In [1]:
```
from sklearn.datasets import load_breast_cancer
import pandas as pd
import numpy as np
```

In [2]:
```
data=load_breast_cancer()
df=pd.DataFrame(data.data,columns=data.feature_names)
```

In [4]:
```
df.head()
```

Out[4]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mea fract dimensio
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.0787
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.0566
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.0599
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.0974
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.0588

5 rows × 30 columns

In [5]:
```
1 df.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 [6]:
```
1 df.describe()
```

Out[6]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mea concav poin
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.0000(
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.0489
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.0388(
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.0000(
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.0203
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.0335(
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.0740(
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.2012(

8 rows × 30 columns

In [7]:
```
1 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
dtype: int64
```

Checked that if there were any missing values. Missing values can cause errors during model training so its crucial to handle missing values.In this data there is no missing values.It's always crucial to ensure the data is reliable and complete

In [8]:
```
1 df.duplicated().sum()
```

Out[8]:
```
0
```

Ensured that there were no duplicates,because duplicates can lead to a biased result

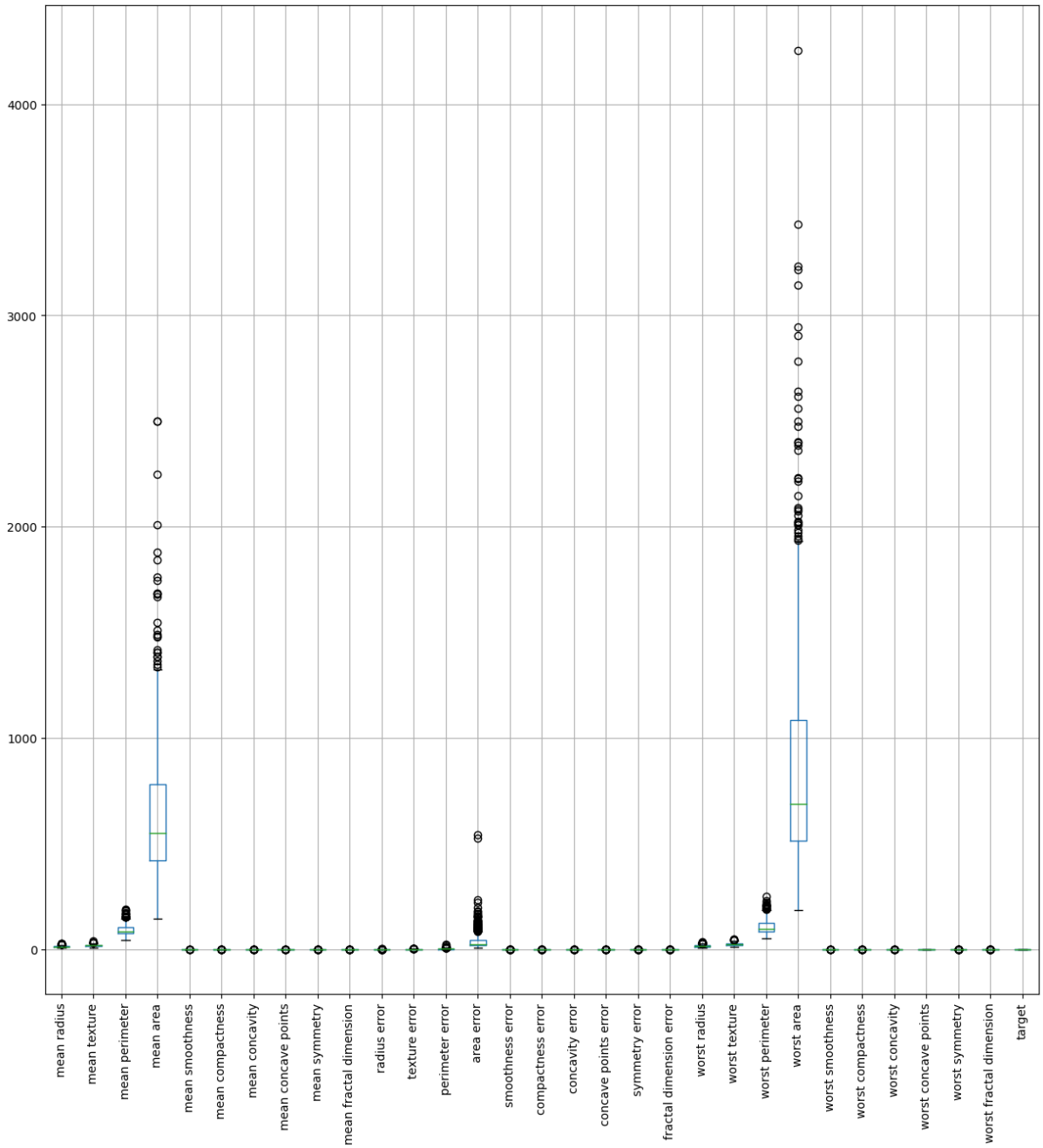
In [10]:
```
1 df['target']=data.target
```

In [11]:
```
1 df['target'].unique()
```

Out[11]:
```
array([0, 1])
```

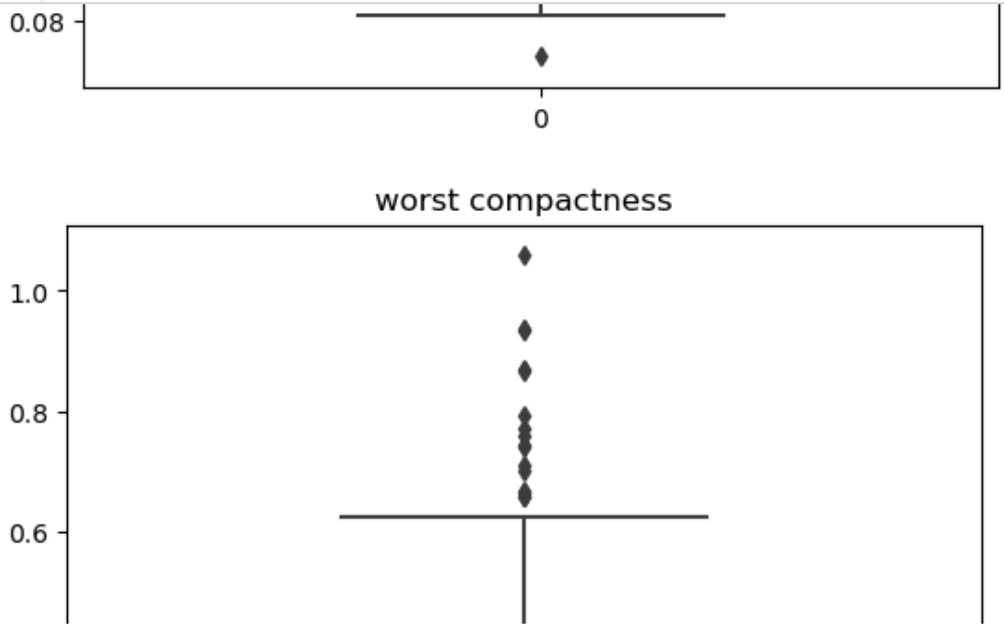
In [12]:
```
1 import seaborn as sns
2 import matplotlib.pyplot as plt
```

In [13]:
```
df.boxplot(figsize=(15,15))
plt.xticks(rotation=90)
plt.show()
```



In [15]:
```
columns=df.columns
```

In [21]:
```
for col in columns:
    sns.boxplot(df[col])
    plt.title(col)
    plt.show()
```



In [22]:

