ML:Breast Cancer

Objective:

The objective of this assessment is to evaluate your understanding and ability to apply supervised learning techniques to a real-world dataset.

1. Loading and Preprocessing ¶

Data Exploration:

In [1]:
```
# importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# Load the breast cancer dataset
from sklearn.datasets import load_breast_cancer
data=load_breast_cancer()
df=pd.concat([pd.DataFrame(data.data,columns=data.feature_names),pd.Series(data
```

In [3]:
```
df
```

Out[3]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mear symmetr
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.241
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.181
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.206
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.259
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.180
...	...	...	...	...	...	...	...	...	..
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.172
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.175
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.159
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.239
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.158

569 rows × 31 columns

In [4]:
```
df.shape
```

Out[4]:
```
(569, 31)
```

Data Cleaning

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 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
 30  target                   569 non-null    int32  
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

In [6]:
```
# checking duplicates
df.duplicated().sum()
```

Out[6]: 0

In [7]:
```
# missing values
df.isnull().sum()
```

Out[7]:
```
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
target                     0
dtype: int64
```

Remove the outliers

In [8]:
```
df.describe()
```

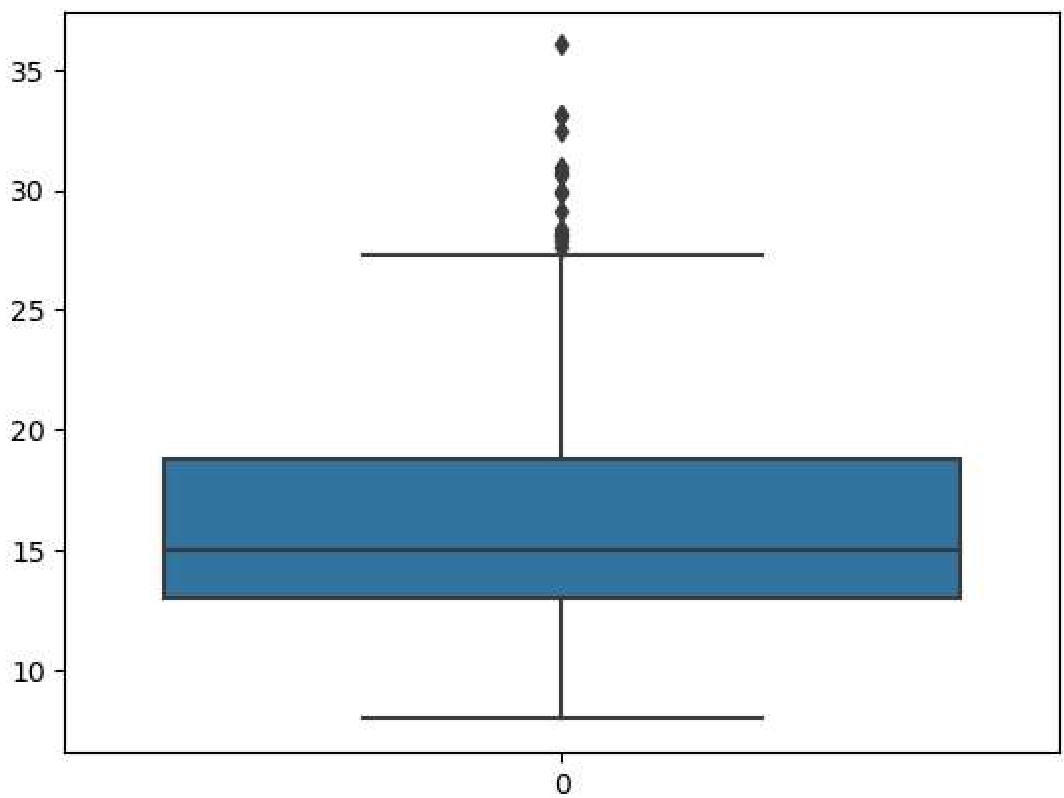
Out[8]:

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

8 rows × 31 columns

In [9]:
```
sns.boxplot(df["worst radius"])
```

Out[9]: <Axes: >



In [10]:
```
df.skew()
```

Out[10]:
```
mean radius                0.942380
mean texture               0.650450
mean perimeter             0.990650
mean area                  1.645732
mean smoothness            0.456324
mean compactness           1.190123
mean concavity             1.401180
mean concave points        1.171180
mean symmetry              0.725609
mean fractal dimension     1.304489
radius error               3.088612
texture error              1.646444
perimeter error            3.443615
area error                 5.447186
smoothness error           2.314450
compactness error          1.902221
concavity error            5.110463
concave points error       1.444678
symmetry error             2.195133
fractal dimension error    3.923969
worst radius               1.103115
worst texture              0.498321
worst perimeter            1.128164
worst area                 1.859373
worst smoothness           0.415426
worst compactness          1.473555
worst concavity            1.150237
worst concave points       0.492616
worst symmetry             1.433928
worst fractal dimension    1.662579
target                    -0.528461
dtype: float64
```

In [11]:
```
# sort the values
df.skew().sort_values()
```

Out[11]:
```
target                    -0.528461
worst smoothness           0.415426
mean smoothness            0.456324
worst concave points       0.492616
worst texture              0.498321
mean texture               0.650450
mean symmetry              0.725609
mean radius                0.942380
mean perimeter             0.990650
worst radius               1.103115
worst perimeter            1.128164
worst concavity            1.150237
mean concave points        1.171180
mean compactness           1.190123
mean fractal dimension     1.304489
mean concavity             1.401180
worst symmetry             1.433928
concave points error       1.444678
worst compactness          1.473555
mean area                  1.645732
texture error              1.646444
worst fractal dimension    1.662579
worst area                 1.859373
compactness error          1.902221
symmetry error             2.195133
smoothness error           2.314450
radius error               3.088612
perimeter error            3.443615
fractal dimension error    3.923969
concavity error            5.110463
area error                 5.447186
dtype: float64
```

