.79210609,
       -0.43643578, -1.040833  ,  0.75215011,  1.42029623, -0.85378009,
       -0.92827912, -0.68375373]),
array([ 0.04160459,  0.6950186 , -0.16297622, -0.38877087,  0.44807628,
       -0.43643578,  0.96076892,  0.06345805, -0.70407847, -0.44510527,
        2.35351575,  0.39423638]),
array([ 0.36761068,  0.6950186 ,  0.89335112,  2.0519431 ,  0.74872655,
        2.29128785,  0.96076892, -1.14175304, -0.70407847, -0.03643044,
        0.71261832,  2.55021661]),
array([ 0.69361678,  0.6950186 ,  0.89335112,  0.52649687,  0.82388912,
       -0.43643578,  0.96076892,  0.83823662, -0.70407847,  0.12703949,
        0.71261832,  1.4722265 ]),
array([-0.39307021, -1.43881042,  0.89335112, -1.30403861,  0.09105408,
       -0.43643578,  0.96076892,  0.36476083, -0.70407847, -0.85378009,
       -0.92827912, -0.68375373]),
array([ 0.47627938,  0.6950186 , -0.16297622,  0.52649687, -0.71694352,
        2.29128785,  0.96076892,  0.62302035, -0.70407847, -0.85378009,
       -0.92827912, -0.68375373]),
array([ 2.54098466,  0.6950186 ,  0.89335112, -0.38877087,  1.03058618,
       -0.43643578,  0.96076892,  0.49389059,  1.42029623, -0.85378009,
       -0.92827912,  2.55021661]),
array([ 0.91095418,  0.6950186 , -1.21930355, -0.69386011,  0.59840142,
       -0.43643578,  0.96076892, -2.04566136, -0.70407847,  0.29050942,
```

```
        0.71261832,  0.39423638]),
array([-0.61040761,  0.6950186 , -1.21930355, -1.30403861, -0.37871197,
       -0.43643578, -1.040833  ,  0.75215011, -0.70407847, -0.03643044,
        2.35351575, -0.68375373]),
array([-0.28440151,  0.6950186 ,  0.89335112,  0.52649687,  0.91784233,
       -0.43643578, -1.040833  , -1.22783955,  1.42029623,  2.57908844,
        0.71261832,  2.55021661]),
array([-1.3710885 , -1.43881042, -1.21930355, -1.60912785, -0.96122187,
       -0.43643578, -1.040833  ,  0.75215011, -0.70407847, -0.85378009,
       -0.92827912,  0.39423638]),
array([ 0.58494808,  0.6950186 ,  0.89335112, -1.30403861, -0.19080555,
       -0.43643578,  0.96076892, -0.36697448,  1.42029623,  0.12703949,
        0.71261832,  0.39423638]),
array([-0.9364137 ,  0.6950186 ,  0.89335112, -1.6701457 , -0.77331545,
       -0.43643578,  0.96076892, -0.10871496,  1.42029623,  1.59826886,
        0.71261832, -0.68375373]),
array([-1.0450824 ,  0.6950186 ,  0.89335112, -1.18200291,  0.76751719,
       -0.43643578,  0.96076892,  0.10650131, -0.70407847, -0.85378009,
       -0.92827912,  0.39423638]),
array([-1.3710885 ,  0.6950186 , -0.16297622, -0.08368162, -0.6605716 ,
       -0.43643578,  0.96076892,  0.75215011, -0.70407847,  0.78091921,
        0.71261832, -0.68375373]),
array([ 0.04160459,  0.6950186 ,  0.89335112,  0.52649687, -0.19080555,
       -0.43643578, -1.040833  ,  0.40780408, -0.70407847,  0.12703949,
       -0.92827912, -0.68375373]),
array([ 1.45429767, -1.43881042, -0.16297622,  1.25871106,  0.52323885,
       -0.43643578, -1.040833  ,  0.92432312, -0.70407847, -0.85378009,
       -0.92827912,  0.39423638]),
array([ 0.91095418, -1.43881042,  0.89335112,  0.52649687,  2.72174396,
       -0.43643578,  0.96076892,  0.27867432, -0.70407847,  0.12703949,
        0.71261832, -0.68375373]),
array([ 1.45429767,  0.6950186 ,  0.89335112, -0.69386011, -0.22838683,
       -0.43643578, -1.040833  , -3.42304547, -0.70407847, -0.03643044,
        0.71261832, -0.68375373]),
array([ 1.34562898,  0.6950186 ,  0.89335112, -0.69386011,  0.9930049 ,
       -0.43643578,  0.96076892,  0.0204148 , -0.70407847, -0.52684023,
        0.71261832, -0.68375373]),
array([ 1.45429767, -1.43881042,  0.89335112, -1.54811   , -0.49145582,
       -0.43643578, -1.040833  , -0.36697448, -0.70407847, -0.6085752 ,
       -0.92827912,  1.4722265 ]),
array([ 0.69361678,  0.6950186 ,  0.89335112, -0.08368162, -0.81089673,
       -0.43643578,  0.96076892, -0.79740701,  1.42029623,  1.10785907,
        0.71261832,  1.4722265 ]),
array([-1.3710885 , -1.43881042, -1.21930355, -0.08368162, -0.84847802,
       -0.43643578,  0.96076892,  0.92432312, -0.70407847,  0.29050942,
       -0.92827912, -0.68375373]),
array([-1.1537511 , -1.43881042,  0.89335112,  0.03835408,  1.72583994,
```