```
df.corr()
```

Out[22]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points
mean radius	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.676764	0.822529
mean texture	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.302418	0.293464
mean perimeter	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.716136	0.850977
mean area	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.685983	0.823269
mean smoothness	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.521984	0.553695
mean compactness	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.883121	0.831135
mean concavity	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	1.000000	0.921391
mean concave points	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.921391	1.000000
mean symmetry	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.500667	0.462497
mean fractal dimension	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	0.336783	0.166917
radius error	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	0.631925	0.698050
texture error	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.046205	0.076218	0.021480
perimeter error	0.674172	0.281673	0.693135	0.726628	0.296092	0.548905	0.660391	0.710650
area error	0.735864	0.259845	0.744983	0.800086	0.246552	0.455653	0.617427	0.690299
smoothness error	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.135299	0.098564	0.027653
compactness error	0.206000	0.191975	0.250744	0.212583	0.318943	0.738722	0.670279	0.490424
concavity error	0.194204	0.143293	0.228082	0.207660	0.248396	0.570517	0.691270	0.439167
concave points error	0.376169	0.163851	0.407217	0.372320	0.380676	0.642262	0.683260	0.615634
symmetry error	-0.104321	0.009127	-0.081629	-0.072497	0.200774	0.229977	0.178009	0.095351
fractal dimension error	-0.042641	0.054458	-0.005523	-0.019887	0.283607	0.507318	0.449301	0.257584
worst radius	0.969539	0.352573	0.969476	0.962746	0.213120	0.535315	0.688236	0.830318
worst texture	0.297008	0.912045	0.303038	0.287489	0.036072	0.248133	0.299879	0.292752
worst perimeter	0.965137	0.358040	0.970387	0.959120	0.238853	0.590210	0.729565	0.855923
worst area	0.941082	0.343546	0.941550	0.959213	0.206718	0.509604	0.675987	0.809630
worst smoothness	0.119616	0.077503	0.150549	0.123523	0.805324	0.565541	0.448822	0.452753
worst compactness	0.413463	0.277830	0.455774	0.390410	0.472468	0.865809	0.754968	0.667454
worst concavity	0.526911	0.301025	0.563879	0.512606	0.434926	0.816275	0.884103	0.752399
worst concave points	0.744214	0.295316	0.771241	0.722017	0.503053	0.815573	0.861323	0.910155

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points
worst symmetry	0.163953	0.105008	0.189115	0.143570	0.394309	0.510223	0.409464	0.375744
worst fractal dimension	0.007066	0.119205	0.051019	0.003738	0.499316	0.687382	0.514930	0.368661
target	-0.730029	-0.415185	-0.742636	-0.708984	-0.358560	-0.596534	-0.696360	-0.776614

31 rows × 31 columns

In [23]:
```
df.skew()
```

Out[23]:
```
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 [24]:
```
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_bound=q1-1.5*iqr
    upper_bound=q3+1.5*iqr

    df_filtered=df_filtered[(df_filtered[col]>=lower_bound) & (df_filtered[col]<=
    return df_filtered
```