In [12]:
```
# Remove the outlier all columns by using functions of IQR method
def remove_outliers(df,columns):
    df_filtered = df.copy()

    for col in columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1

        lower_whisker = Q1 - 1.5 * IQR
        upper_whisker = Q3 + 1.5 * IQR

        df_filtered = df_filtered[(df_filtered[col] <= upper_whisker) & (df_fil
    return df_filtered
```

In [13]:
```
dff=remove_outliers(df,["worst radius","worst perimeter","worst concavity","mea
                        "mean fractal dimension","mean concavity","worst symmet
                         "mean area","texture error","worst fractal dimension",
                        "smoothness error","radius error","perimeter error","fr
```

In [14]:
```
# After removed the outlier
dff
```

Out[14]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry
6	18.25	19.98	119.60	1040.0	0.09463	0.10900	0.11270	0.07400	0.1794
7	13.71	20.83	90.20	577.9	0.11890	0.16450	0.09366	0.05985	0.2196
10	16.02	23.24	102.70	797.8	0.08206	0.06669	0.03299	0.03323	0.1528
11	15.78	17.89	103.60	781.0	0.09710	0.12920	0.09954	0.06606	0.1842
13	15.85	23.95	103.70	782.7	0.08401	0.10020	0.09938	0.05364	0.1847
...	...	...	...	...	...	...	...	...	..
555	10.29	27.61	65.67	321.4	0.09030	0.07658	0.05999	0.02738	0.1593
558	14.59	22.68	96.39	657.1	0.08473	0.13300	0.10290	0.03736	0.1454
560	14.05	27.15	91.38	600.4	0.09929	0.11260	0.04462	0.04304	0.1537
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587

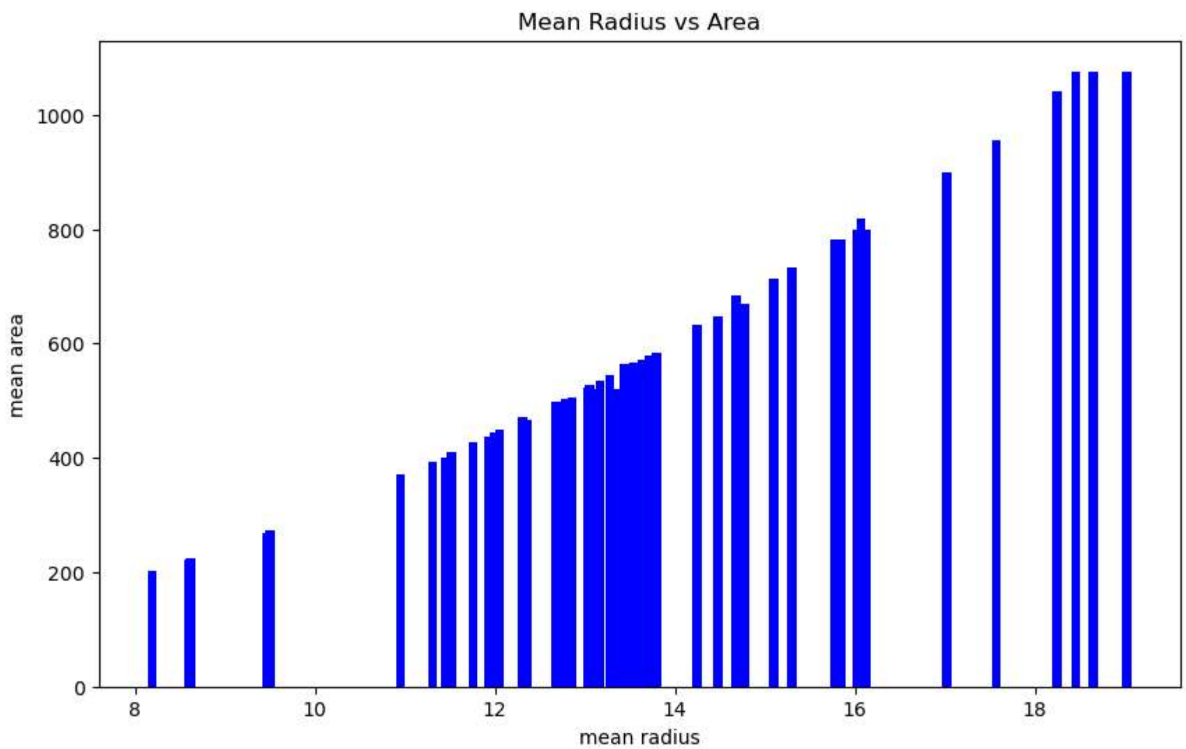
405 rows × 31 columns

In [15]:
```
dff.skew()
```

Out[15]:
```
mean radius                0.697041
mean texture               0.889073
mean perimeter             0.711286
mean area                  1.224444
mean smoothness            0.100058
mean compactness           0.748616
mean concavity             1.348930
mean concave points        1.128692
mean symmetry              0.227017
mean fractal dimension     0.498133
radius error               1.044660
texture error              0.617266
perimeter error            1.057219
area error                 1.561884
smoothness error           0.662784
compactness error          1.013291
concavity error            0.948206
concave points error       0.327594
symmetry error             0.823376
fractal dimension error    1.203323
worst radius               0.900488
worst texture              0.589159
worst perimeter            0.889128
worst area                 1.361273
worst smoothness           0.261996
worst compactness          0.948003
worst concavity            0.887862
worst concave points       0.584965
worst symmetry             0.305589
worst fractal dimension    0.776603
target                    -1.162818
dtype: float64
```