```
        2.29128785,  0.96076892, -0.625234  ,  1.42029623,  1.59826886,
        0.71261832, -0.68375373]),
array([ 0.91095418, -1.43881042, -0.16297622, -0.08368162,  0.26016986,
       -0.43643578, -1.040833  , -2.30392088, -0.70407847,  0.12703949,
        0.71261832,  0.39423638]),
array([ 0.04160459,  0.6950186 ,  0.89335112, -0.44978872,  0.31654179,
       -0.43643578,  0.96076892, -1.78740184,  1.42029623,  0.94438914,
        0.71261832,  0.39423638]),
array([ 1.34562898, -1.43881042,  0.89335112,  2.84517514, -0.39750261,
        2.29128785, -1.040833  ,  0.62302035,  1.42029623, -0.03643044,
        0.71261832,  1.4722265 ]),
array([-1.0450824 ,  0.6950186 , -1.21930355, -0.69386011, -0.54782775,
       -0.43643578, -1.040833  ,  0.83823662, -0.70407847, -0.85378009,
       -0.92827912, -0.68375373]),
array([-1.3710885 ,  0.6950186 ,  0.89335112, -1.30403861, -1.44977856,
       -0.43643578,  0.96076892,  0.32171757, -0.70407847, -0.85378009,
       -0.92827912, -0.68375373]),
array([-0.827745  ,  0.6950186 ,  0.89335112, -0.69386011, -0.00289913,
       -0.43643578,  0.96076892, -0.28088797, -0.70407847, -0.19990037,
       -0.92827912, -0.68375373]),
array([-0.28440151, -1.43881042, -0.16297622, -0.69386011,  0.8614704 ,
       -0.43643578,  0.96076892,  0.27867432, -0.70407847, -0.3633703 ,
       -0.92827912, -0.68375373]),
array([-0.28440151, -1.43881042, -0.16297622, -0.08368162,  0.12863537,
       -0.43643578,  0.96076892, -0.06567171, -0.70407847, -0.44510527,
       -0.92827912, -0.68375373]),
array([ 0.80228548, -1.43881042,  0.89335112,  0.83158612,  1.08695811,
       -0.43643578,  0.96076892, -0.19480147,  1.42029623, -0.03643044,
        0.71261832, -0.68375373]),
array([-0.06706411, -1.43881042,  0.89335112,  0.40446117, -0.28475876,
       -0.43643578,  0.96076892,  0.40780408, -0.70407847, -0.85378009,
       -0.92827912, -0.68375373]),
array([ 0.58494808,  0.6950186 , -2.27563089,  1.74685385,  0.44807628,
       -0.43643578,  0.96076892, -1.09870979, -0.70407847, -0.85378009,
       -0.92827912, -0.68375373]),
array([-0.17573281,  0.6950186 ,  0.89335112, -1.4260743 , -0.3035494 ,
        2.29128785, -1.040833  , -0.15175821, -0.70407847, -0.77204513,
       -0.92827912,  2.55021661]),
array([ 0.04160459,  0.6950186 , -0.16297622,  1.13667536, -0.32234004,
       -0.43643578,  0.96076892,  0.62302035, -0.70407847,  0.45397935,
       -0.92827912, -0.68375373]),
array([ 1.45429767,  0.6950186 ,  0.89335112, -0.69386011, -0.37871197,
       -0.43643578,  0.96076892, -0.92653677,  1.42029623,  1.271329  ,
        0.71261832,  1.4722265 ]),
array([ 1.56296637,  0.6950186 , -0.16297622,  2.96721084,  0.46686692,
        2.29128785,  0.96076892, -0.02262845,  1.42029623,  0.45397935,
        0.71261832, -0.68375373]),
```

```
array([ 0.25894199, -1.43881042,  0.89335112,  0.16038978,  3.00360359,
       -0.43643578,  0.96076892, -0.02262845,  1.42029623,  0.69918424,
        0.71261832,  1.4722265 ]),
array([ 0.47627938,  0.6950186 , -0.16297622, -1.60912785, -0.17201491,
       -0.43643578,  0.96076892,  0.14954456,  1.42029623, -0.3633703 ,
        0.71261832, -0.68375373]),
array([ 0.69361678, -1.43881042, -0.16297622, -1.7921814 ,  1.29365517,
       -0.43643578, -1.040833  ,  0.40780408, -0.70407847, -0.85378009,
       -0.92827912,  0.39423638]),
array([-0.17573281,  0.6950186 ,  0.89335112, -1.18200291, -0.35992133,
       -0.43643578, -1.040833  ,  0.40780408, -0.70407847, -0.85378009,
       -0.92827912,  0.39423638]),
array([ 1.23696028,  0.6950186 ,  0.89335112, -0.69386011, -1.35582535,
       -0.43643578, -1.040833  , -0.45306099, -0.70407847, -0.52684023,
       -0.92827912, -0.68375373]),
array([-1.1537511 ,  0.6950186 ,  0.89335112, -0.99894936,  1.01179554,
       -0.43643578, -1.040833  ,  1.3117124 , -0.70407847,  0.12703949,
        0.71261832, -0.68375373]),
array([ 0.47627938, -1.43881042,  0.89335112, -1.9142171 , -0.02168977,
       -0.43643578,  0.96076892, -1.22783955, -0.70407847, -0.03643044,
        0.71261832, -0.68375373]),
array([ 0.15027329, -1.43881042,  0.89335112,  2.96721084,  1.46277095,
       -0.43643578, -0.04003204, -1.44305582,  1.42029623,  1.92520872,
        0.71261832, -0.68375373]),
array([-1.0450824 ,  0.6950186 , -0.16297622,  0.52649687, -0.26596812,
       -0.43643578,  0.96076892,  1.26866915, -0.70407847, -0.85378009,
       -0.92827912, -0.68375373]),
array([ 0.47627938,  0.6950186 ,  0.89335112, -0.38877087,  0.95542361,
       -0.43643578,  0.96076892,  0.88127987, -0.70407847, -0.85378009,
       -0.92827912,  1.4722265 ]),
array([ 0.69361678, -1.43881042, -2.27563089,  1.13667536, -0.17201491,
       -0.43643578, -1.040833  ,  0.88127987, -0.70407847, -0.11816541,
       -0.92827912, -0.68375373]),
array([-0.39307021,  0.6950186 , -0.16297622, -0.14469947, -0.99880315,
       -0.43643578, -1.040833  ,  0.53693384, -0.70407847, -0.85378009,
       -0.92827912, -0.68375373]),
array([ 0.58494808,  0.6950186 , -1.21930355,  0.52649687, -0.5290371 ,
       -0.43643578, -1.040833  ,  0.57997709,  1.42029623, -0.85378009,
       -0.92827912, -0.68375373]),
array([-1.3710885 ,  0.6950186 , -1.21930355, -1.30403861, -0.26596812,
       -0.43643578, -1.040833  ,  0.10650131, -0.70407847, -0.85378009,
       -0.92827912, -0.68375373]),
array([ 2.43231597, -1.43881042, -0.16297622,  0.52649687, -0.98001251,
       -0.43643578, -0.04003204, -1.48609907, -0.70407847,  0.04530452,
        0.71261832, -0.68375373]),
array([ 0.58494808,  0.6950186 , -0.16297622, -0.32775302, -0.58540903,
        2.29128785, -1.040833  , -0.71132051, -0.70407847,  0.94438914,
```

```
        0.71261832,  0.39423638]),
array([ 1.56296637,  0.6950186 , -0.16297622, -0.81589581,  0.52323885,
       -0.43643578, -1.040833  ,  0.0204148 , -0.70407847, -0.03643044,
       -0.92827912,  0.39423638]),
array([ 1.67163507,  0.6950186 , -0.16297622,  0.52649687,  0.09105408,
       -0.43643578,  0.96076892, -0.19480147, -0.70407847,  0.78091921,
        0.71261832,  2.55021661]),
array([ 0.25894199,  0.6950186 , -2.27563089, -0.69386011, -1.05517508,
       -0.43643578,  0.96076892,  0.49389059, -0.70407847,  0.69918424,
        0.71261832, -0.68375373]),
array([ 0.04160459, -1.43881042, -0.16297622,  1.74685385, -0.90484994,
       -0.43643578, -1.040833  ,  0.53693384, -0.70407847, -0.85378009,
       -0.92827912,  0.39423638]),
array([-0.28440151,  0.6950186 , -0.16297622, -1.9142171 , -0.51024646,
       -0.43643578, -1.040833  , -0.32393123,  1.42029623,  0.12703949,
        0.71261832, -0.68375373]),
array([ 1.12829158,  0.6950186 , -2.27563089, -1.30403861, -0.71694352,
       -0.43643578,  0.96076892, -0.28088797,  1.42029623,  0.61744928,
        0.71261832, -0.68375373]),
array([ 0.25894199,  0.6950186 ,  0.89335112, -0.38877087, -0.00289913,
        2.29128785,  0.96076892, -0.28088797,  1.42029623,  0.12703949,
        0.71261832,  0.39423638]),
array([ 1.88897247, -1.43881042, -1.21930355,  1.74685385,  0.9930049 ,
       -0.43643578, -1.040833  ,  0.49389059, -0.70407847, -0.52684023,
       -0.92827912,  1.4722265 ]),
array([-0.39307021,  0.6950186 ,  0.89335112,  1.13667536, -0.11564298,
       -0.43643578,  0.96076892, -0.96958003, -0.70407847,  1.271329  ,
        0.71261832, -0.68375373]),
array([ 0.36761068, -1.43881042,  0.89335112, -0.69386011,  1.97011828,
       -0.43643578, -1.040833  ,  0.53693384,  1.42029623, -0.3633703 ,
       -0.92827912, -0.68375373]),
array([ 0.58494808,  0.6950186 ,  0.89335112,  0.52649687, -1.35582535,
       -0.43643578, -1.040833  ,  0.49389059,  1.42029623, -0.85378009,
       -0.92827912,  0.39423638]),
array([ 0.36761068,  0.6950186 , -0.16297622,  1.13667536, -1.52494113,
       -0.43643578, -1.040833  ,  1.01040963, -0.70407847,  0.45397935,
       -0.92827912, -0.68375373]),
array([-1.1537511 ,  0.6950186 ,  0.89335112,  1.13667536, -0.04048041,
       -0.43643578, -1.040833  ,  0.88127987, -0.70407847,  0.37224438,
       -0.92827912, -0.68375373]),
array([ 0.91095418,  0.6950186 ,  0.89335112, -0.69386011,  0.33533243,
       -0.43643578, -1.040833  , -2.21783438,  1.42029623,  0.61744928,
        0.71261832,  1.4722265 ]),
array([ 0.58494808,  0.6950186 ,  0.89335112,  2.35703235,  1.44398031,
       -0.43643578,  0.96076892, -0.45306099,  1.42029623,  1.92520872,
        2.35351575, -0.68375373]),
array([ 0.47627938,  0.6950186 ,  0.89335112, -1.9142171 , -0.28475876,
```