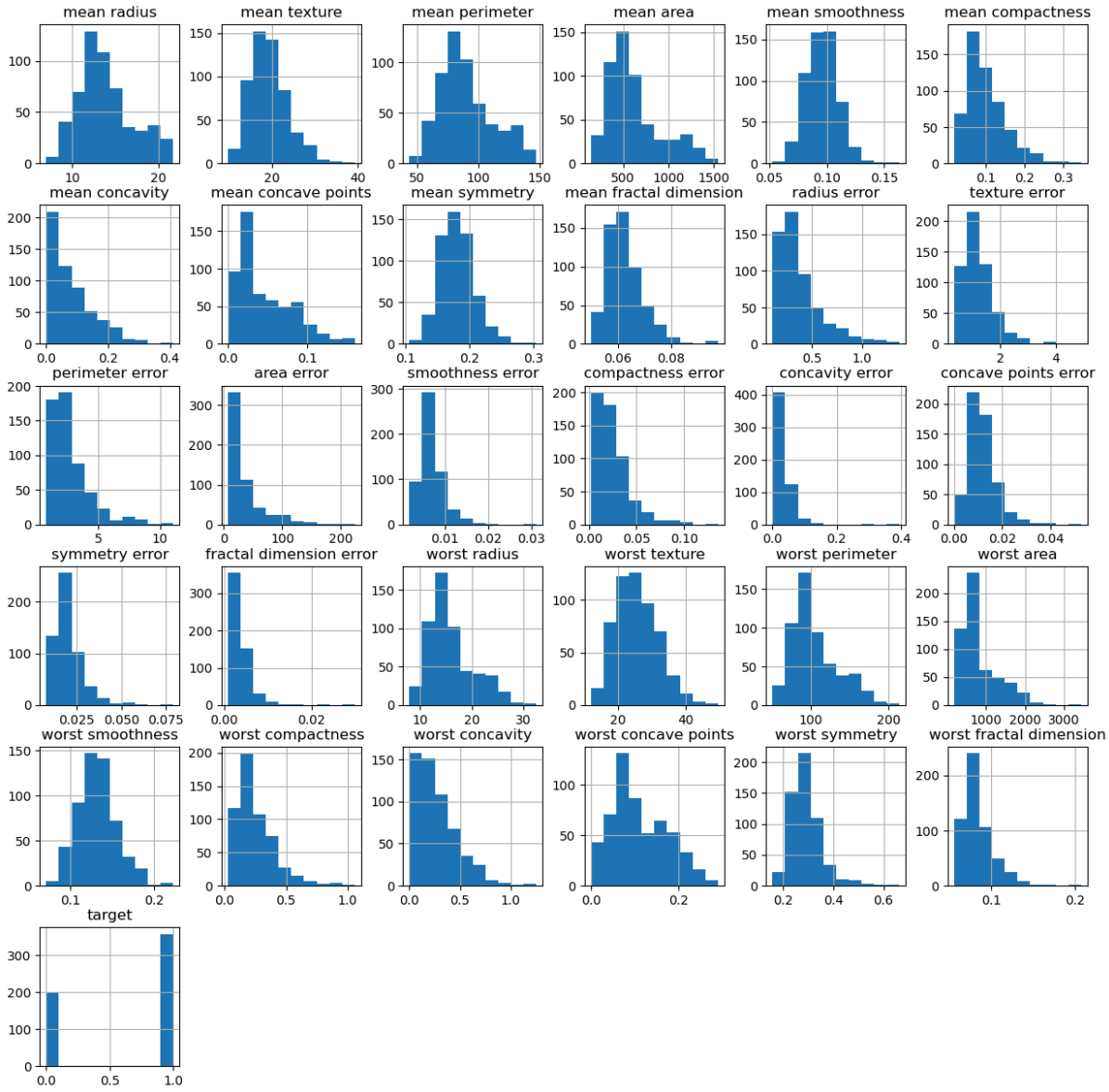
In [25]:
```
dff=remove_outliers(df,df.columns)
```

In [26]:
```
dff.skew()
```

Out[26]:
```
mean radius                0.605931
mean texture               0.684005
mean perimeter             0.634737
mean area                  1.020780
mean smoothness            0.433815
mean compactness           1.207204
mean concavity             1.269207
mean concave points        0.953652
mean symmetry              0.708584
mean fractal dimension     1.309354
radius error               1.663156
texture error              1.637027
perimeter error            1.911531
area error                 2.220182
smoothness error           2.235035
compactness error          1.928395
concavity error            5.351769
concave points error       1.520346
symmetry error             2.223645
fractal dimension error    3.946392
worst radius               0.909712
worst texture              0.522085
worst perimeter            0.886711
worst area                 1.538079
worst smoothness           0.437171
worst compactness          1.549871
worst concavity            1.210024
worst concave points       0.457747
worst symmetry             1.472409
worst fractal dimension    1.674308
target                    -0.599662
dtype: float64
```

Checked for outliers,and created a function to remove outliers. Outliers are extreme values that can distort model predictions, especially for algorithms sensitive to numerical distances.

In [27]:
```
dff.hist(figsize=(15,15))
plt.show()
```



Plotted histogram that gives a visual summary of how data is distributed

Finally,I ensured that the data is cleaned,free from other issues and ready for relaible machine learning model.

In [28]:
```
x=dff.drop('target',axis=1)
y=dff['target']
```

In [29]:
```

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [30]:
```
scaler=StandardScaler()
x=scaler.fit_transform(x)
x
```

Out[30]:
```
array([[ 1.31568926, -2.0551815 ,  1.51863226, ...,  2.45639674,
         2.75031353,  1.9240639 ],
       [ 2.13863766, -0.3359071 ,  1.98816286, ...,  1.19107395,
        -0.24023226,  0.27974976],
       [ 1.85794208,  0.47371062,  1.85334714, ...,  2.0994291 ,
         1.15406327,  0.2005057 ],
       ...,
       [ 0.87231783,  2.06270172,  0.84455367, ...,  0.48670031,
        -1.09975163, -0.31568127],
       [ 2.14820682,  2.35351268,  2.32287775, ...,  2.45002232,
         1.91987564,  2.20471992],
       [-1.94739684,  1.23912507, -1.96240255, ..., -1.77303233,
        -0.04474007, -0.7454702 ]])
```

In [31]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2, random_state=42
```

In [32]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(444, 30)
(111, 30)
(444,)
(111,)
```

LOGISTIC REGRESSION

In [33]:
```
from sklearn.linear_model import LogisticRegression
```

In [34]:
```
model=LogisticRegression()
model.fit(x_train,y_train)
```

Out[34]:
```
LogisticRegression
LogisticRegression()
```

In [35]:
```
y_pred= model.predict(x_test)
y_pred
```

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

In [36]:
```
results_df = pd.DataFrame({
    'Actual Values': y_test,
    'Predicted Values': y_pred
})

print(results_df)

```

```
     Actual Values  Predicted Values
239              0                 0
385              0                 0
55               1                 1
392              0                 0
70               0                 0
..             ...               ...
11               0                 0
289              1                 1
22               0                 0
386              1                 1
489              0                 0

[111 rows x 2 columns]
```

In [37]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score, ac

```

In [38]:
```
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'MAE: {mae}')
print(f'MSE: {mse}')
print(f'R2: {r2}')


```

```
MAE: 0.036036036036036036
MSE: 0.036036036036036036
R2: 0.8515050167224081
```

In [39]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Score:")
print(accuracy)
```

```
Accuracy Score:
0.963963963963964
```

Logistic regression is a classification algorithm used to predict binary outcomes,The breast cancer dataset involves classifying tumors into two categories,making logistic regression an appropriate choice.

Support Vector Machine (SVM)

In [40]:
```
from sklearn.svm import SVC
```

In [41]:
```
svm_model=SVC()
svm_model.fit(x_train,y_train)
```

Out[41]:
```
SVC
SVC()
```

In [42]:
```
y_pred=svm_model.predict(x_test)
y_pred
```

Out[42]:
```
array([0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1,
       0])
```

In [43]:
```
y_test
```

Out[43]:
```
239    0
385    0
55     1
392    0
70     0
      ..
11     0
289    1
22     0
386    1
489    0
Name: target, Length: 111, dtype: int32
```

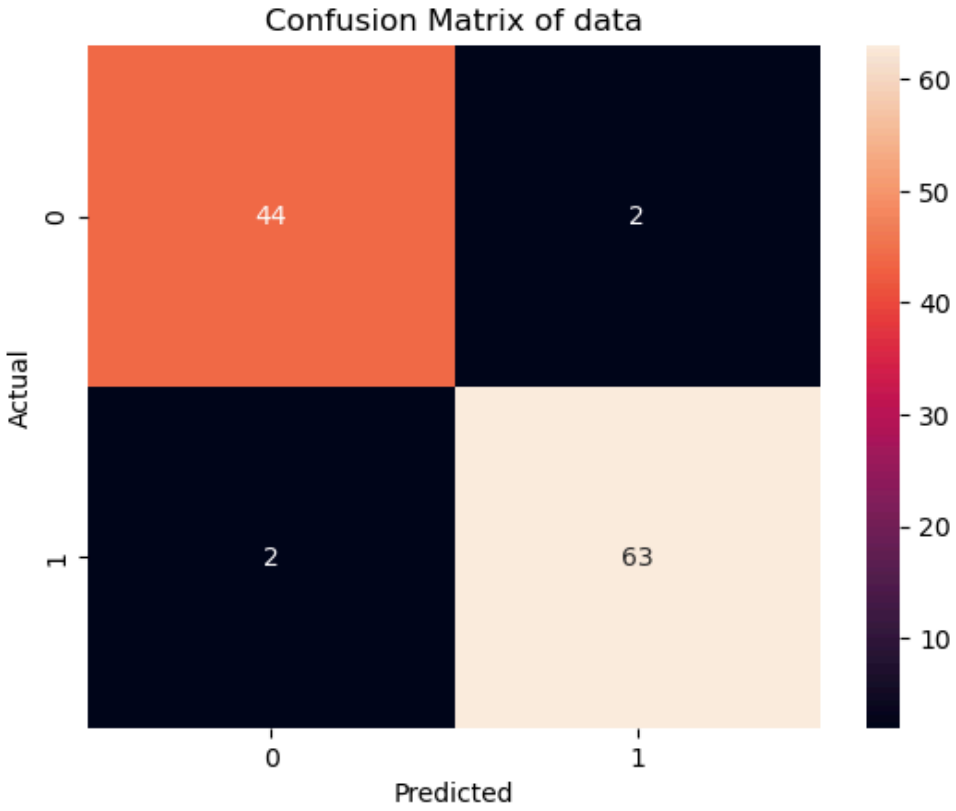
In [44]:
```
from sklearn.metrics import  confusion_matrix
```

In [45]:
```
print(confusion_matrix(y_test, y_pred))

```

```
[[44  2]
 [ 2 63]]
```

In [46]:
```
con=confusion_matrix(y_test, y_pred)
import matplotlib.pyplot as plt
sns.heatmap(con, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix of data')
plt.show()
```



In [47]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Score:")
print(accuracy)
```

```
Accuracy Score:
0.963963963963964
```

SVM is a classification algorithm that identifies the optimal hyperplane to separate data points of different classes by maximizing the margin between them.And it is ideal for distinguishing between malignant and benign tumors,Effective with multiple features, which is the case in this dataset,Can prevent overfitting with proper tuning.

DECISION TREE

In [48]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [49]:
```
tree_model = DecisionTreeClassifier()
tree_model.fit(x_train, y_train)
```

Out[49]:
```
DecisionTreeClassifier()
```

In [50]:
```
y_pred=tree_model.predict(x_test)
y_pred
```

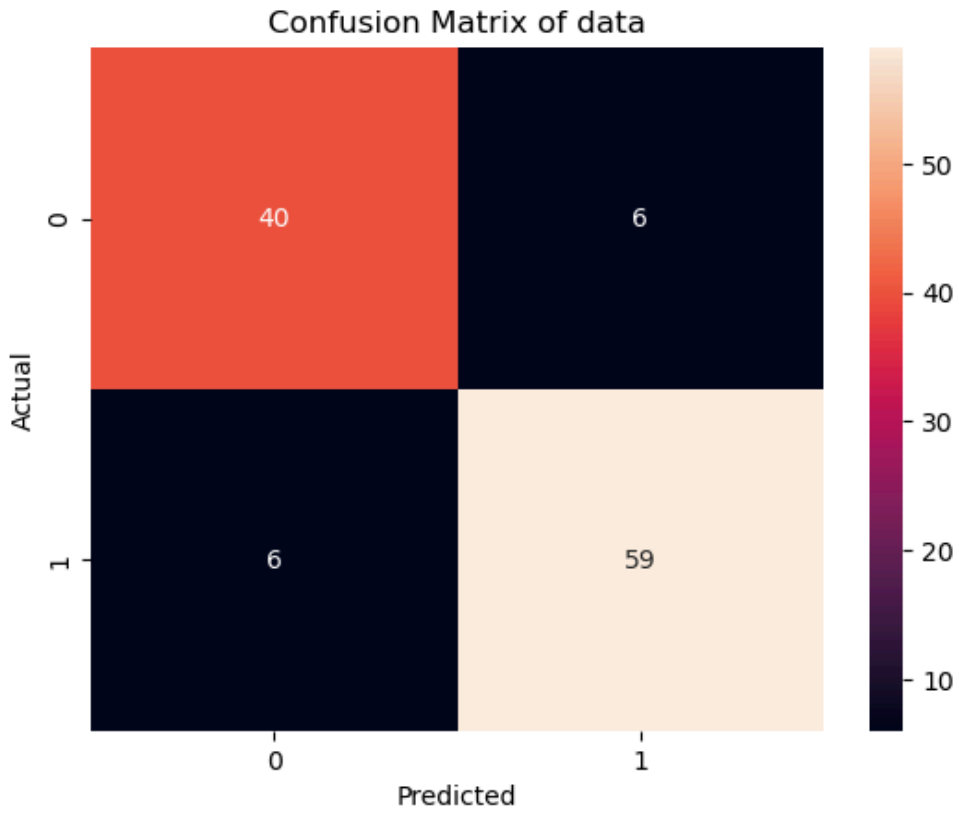
Out[50]:
```
array([0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       1])
```

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

```

```
[[40  6]
 [ 6 59]]
```

In [52]:
```
con=confusion_matrix(y_test, y_pred)
import matplotlib.pyplot as plt
sns.heatmap(con, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix of data')
plt.show()
```



In [53]:
```
from sklearn.metrics import classification_report
```

In [54]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.87      0.87        46
           1       0.91      0.91      0.91        65

    accuracy                           0.89       111
   macro avg       0.89      0.89      0.89       111
weighted avg       0.89      0.89      0.89       111
```

In [55]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Score:")
print(accuracy)
```

```
Accuracy Score:
0.8918918918918919
```

Decision trees are a versatile and interpretable tool for classification tasks, making them well-suited for analyzing the breast cancer dataset, They can handle both numerical and categorical data effectively.

RANDOM FOREST

In [56]:
```
from sklearn.ensemble import RandomForestClassifier

```

In [57]:
```
clf=RandomForestClassifier()
clf.fit(x_train,y_train)
```

Out[57]:
```
▾ RandomForestClassifier

RandomForestClassifier()
```

In [58]:
```
y_pred=clf.predict(x_test)
y_pred
```

Out[58]:
```
array([0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1,
       0])
```

In [59]:
```
1 y_test
```

Out[59]:
```
239    0
385    0
55     1
392    0
70     0
      ..
11     0
289    1
22     0
386    1
489    0
Name: target, Length: 111, dtype: int32
```

In [60]:
```
1 accuracy = accuracy_score(y_test, y_pred)
2
3 print(f"Accuracy: {accuracy}")
4
```

```
Accuracy: 0.9459459459459459
```

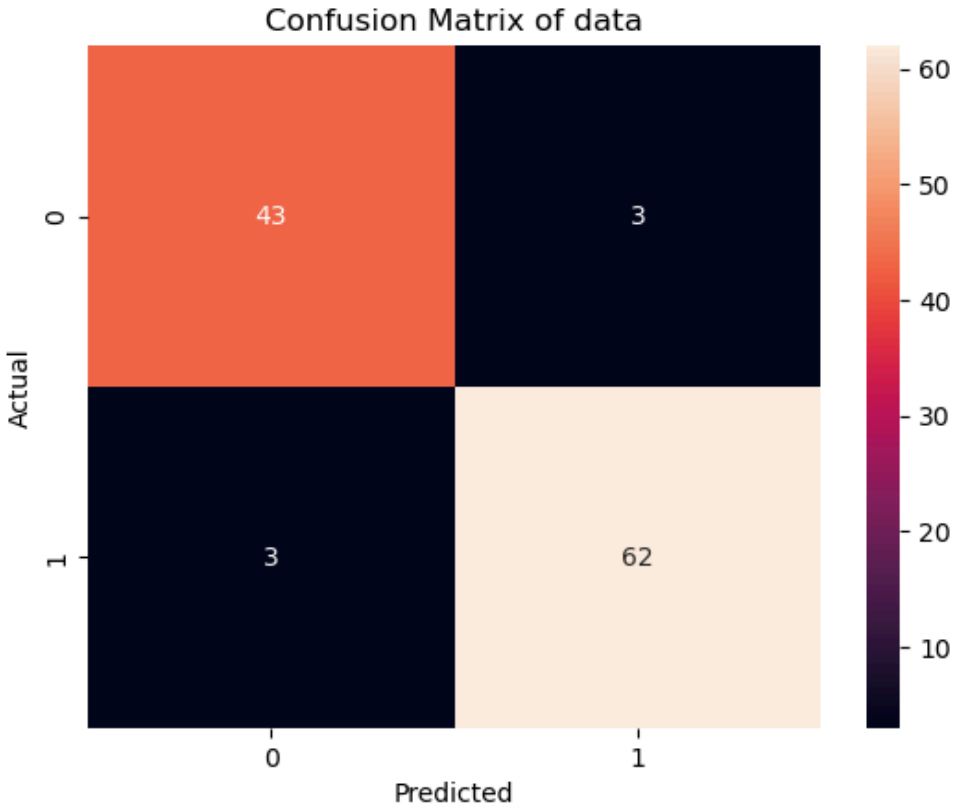
In [61]:
```
1 report = classification_report(y_test, y_pred)
2 print(f"Classification Report:\n{report}")
```

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

           0       0.93      0.93      0.93        46
           1       0.95      0.95      0.95        65

    accuracy                           0.95       111
   macro avg       0.94      0.94      0.94       111
weighted avg       0.95      0.95      0.95       111
```

In [62]:
```
con=confusion_matrix(y_test, y_pred)
import matplotlib.pyplot as plt
sns.heatmap(con, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix of data')
plt.show()
```



Random Forest is a powerful and flexible classification algorithm that enhances prediction accuracy and interpretability

k-NEAREST NEIGHBORS (k-NN):

In [63]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [64]:
```

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(x_train, y_train)

```

Out[64]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

In [65]:
```
y_pred = knn.predict(x_test)

```

In [66]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:",accuracy)
```

```
Accuracy: 0.9369369369369369
```

In [67]:
```
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)

```

```
Confusion Matrix:
 [[42  4]
 [ 3 62]]
```

In [68]:
```

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

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

           0       0.93      0.91      0.92        46
           1       0.94      0.95      0.95        65

    accuracy                           0.94       111
   macro avg       0.94      0.93      0.93       111
weighted avg       0.94      0.94      0.94       111
```

k-NN is a flexible and intuitive algorithm suitable for the breast cancer dataset, particularly for cases where local patterns and relationships are significant for classification.

When i compared the accuracies of each model,I finalize that Logistic Regression and SVM performed the best in this analysis, making them strong candidates for further exploration and validation in predicting breast cancer outcomes. The Decision Tree had the lowest accuracy at 90%, suggesting it may be more prone to overfitting.

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```