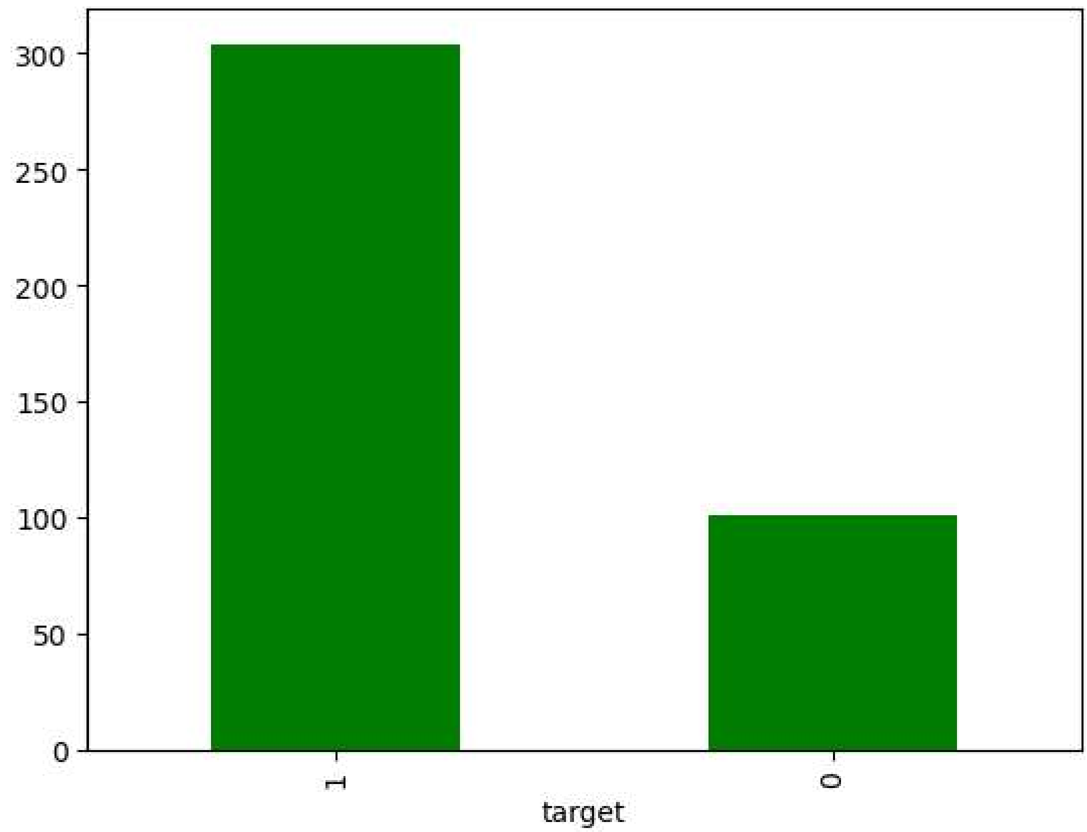
Data Analysis

In [16]:
```
# Plot the chart with age and salary
# data for plotting
x=dff['mean radius'].head(50)
y=dff['mean area'].head(50)
# Plotting the bar chart
plt.figure(figsize=(10, 6))
plt.bar(x,y, color=['blue'],width=0.1)

# Adding labels and title
plt.xlabel('mean radius')
plt.ylabel('mean area')
plt.title('Mean Radius vs Area')

# Adding the values on top of the bars
plt.show()
```



In [17]:
```
# count the no.of people from each place and plot it.
dff["target"].value_counts().plot(kind='bar',color='green')
```

Out[17]: <Axes: xlabel='target'>



In [18]:
```
# Features- x,Target- y
x=dff[['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error', 'fractal dimension error',
       'worst radius', 'worst texture', 'worst perimeter', 'worst area',
       'worst smoothness', 'worst compactness', 'worst concavity',
       'worst concave points', 'worst symmetry', 'worst fractal dimension']]
y=dff['target']
```

In [19]:
```
dff.columns
```

Out[19]:
```
Index(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error', 'fractal dimension error',
       'worst radius', 'worst texture', 'worst perimeter', 'worst area',
       'worst smoothness', 'worst compactness', 'worst concavity',
       'worst concave points', 'worst symmetry', 'worst fractal dimension',
       'target'],
      dtype='object')
```