```
       -0.43643578, -1.040833   ,  0.23563107, -0.70407847, -0.77204513,
       -0.92827912,  0.39423638]),
 array([-0.28440151,  0.6950186 , -0.16297622, -2.28032419, -0.41629325,
       -0.43643578, -1.040833   ,  0.14954456,  1.42029623, -0.85378009,
       -0.92827912,  0.39423638]),
 array([-0.9364137 ,  0.6950186 ,  0.89335112, -0.99894936,  0.20379793,
       -0.43643578,  0.96076892,  1.48388542, -0.70407847, -0.85378009,
       -0.92827912, -0.68375373]),
 array([ 0.04160459,  0.6950186 ,  0.89335112, -0.69386011, -1.14912829,
       -0.43643578, -1.040833   , -1.61522883, -0.70407847,  0.29050942,
        0.71261832,  0.39423638]),
 array([-0.61040761,  0.6950186 , -1.21930355, -0.08368162, -0.0780617 ,
       -0.43643578,  0.96076892,  1.26866915, -0.70407847, -0.69031016,
        0.71261832, -0.68375373]),
 array([ 0.91095418,  0.6950186 , -0.16297622, -0.08368162, -0.34113068,
       -0.43643578, -1.040833   , -0.19480147, -0.70407847,  0.61744928,
        0.71261832,  2.55021661]),
 array([ 0.91095418, -1.43881042,  0.89335112,  1.13667536, -0.09685234,
       -0.43643578, -1.040833   ,  0.14954456,  1.42029623,  0.29050942,
        0.71261832, -0.68375373]),
 array([-0.827745  ,  0.6950186 ,  0.89335112,  0.52649687,  1.16212068,
       -0.43643578, -1.040833   , -1.31392606,  1.42029623,  0.61744928,
        0.71261832,  1.4722265 ]),
 array([ 1.34562898, -1.43881042, -2.27563089,  1.13667536, -0.43508389,
       -0.43643578, -1.040833   , -1.57218558, -0.70407847,  1.271329  ,
        2.35351575, -0.68375373]),
 array([-0.9364137 ,  0.6950186 ,  0.89335112,  0.64853257,  1.12453939,
       -0.43643578,  0.96076892, -0.15175821,  1.42029623, -0.85378009,
        0.71261832,  2.55021661]),
 array([-0.61040761,  0.6950186 , -0.16297622, -0.44978872,  0.10984472,
        2.29128785, -1.040833   ,  1.05345288, -0.70407847, -0.85378009,
       -0.92827912,  1.4722265 ]),
 array([ 0.58494808,  0.6950186 , -2.27563089,  2.35703235,  0.72993591,
       -0.43643578,  0.96076892,  0.36476083, -0.70407847, -0.69031016,
        0.71261832, -0.68375373]),
 array([-0.06706411,  0.6950186 ,  0.89335112, -0.51080657,  0.61719206,
       -0.43643578, -1.040833   , -2.39000739,  1.42029623,  0.78091921,
        0.71261832,  1.4722265 ]),
 array([-0.9364137 , -1.43881042, -1.21930355, -1.18200291, -1.67526627,
       -0.43643578, -1.040833   , -0.53914749, -0.70407847, -0.85378009,
        0.71261832, -0.68375373])]
```

2.0.1 Random Forest

[106]:
```
from sklearn.ensemble import RandomForestClassifier
```

[107]:
```
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(x_train, y_train)
```

[107]:
```
RandomForestClassifier(random_state=42)
```

[108]:
```
y_pred = rf_classifier.predict(x_test)
y_pred
```

[108]:
```
array(['Absence', 'Absence', 'Presence', 'Presence', 'Absence',
       'Presence', 'Absence', 'Absence', 'Presence', 'Absence', 'Absence',
       'Absence', 'Presence', 'Presence', 'Presence', 'Absence',
       'Absence', 'Presence', 'Presence', 'Absence', 'Presence',
       'Absence', 'Presence', 'Presence', 'Presence', 'Absence',
       'Absence', 'Absence', 'Presence', 'Presence', 'Absence',
       'Presence', 'Presence', 'Absence', 'Presence', 'Absence',
       'Presence', 'Presence', 'Absence', 'Absence', 'Absence', 'Absence',
       'Absence', 'Presence', 'Absence', 'Absence', 'Presence', 'Absence',
       'Presence', 'Presence', 'Presence', 'Presence', 'Absence',
       'Presence', 'Absence', 'Presence', 'Absence', 'Presence',
       'Absence', 'Presence', 'Absence', 'Absence', 'Absence', 'Absence',
       'Absence', 'Presence', 'Presence', 'Presence', 'Absence',
       'Absence', 'Presence', 'Presence', 'Presence', 'Absence',
       'Presence', 'Presence', 'Presence', 'Absence', 'Absence',
       'Presence', 'Presence', 'Presence', 'Presence', 'Absence',
       'Presence', 'Absence', 'Presence', 'Presence', 'Presence',
       'Absence', 'Presence', 'Absence', 'Absence', 'Presence', 'Absence'],
      dtype=object)
```

[109]:
```
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

[109]:
```
0.7578947368421053
```

[110]:
```
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid,␣
 ↪cv=5, scoring='accuracy')
grid_search.fit(x_train, y_train)
```

[110]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42),
             param_grid={'max_depth': [None, 10, 20, 30],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
```

```
                                'n_estimators': [50, 100, 200]},
                    scoring='accuracy')
```

[111]:
```
best_params = grid_search.best_params_
best_params
```

[111]:
```
{'max_depth': None,
 'min_samples_leaf': 1,
 'min_samples_split': 10,
 'n_estimators': 100}
```

[112]:
```
best_rf_classifier = RandomForestClassifier(**best_params, random_state=42)
best_rf_classifier.fit(x_train, y_train)
y_pred = best_rf_classifier.predict(x_test)
```

[113]:
```
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

[113]:
```
0.7894736842105263
```

2.0.2 Decision Tree

[114]:
```
from sklearn.tree import DecisionTreeClassifier
```

[115]:
```
deci=DecisionTreeClassifier()
deci.fit(x_train,y_train)
```

[115]:
```
DecisionTreeClassifier()
```

[116]:
```
y_pred=deci.predict(x_test)
```

[117]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
```

[120]:
```
score=accuracy_score(y_train,deci.predict(x_train))
score
```

[120]:
```
1.0
```

[121]:
```
score=accuracy_score(y_test,y_pred)
score
```

[121]:
```
0.6631578947368421
```

[122]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

```

```
best_accuracy_dt = 0
best_depth_dt = 1

for depth in range(1, 26):
    dt_classifier = DecisionTreeClassifier(max_depth=depth)
    dt_classifier.fit(x_train, y_train)
    y_pred_dt = dt_classifier.predict(x_test)
    accuracy_dt = accuracy_score(y_test, y_pred_dt)

    if accuracy_dt > best_accuracy_dt:
        best_accuracy_dt = accuracy_dt
        best_depth_dt = depth

best_depth_dt
```

[122]: 3

[123]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

for depth in range(1, 26):
    dt_classifier = DecisionTreeClassifier(max_depth=depth)
    dt_classifier.fit(x_train, y_train)

    y_pred_train = dt_classifier.predict(x_train)
    y_pred_test = dt_classifier.predict(x_test)

    score_test = accuracy_score(y_pred_test, y_test)
    print('for', depth, 'test accuracy is ', score_test)

    score_train = accuracy_score(y_pred_train, y_train)
    print('train accuracy is ', score_train)
```

```
for 1 test accuracy is  0.7157894736842105
train accuracy is  0.7714285714285715
for 2 test accuracy is  0.6842105263157895
train accuracy is  0.8228571428571428
for 3 test accuracy is  0.7263157894736842
train accuracy is  0.8742857142857143
for 4 test accuracy is  0.6736842105263158
train accuracy is  0.9257142857142857
for 5 test accuracy is  0.6736842105263158
train accuracy is  0.96
for 6 test accuracy is  0.6526315789473685
train accuracy is  0.9828571428571429
for 7 test accuracy is  0.6631578947368421
train accuracy is  0.9942857142857143
```

```
for 8 test accuracy is  0.631578947368421
train accuracy is  0.9942857142857143
for 9 test accuracy is  0.6526315789473685
train accuracy is  1.0
for 10 test accuracy is  0.6421052631578947
train accuracy is  1.0
for 11 test accuracy is  0.6421052631578947
train accuracy is  1.0
for 12 test accuracy is  0.6736842105263158
train accuracy is  1.0
for 13 test accuracy is  0.6526315789473685
train accuracy is  1.0
for 14 test accuracy is  0.6421052631578947
train accuracy is  1.0
for 15 test accuracy is  0.6421052631578947
train accuracy is  1.0
for 16 test accuracy is  0.6736842105263158
train accuracy is  1.0
for 17 test accuracy is  0.6526315789473685
train accuracy is  1.0
for 18 test accuracy is  0.6421052631578947
train accuracy is  1.0
for 19 test accuracy is  0.6526315789473685
train accuracy is  1.0
for 20 test accuracy is  0.6421052631578947
train accuracy is  1.0
for 21 test accuracy is  0.6526315789473685
train accuracy is  1.0
for 22 test accuracy is  0.6526315789473685
train accuracy is  1.0
for 23 test accuracy is  0.6631578947368421
train accuracy is  1.0
for 24 test accuracy is  0.6421052631578947
train accuracy is  1.0
for 25 test accuracy is  0.6631578947368421
train accuracy is  1.0
```

[124]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier


parameters = {'max_depth': [3, 5, 10, 15, 20], 'min_samples_split': [2, 5, 10],␣
 ↪'max_leaf_nodes': [15,20,25,30]}
dt_classifier = DecisionTreeClassifier()

grid_search = GridSearchCV(dt_classifier, parameters, cv=5)
grid_search.fit(x_train, y_train)
```

```
print("Best Hyperparameters:", grid_search.best_params_)


best_dt_model = grid_search.best_estimator_
y_pred_test = best_dt_model.predict(x_test)
```

```
Best Hyperparameters: {'max_depth': 3, 'max_leaf_nodes': 15,
'min_samples_split': 5}
```

[125]:
```
prune=DecisionTreeClassifier(max_depth=3, max_leaf_nodes=25, random_state=101)
prune.fit(x_train,y_train)
```

[125]:
```
DecisionTreeClassifier(max_depth=3, max_leaf_nodes=25, random_state=101)
```

[126]:
```
y_prune_pred=prune.predict(x_test)
```

[128]:
```
score=accuracy_score(y_train,prune.predict(x_train))
print('train accuracy score is : ',score)
```

```
train accuracy score is :  0.8742857142857143
```

[129]:
```
score=accuracy_score(y_test,y_prune_pred)
print('test accuracy score is : ',score)
```

```
test accuracy score is :  0.7263157894736842
```

2.0.3 Naive Bayes

[132]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report
```

[133]:
```
x, y = make_classification(n_samples=1000, n_features=20, n_classes=2,␣
↪random_state=42)
```

[134]:
```
X_train,X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
↪random_state=42)
```

[135]:
```
naive_bayes_classifier = GaussianNB()
naive_bayes_classifier.fit(X_train, y_train)
```

[135]:
```
GaussianNB()
```

[136]:
```
y_pred = naive_bayes_classifier.predict(X_test)
```

[137]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
confusion_mat = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", confusion_mat)
classification_rep = classification_report(y_test, y_pred)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.795
Confusion Matrix:
 [[84  9]
 [32 75]]
Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.90      0.80        93
           1       0.89      0.70      0.79       107

    accuracy                           0.80       200
   macro avg       0.81      0.80      0.79       200
weighted avg       0.81      0.80      0.79       200
```

2.0.4 Logistic Regression

[138]:
```
from sklearn.linear_model import LogisticRegression
```

[145]:
```
X = df.drop('Heart Disease', axis=1)
y = df['Heart Disease']

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
```

[146]:
```
logistic_classifier = LogisticRegression(random_state=42)

logistic_classifier.fit(x_train, y_train)
```

[146]:
```
LogisticRegression(random_state=42)
```

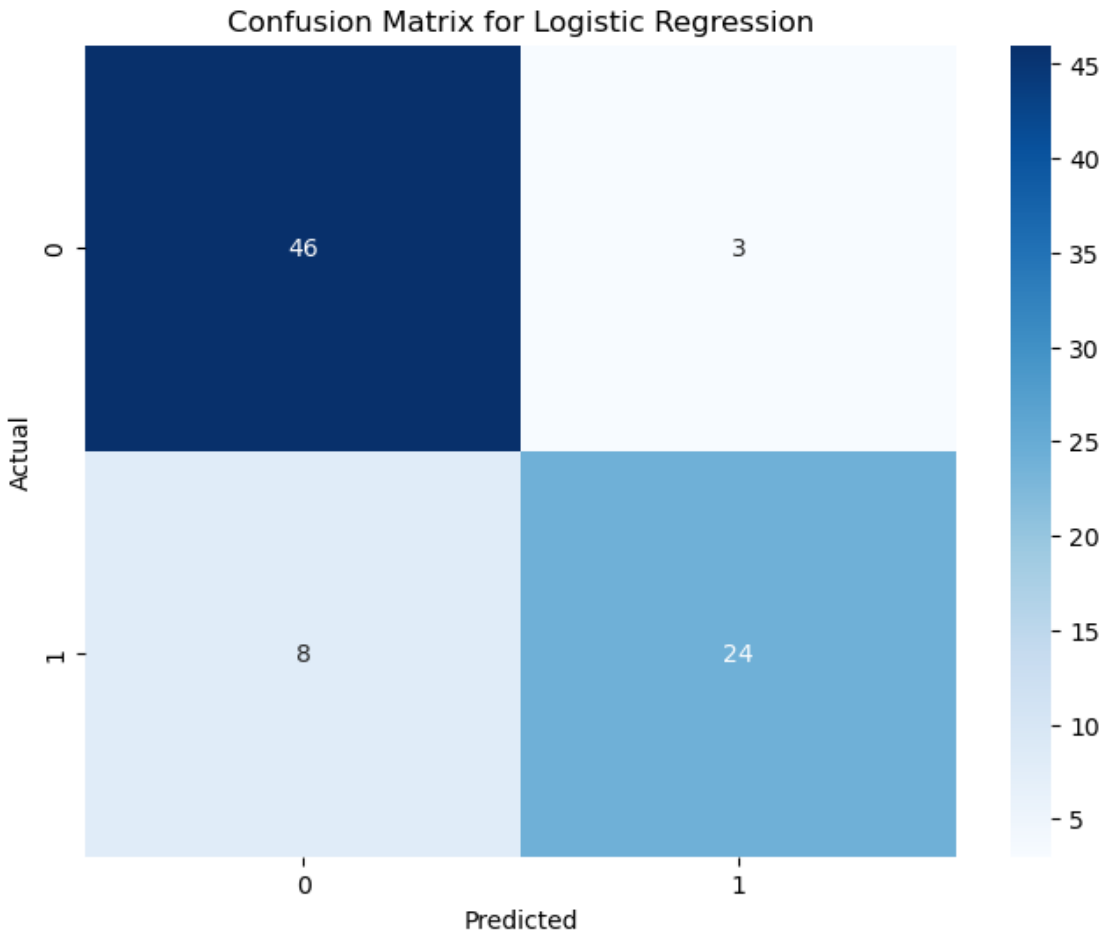
[150]:
```
y_pred = logistic_classifier.predict(x_test)
```

[151]:
```
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

[151]:
```
0.8641975308641975
```

[153]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
```

```
plt.title('Confusion Matrix for Logistic Regression')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



[ ]: