In [1]:
```
from sklearn.datasets import load_breast_cancer
import pandas as pd
data = load_breast_cancer()

x = pd.DataFrame(data.data, columns=data.feature_names)
y=pd.Series(data.target,name='target')
```

In [2]:
```
x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
dtypes: float64(30)
memory usage: 133.5 KB
```

In [3]:
```
y.info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 569 entries, 0 to 568
Series name: target
Non-Null Count  Dtype
--------------  -----
569 non-null    int32
dtypes: int32(1)
memory usage: 2.4 KB
```

In [4]:
```
x.duplicated().sum()
```

Out[4]:
```
0
```

In [5]:
```
x.isnull().sum()
```

Out[5]:
```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
dtype: int64
```

In [6]:
```
x.describe()
```

Out[6]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	m conc po
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201

8 rows × 30 columns

In [7]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

x_train, x_test, y_train, y_test = train_test_split(x, y)
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

In [8]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

In [9]:
```
logreg=LogisticRegression()
logreg.fit(x_train,y_train)
```

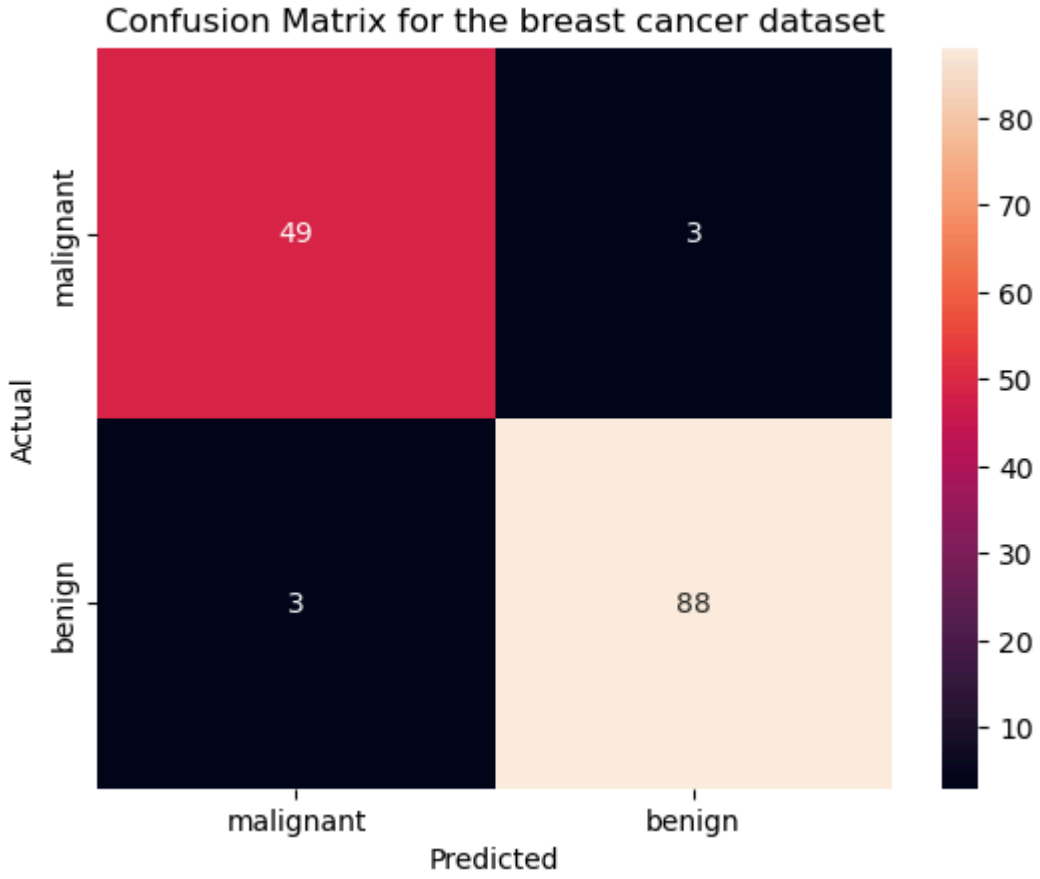
Out[9]:
```
▾ LogisticRegression

LogisticRegression()
```

In [10]:
```
y_pred=logreg.predict(x_test)
y_pred
```

Out[10]:
```
array([0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0,
       1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1])
```

In [11]:
```
import seaborn as sns
con=confusion_matrix(y_test,y_pred)
import matplotlib.pyplot as plt
sns.heatmap(con,annot=True,xticklabels=data.target_names,yticklabels=data.target_nam
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for the breast cancer dataset')
plt.show()
```



In [12]:
```
cm=confusion_matrix(y_test,y_pred)
print('confusion_matrix :',)
print(cm)
```

```
confusion_matrix :
[[49  3]
 [ 3 88]]
```

In [13]:
```
cr=classification_report(y_test,y_pred)
print("Classification Report:")
print(cr)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.94      0.94        52
           1       0.97      0.97      0.97        91

    accuracy                           0.96       143
   macro avg       0.95      0.95      0.95       143
weighted avg       0.96      0.96      0.96       143
```

In [14]:
```
accuracy=accuracy_score(y_test,y_pred)
print("Accuracy Score: ")
print(accuracy)
```

```
Accuracy Score:
0.958041958041958
```

Decision Tree Classifier

Decision Tree Classifier builds a tree structure by recursively splitting the dataset into subsets based on feature values that result in the highest information gain (or lowest impurity). Each internal node represents a feature decision, while leaves represent class labels. It is suitable for this dataset because it can handle non-linear relationships, is easy to interpret, and works well for smaller datasets like the breast cancer data.

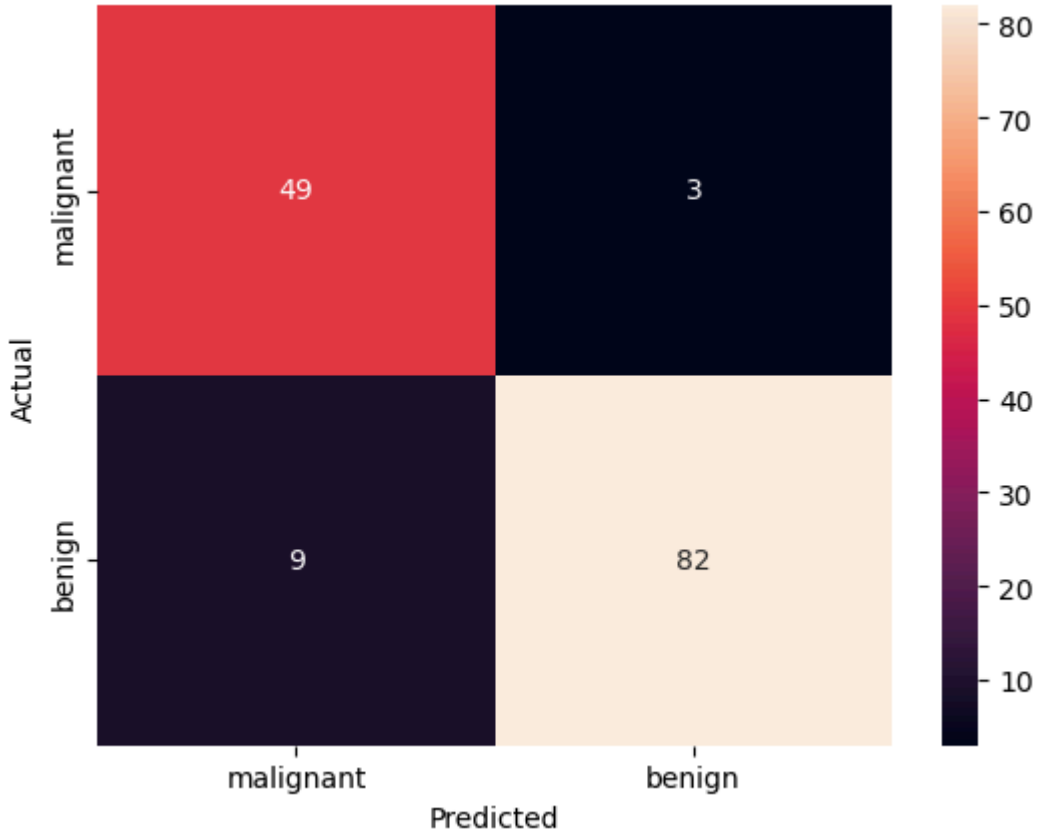
In [15]:
```
from sklearn.tree import DecisionTreeClassifier

tree=DecisionTreeClassifier()
tree.fit(x_train,y_train)
y_pred=tree.predict(x_test)
```

In [16]:
```
con=confusion_matrix(y_test,y_pred)
print('confusion matrix :')
print(con)
```

```
confusion matrix :
[[49  3]
 [ 9 82]]
```

In [17]:
```
sns.heatmap(con,annot=True,xticklabels=data.target_names,yticklabels=data.target_nam
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



In [18]:
```
cr=classification_report(y_test,y_pred)
print("Classification Report:")
print(cr)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.94      0.89        52
           1       0.96      0.90      0.93        91

    accuracy                           0.92       143
   macro avg       0.90      0.92      0.91       143
weighted avg       0.92      0.92      0.92       143
```

In [19]:
```
accuracy=accuracy_score(y_test,y_pred)
print("Accuracy Score: ")
print(accuracy)
```

```
Accuracy Score:
0.916083916083916
```

Random Forest Classifier

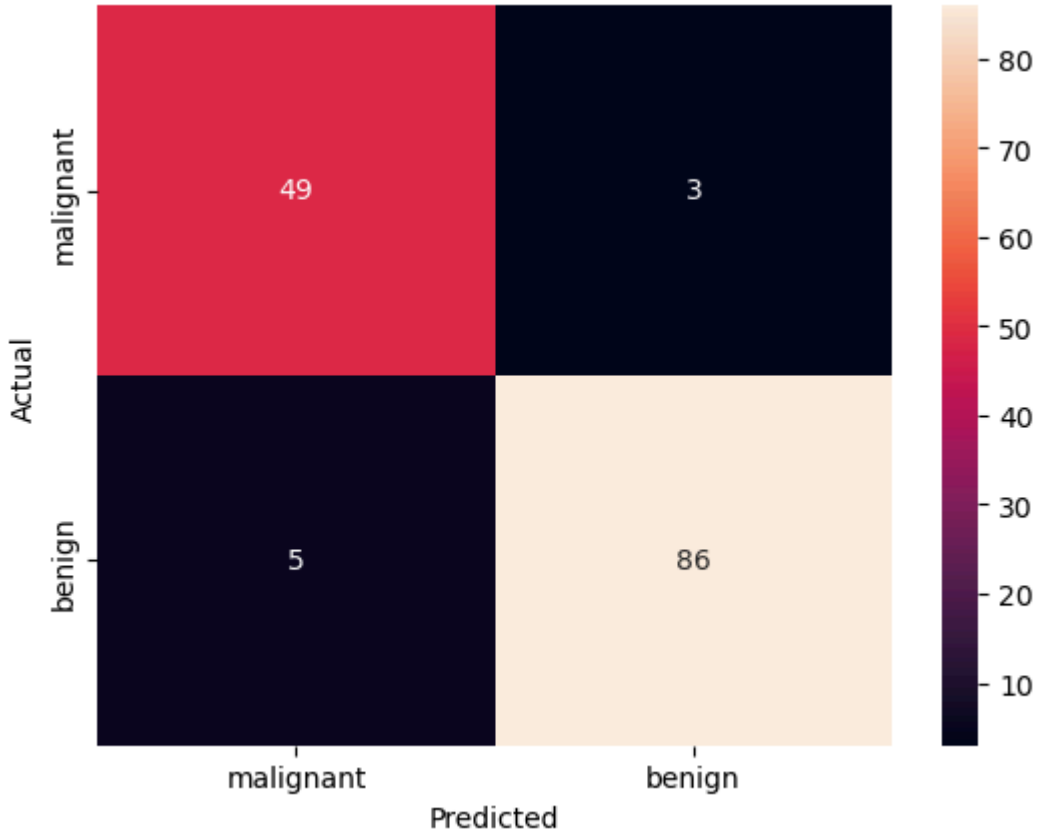
Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the class that is the mode of the predictions from individual trees. It reduces overfitting by averaging predictions, which improves generalization. This model is suitable for the breast cancer dataset because it increases robustness and accuracy compared to individual decision trees, especially in handling noisy data.

In [20]:
```
from sklearn.ensemble import RandomForestClassifier
forest=RandomForestClassifier()
forest.fit(x_train,y_train)
y_pred=forest.predict(x_test)
```

In [21]:
```
con=confusion_matrix(y_test,y_pred)
print('confusion_matrix :',)
print(con)
```

```
confusion_matrix :
[[49  3]
 [ 5 86]]
```

In [22]:
```
sns.heatmap(con,annot=True,xticklabels=data.target_names,yticklabels=data.target_nam
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



In [23]:
```
cr=classification_report(y_test,y_pred)
print("Classification Report:")
print(cr)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.94      0.92        52
           1       0.97      0.95      0.96        91

    accuracy                           0.94       143
   macro avg       0.94      0.94      0.94       143
weighted avg       0.94      0.94      0.94       143
```

In [24]:
```
accuracy=accuracy_score(y_test,y_pred)
print("Accuracy Score: ")
print(accuracy)
```

```
Accuracy Score:
0.9440559440559441
```

Support Vector Machine (SVM)

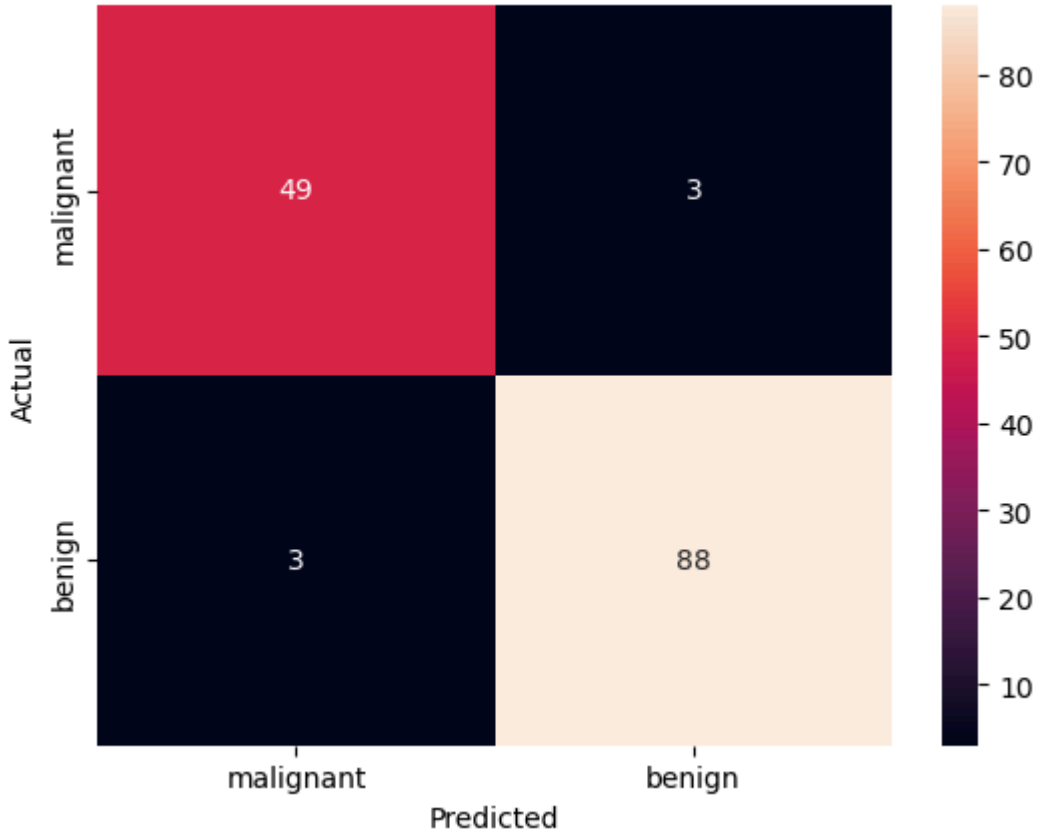
Support Vector Machine (SVM) is a powerful classification algorithm that finds the optimal hyperplane to separate data points of different classes by maximizing the margin between them. It works well for high-dimensional data and can handle non-linear boundaries using kernel

In [25]:
```
from sklearn.svm import SVC
svm=SVC()
svm.fit(x_train,y_train)
y_pred=svm.predict(x_test)
```

In [26]:
```
con=confusion_matrix(y_test,y_pred)
print('confusion_matrix :',)
print(con)
```

```
confusion_matrix :
[[49  3]
 [ 3 88]]
```

In [27]:
```
sns.heatmap(con,annot=True,xticklabels=data.target_names,yticklabels=data.target_nam
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



In [28]:
```
cr=classification_report(y_test,y_pred)
print("Classification Report:")
print(cr)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.94      0.94        52
           1       0.97      0.97      0.97        91

    accuracy                           0.96       143
   macro avg       0.95      0.95      0.95       143
weighted avg       0.96      0.96      0.96       143
```

In [29]:
```
accuracy=accuracy_score(y_test,y_pred)
print("Accuracy Score: ")
print(accuracy)
```

```
Accuracy Score:
0.958041958041958
```

k-Nearest Neighbors (k-NN)

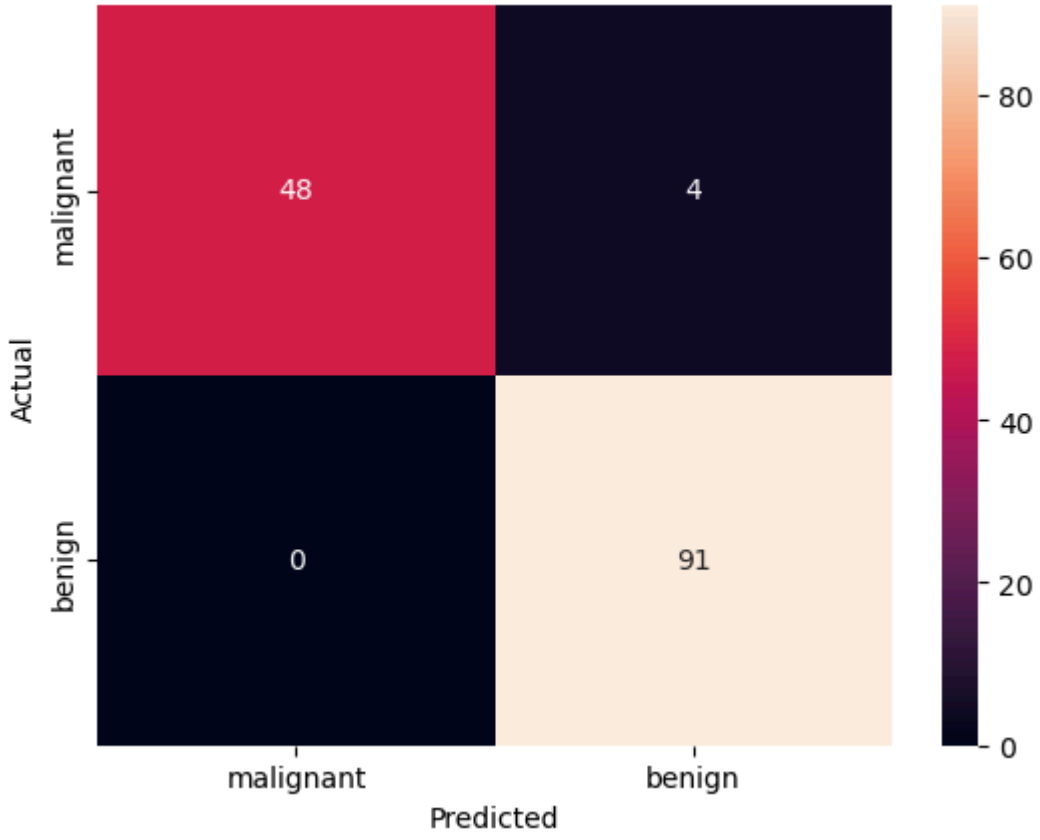
k-NN is a simple, instance-based algorithm that classifies a data point by majority voting among its k-nearest neighbors based on distance metrics (e.g., Euclidean distance). It doesn't assume any underlying data distribution, making it flexible. It is suitable for the breast cancer dataset because it can classify data with complex boundaries and performs well when the dataset is properly scaled, as is required here with many features.

In [30]:
```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()
knn.fit(x_train,y_train)
y_pred=knn.predict(x_test)
```

In [31]:
```
con=confusion_matrix(y_test,y_pred)
print('confusion matrix')
print(con)
```

```
confusion matrix
[[48  4]
 [ 0 91]]
```

In [32]:
```
sns.heatmap(con,annot=True,xticklabels=data.target_names,yticklabels=data.target_nam
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



In [33]:
```
cr=classification_report(y_test,y_pred)
print("Classification Report:")
print(cr)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.92      0.96        52
           1       0.96      1.00      0.98        91

    accuracy                           0.97       143
   macro avg       0.98      0.96      0.97       143
weighted avg       0.97      0.97      0.97       143
```

In [34]:
```
accuracy=accuracy_score(y_test,y_pred)
print("Accuracy Score: ")
print(accuracy)
```

```
Accuracy Score:
0.972027972027972
```

The k-Nearest Neighbors (k-NN) algorithm performed the best with an accuracy of 0.972, demonstrating its effectiveness in classifying the breast cancer dataset. In contrast, the Decision Tree Classifier, with an accuracy of 0.91, was the worst-performing algorithm compared to the other models, likely due to its tendency to overfit the training data.