Data Scaling

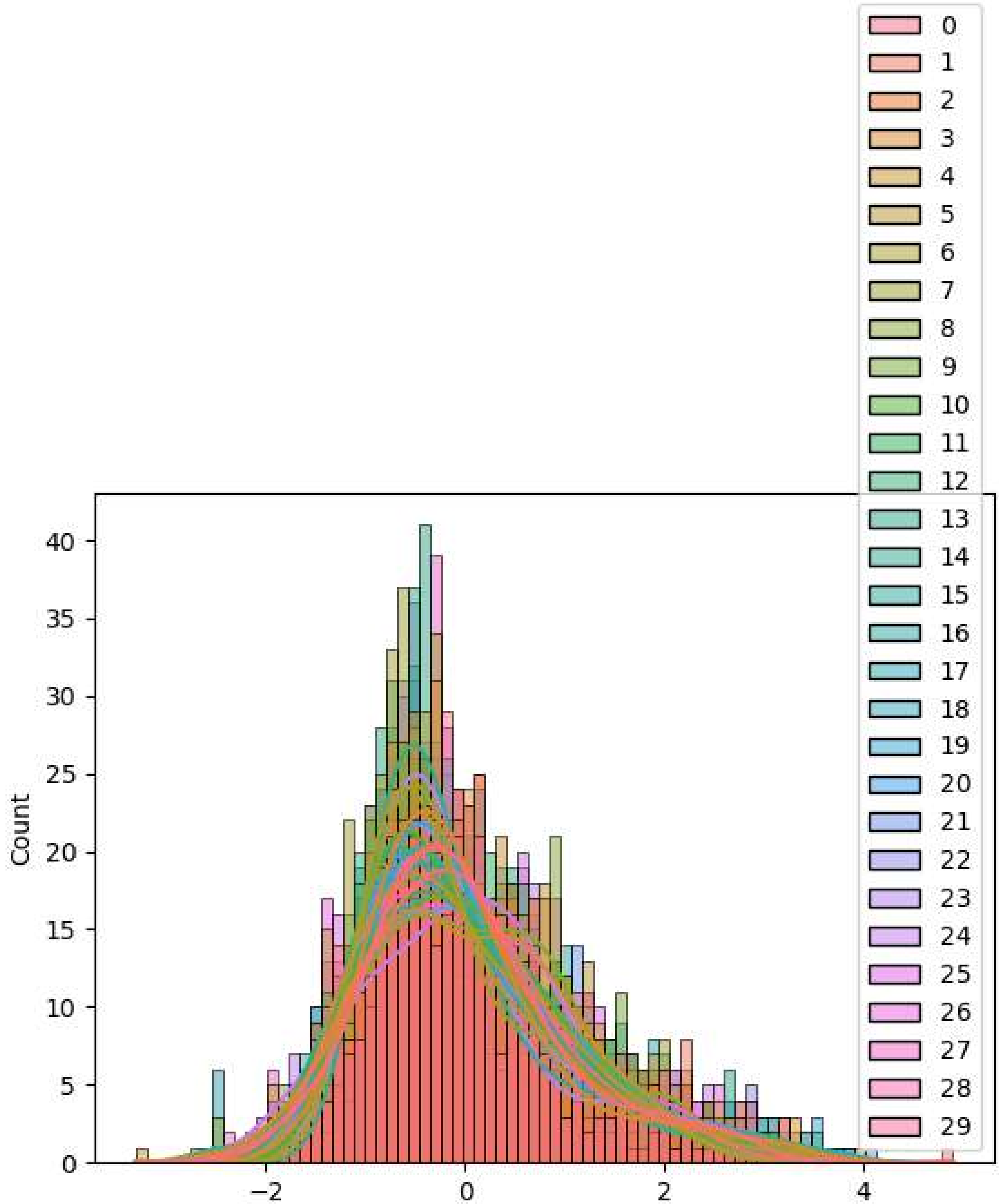
In [20]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

In [21]:
```
# standard scaling the all columns
z = StandardScaler()
dff_standard_scaled = z.fit_transform(x)
dff_standard_scaled
```

Out[21]:
```
array([[ 1.96076691,  0.30704867,  1.96911778, ...,  1.91695713,
         0.62115018,  0.32409483],
       [ 0.13828815,  0.50934046,  0.22873849, ...,  1.17198319,
         0.92747713,  2.82045642],
       [ 1.06558461,  1.08289718,  0.96869567, ...,  0.0654195 ,
         0.35628102,  0.39083397],
       ...,
       [ 0.27477334,  2.0134394 ,  0.29859045, ...,  0.16547584,
        -1.25135966,  0.28675269],
       [ 1.29841229,  2.23477042,  1.30019649, ...,  0.8985619 ,
        -1.32506238, -0.1112986 ],
       [-2.25020275,  1.39228462, -2.27409269, ..., -1.91094088,
         0.17893384, -0.73181368]])
```

In [22]:
```
# after the scaling
sns.histplot(dff_standard_scaled,kde=True)
```

Out[22]: <Axes: ylabel='Count'>



Preprocessing:

I utilized the widely-used breast cancer dataset available in Sklearn.Firstly preprocessing the dataset step by step. Data exploration:We can import required libraries.Then load the dataset Breast_Cancer from Sklearn,concat the features and target of data by using pandas inorder to create a dataframe. Data Cleaning:check the columns and rows,identify missing values and duplicates.then remove the outliers Data Analysing:Analyzing any relationships or patterns that may exist.plot the daigram.By visualizing the data,we can better understand its characteristics and make informed decisions during the modeling process.then encode the catagorical variables. Data Scaling:Scaling the numerical features.Additionally,splitting the dataset into training and testing subsets allows us to evaluate the model's performance on unseen data..All these are minimal for the breast_cancer dataset.

```
   In this particular dataset,it is clean and ready dataset from Sklea
rn that doesn't require much.
   Next,We will look apply some of the best classification algorithms.
```

2. Classification Algorithm Implementation

Implement the following five classification algorithms:

1. Logistic Regression
2. Decision Tree Classifier
3. Random Forest Classifier
4. Support Vector Machine (SVM)
5. k-Nearest Neighbors (k-NN)

1. Logistic Regression

Logistic regression is a statistical method used in machine learning to predict the probability of a binary outcome based on independent variables.it's often used for classification tasks,such as identifying spam or diagnosing diseases

In [23]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix,accuracy_sco
```

In [24]:
```
# 1. Logistic Regression
x_train,x_test,y_train,y_test = train_test_split(dff_standard_scaled,y,test_siz
```

In [25]:
```
x_train.shape
```

Out[25]:
```
(324, 30)
```

In [26]:
```
y_train.shape
```

Out[26]:
```
(324,)
```

In [27]:
```
x_test.shape
```

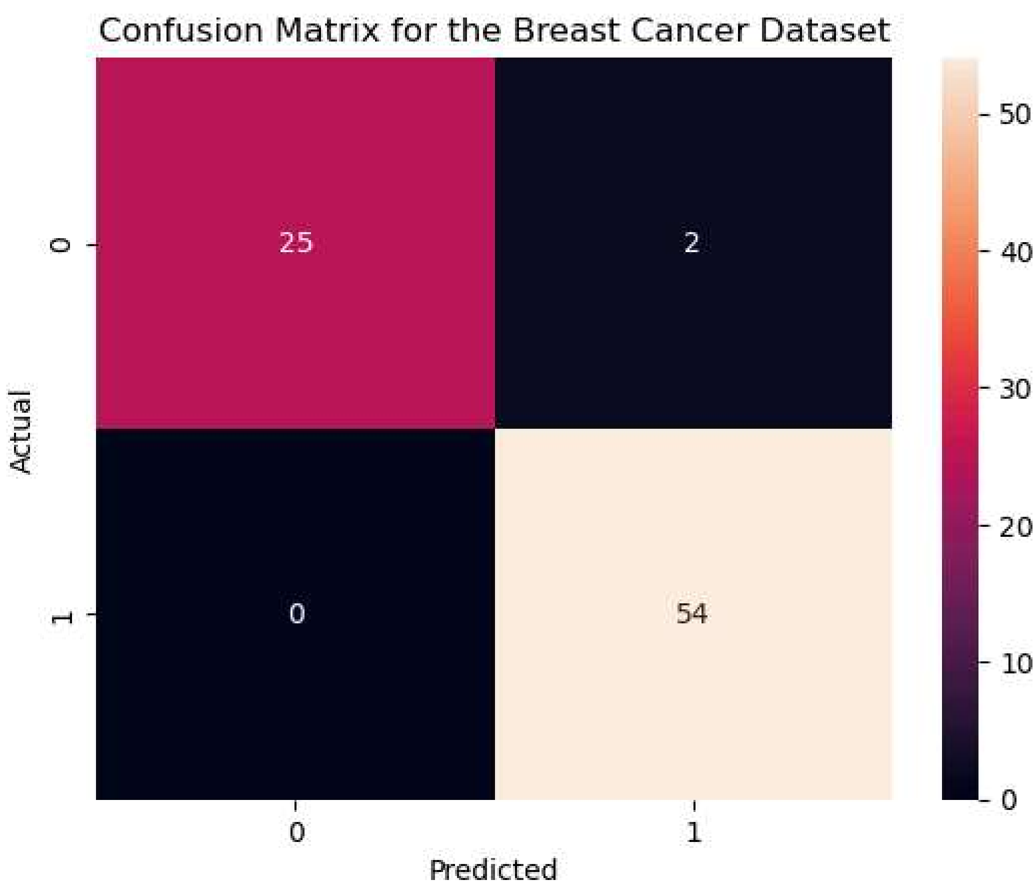
Out[27]:
```
(81, 30)
```

In [28]:
```
y_test.shape
```
Out[28]:
```
(81,)
```

In [29]:
```
model = LogisticRegression()
model.fit(x_train,y_train)
```
Out[29]:
```
▾ LogisticRegression
LogisticRegression()
```

In [30]:
```
y_pred = model.predict(x_test)
y_pred  #PREDICTED
```
Out[30]:
```
array([1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1])
```

In [31]:
```
con = confusion_matrix(y_test,y_pred)
sns.heatmap(con,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for the Breast Cancer Dataset')
plt.show()
```



In [32]:
```
cm=confusion_matrix(y_test,y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[25  2]
 [ 0 54]]
```

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

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

           0       1.00      0.93      0.96        27
           1       0.96      1.00      0.98        54

    accuracy                           0.98        81
   macro avg       0.98      0.96      0.97        81
weighted avg       0.98      0.98      0.98        81
```

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

```
Accuracy Score:
0.9753086419753086
```

In [35]:
```
y_pred #predicted
```

Out[35]:
```
array([1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1])
```

In [36]:
```
y_test #actual
```

Out[36]:
```
115    1
317    0
536    0
61     1
79     1
      ..
84     1
142    1
359    1
566    0
522    1
Name: target, Length: 81, dtype: int32
```

2.SVM(SUPPORT VECTOR MACHINE)

The concept is to find a hyperplane that separates the points into different categories.The points in space represent training data.The points from one class should be separated from another class by the broadest possible distance.This distance is called margin.

In [37]:
```
from sklearn.svm import SVC
```

In [38]:
```
svm_model = SVC()
svm_model.fit(x_train,y_train)
```

Out[38]:
```
▾ SVC
SVC()
```

In [39]:
```
y_pred=svm_model.predict(x_test)
y_pred
```

Out[39]:
```
array([1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1])
```

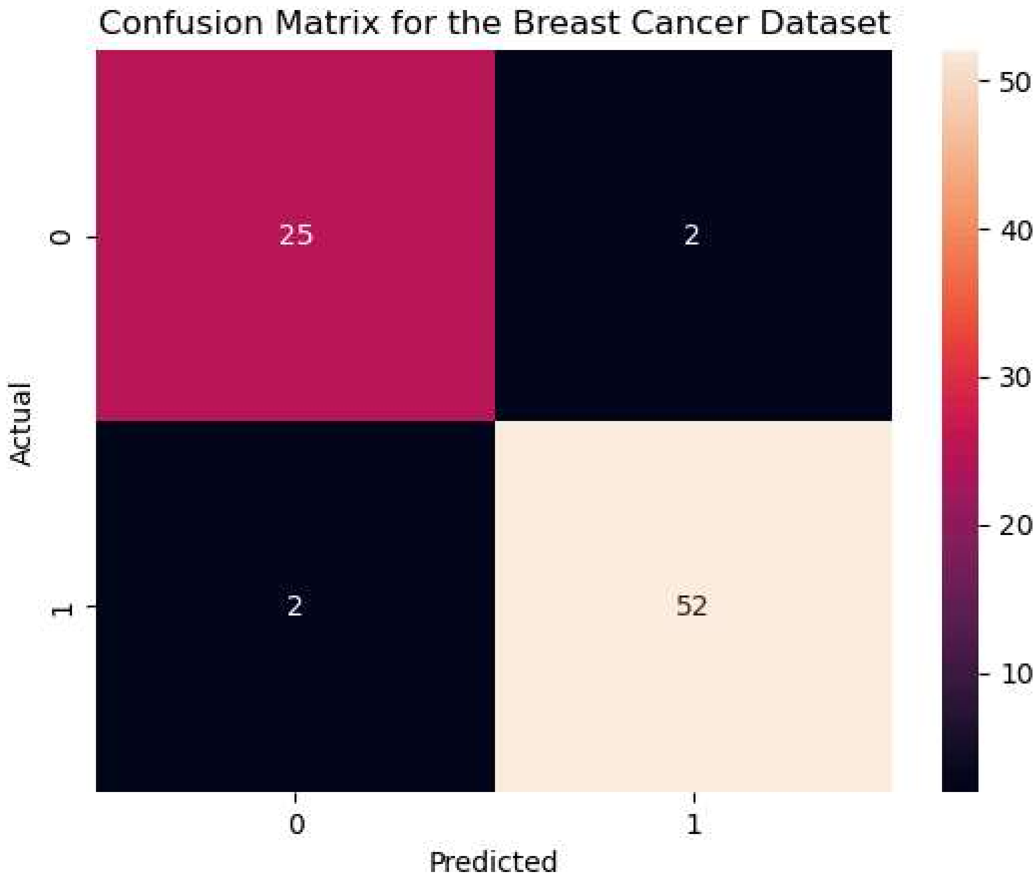
In [40]:
```
y_test
```

Out[40]:
```
115    1
317    0
536    0
61     1
79     1
      ..
84     1
142    1
359    1
566    0
522    1
Name: target, Length: 81, dtype: int32
```

In [41]:
```
print(confusion_matrix(y_test,y_pred))
```

```
[[25  2]
 [ 2 52]]
```

In [42]:
```
con=confusion_matrix(y_test, y_pred)
sns.heatmap(con, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for the Breast Cancer Dataset')
plt.show()
```



In [43]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        27
           1       0.96      0.96      0.96        54

    accuracy                           0.95        81
   macro avg       0.94      0.94      0.94        81
weighted avg       0.95      0.95      0.95        81
```

In [44]:
```
accuracy_score(y_test, y_pred)
```

Out[44]: 0.9506172839506173

3.Decision Tree Classifier

This model builds a form of a tree structure and creates a sequence of rules.Step by step,the original dataset is split into smaller and smaller subsets by these rules.So,an associated decision tree appears with decision nodes and leaf nodes.

In [45]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [46]:
```
dt_model = DecisionTreeClassifier()
dt_model.fit(x_train, y_train)
```

Out[46]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [47]:
```
y_pred = dt_model.predict(x_test)
y_pred
```

Out[47]:
```
array([1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1])
```

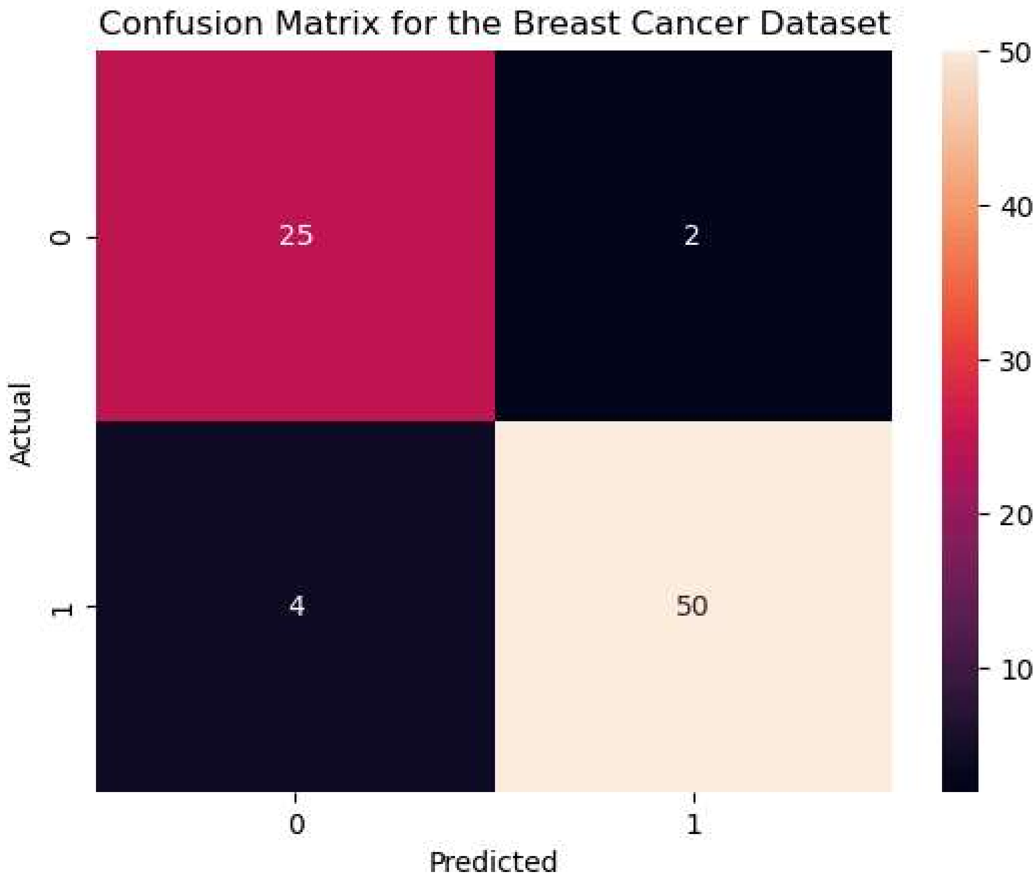
In [48]:
```
y_test
```

Out[48]:
```
115    1
317    0
536    0
61     1
79     1
      ..
84     1
142    1
359    1
566    0
522    1
Name: target, Length: 81, dtype: int32
```

In [49]:
```
print(confusion_matrix(y_test, y_pred))
```

```
[[25  2]
 [ 4 50]]
```

In [50]:
```
con=confusion_matrix(y_test, y_pred)

sns.heatmap(con, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for the Breast Cancer Dataset')
plt.show()
```



In [51]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.93      0.89        27
           1       0.96      0.93      0.94        54

    accuracy                           0.93        81
   macro avg       0.91      0.93      0.92        81
weighted avg       0.93      0.93      0.93        81
```

In [52]:
```
accuracy_score(y_test, y_pred)
```

Out[52]: 0.9259259259259259

4.Random Forest Classifier

Random forest combines the output of multiple decision trees to reach a single result.the random forest takes the prediction from each tree and based on the majority votes of predictions,and it predicts the final output

In [53]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [54]:
```
clf = RandomForestClassifier()
# Train the model
clf.fit(x_train, y_train)
```

Out[54]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [55]:
```

# Make predictions
y_pred = clf.predict(x_test)

y_pred
```

Out[55]:
```
array([1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1])
```

In [56]:
```
y_test #Actual
```

Out[56]:
```
115    1
317    0
536    0
61     1
79     1
      ..
84     1
142    1
359    1
566    0
522    1
Name: target, Length: 81, dtype: int32
```

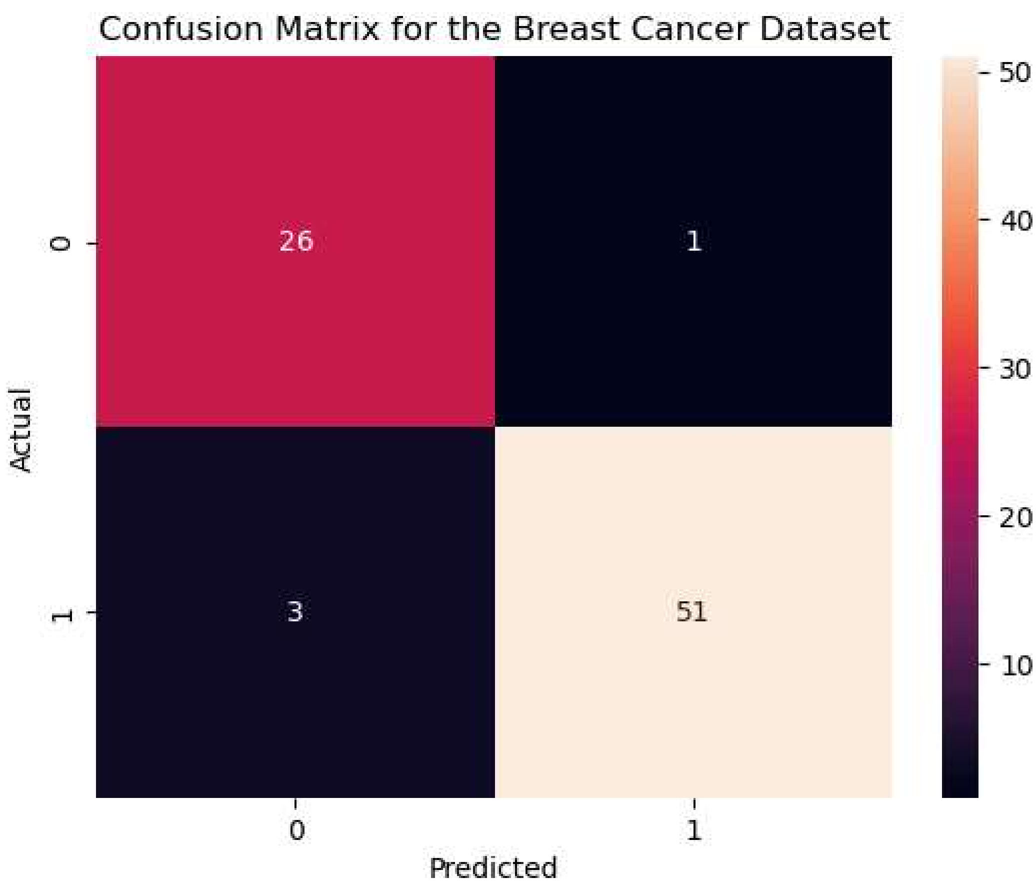
In [57]:
```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.9506172839506173
```

In [58]:
```
report = classification_report(y_test, y_pred)
print("Classification Report:\n",report)
```

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

           0       0.90      0.96      0.93        27
           1       0.98      0.94      0.96        54

    accuracy                           0.95        81
   macro avg       0.94      0.95      0.95        81
weighted avg       0.95      0.95      0.95        81
```

In [59]:
```
con=confusion_matrix(y_test, y_pred)

sns.heatmap(con, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for the Breast Cancer Dataset')
plt.show()
```



In [60]:
```
# Evaluate
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.95
```

5. k-Nearest Neighbors (k-NN)

The K-NN algorithm works by selecting the number of nearest neighbors (k)for each data point ,calculating the distance between the data points and their neighbors,and assigning the class of the majority of the K-nearest neighbors to the new point.The distance measure used can be Euclidean,Manhattan,or Minkowski diatance.

In [61]:
```
from sklearn.neighbors import KNeighborsClassifier
# Create a KNN classifier
knn = KNeighborsClassifier(n_neighbors=11)
# Fit the model
knn.fit(x_train, y_train)
```

Out[61]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=11)
```

In [62]:
```

y_pred = knn.predict(x_test)
y_pred
```

Out[62]:
```
array([1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1])
```

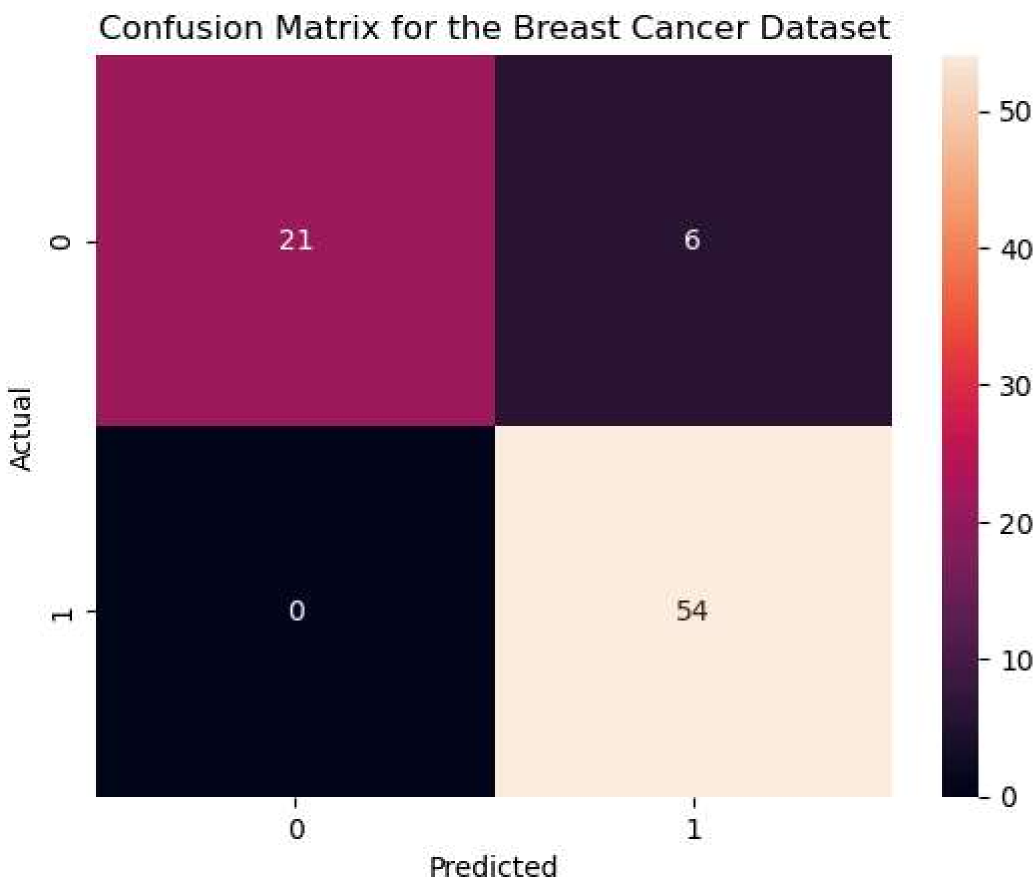
In [63]:
```
# y_test
y_test
```

Out[63]:
```
115    1
317    0
536    0
61     1
79     1
      ..
84     1
142    1
359    1
566    0
522    1
Name: target, Length: 81, dtype: int32
```

In [64]:
```
print(confusion_matrix(y_test, y_pred))
```

```
[[21  6]
 [ 0 54]]
```

In [65]:
```
con=confusion_matrix(y_test, y_pred)

sns.heatmap(con, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for the Breast Cancer Dataset')
plt.show()
```



In [ ]:

In [66]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.78      0.88        27
           1       0.90      1.00      0.95        54

    accuracy                           0.93        81
   macro avg       0.95      0.89      0.91        81
weighted avg       0.93      0.93      0.92        81
```

In [67]:
```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:",accuracy)
```

```
Accuracy: 0.9259259259259259
```

3. Model Comparison (2 marks)

Accuracy of 5 algorithms:

1.Logistic Regression - 97% 2.Support Vector Classification - 95% 3.Decision Tree Classification - 95% 4.Random Forest Classification - 94% 5.kKN - 92%

Logistic regression is a pretty simple algorithm with high efficiency.it calculates the probabilities of an observation assigned toa class. The algorithm works only for machine learning.So that,logistic regression has predicted very accurately 97% in this dataset.The class "1" values are predicted completly.Both training and prediction speed is very fast.

SVC algorithm is efficient in high dimensional spaces.SVC has given a trouble free accuracy(95%).because it is a small dataset.SVC is not suitable for large datasets.Noise in the dataset degrades the result.It works well when there is a clear margin between the categories.

Decision Trees are high varience models,Some changes in the data can dramatically change the predictions produced by the model.Decision tree has given accuracy as same as SVC .It is 95%.It is a small dataset.so,it provide agood accuracy.In the case of large datasets,Decision trees are prone to overfitting and can be biased towards dominant classes.Decision trees may not be the best choice for large datasets or high-dimensional data.so,Decision trees may not always provide the best classification accuracy.

Random Forest Classification is a more accurate algorithm than Decision Tree and is much less subject to overfitting.It is more suitable for large datasets. In this case,accuracy(93%) is lower than others.Both class "1" and "0" predicted equally.Random forest algorithm slows down a real-time prediction.Also,the model can be complex to implement due to many hyperparameters selection.It consist of many trees,so it requires more computing power and more training time.Random Forest not suited for small dataset.

kNN is simple to implement and robust to the noisy training data.It is a non-parametric algorithm.Because of small dataset,We can get accuracy 92% from this dataset.It can be more effective if the training data is large.

All algoithms classified most of the observations correctly ineach class.However,they all had different results,more accurately predicting class "0"or "1" as we have seen in confusion matrices.It influenced the value of the F1 metric for each classifier.

So finally we have built our classification model and we can see that Logistic Regression algorithm gives the best results for our dataset.

We got K-NN algorithm is bad or not well.It is a lazy learner algorithm because it doesn't learn from the trainig set immediately instead it stores the dataset and at the time of classification,it performs an action on the dataset.Always needs to determine the value of K which may be complex some time.The computation cost is high because of calculating the distance between

In [ ]: