Supervised -mini project (classification)

In [5]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,classification_report,confusion_matri
```

Import the modules and libraries need for the project

In [7]:
```
# Load the dataset

data = datasets.load_breast_cancer()
data
```

```
Out[7]: {'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
          1.189e-01],
         [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
          8.902e-02],
         [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
          8.758e-02],
         ...,
         [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
          7.820e-02],
         [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
          1.240e-01],
         [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
          7.039e-02]]),
  'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
 1, 1,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
         0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
         1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
         1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
         1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
         0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
         1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
         0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
         1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
         1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
         0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
         0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
         1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
         1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
         1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
         1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
         1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
         1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]),
  'frame': None,
  'target_names': array(['malignant', 'benign'], dtype='<U9'),
  'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic) d
 ataset\n--------------------------------------------\n\n**Data Set Characterist
 ics:**\n\n:Number of Instances: 569\n\n:Number of Attributes: 30 numeric, predi
 ctive attributes and the class\n\n:Attribute Information:\n    - radius (mean o
 f distances from center to points on the perimeter)\n    - texture (standard de
 viation of gray-scale values)\n    - perimeter\n    - area\n    - smoothness (l
 ocal variation in radius lengths)\n    - compactness (perimeter^2 / area - 1.0)
 \n    - concavity (severity of concave portions of the contour)\n    - concave 
 points (number of concave portions of the contour)\n    - symmetry\n    - fract
 al dimension ("coastline approximation" - 1)\n\n    The mean, standard error, a
 nd "worst" or largest (mean of the three\n    worst/largest values) of these fe
 atures were computed for each image,\n    resulting in 30 features.  For instan
 ce, field 0 is Mean Radius, field\n    10 is Radius SE, field 20 is Worst Radiu
 s.\n\n    - class:\n            - WDBC-Malignant\n            - WDBC-Benign\n   
 \n:Summary Statistics:\n\n===================================== ====== ======\n
 Min    Max\n===================================== ====== ======\nradius (mean):
 6.981  28.11\ntexture (mean):                        9.71   39.28\nperimeter (me
 an):                     43.79  188.5\narea (mean):                            14
```

```
3.5  2501.0\nsmoothness (mean):                    0.053  0.163\ncompactness (m
ean):                   0.019  0.345\nconcavity (mean):                     0.0
0.427\nconcave points (mean):               0.0    0.201\nsymmetry (mean):
0.106  0.304\nfractal dimension (mean):            0.05   0.097\nradius (stand
ard error):             0.112  2.873\ntexture (standard error):             0.
36   4.885\nperimeter (standard error):          0.757  21.98\narea (standard
error):                6.802  542.2\nsmoothness (standard error):          0.00
2  0.031\ncompactness (standard error):         0.002  0.135\nconcavity (standa
rd error):           0.0    0.396\nconcave points (standard error):      0.0
0.053\nsymmetry (standard error):           0.008  0.079\nfractal dimension (s
tandard error):   0.001  0.03\nradius (worst):                       7.93   36.
04\ntexture (worst):                      12.02  49.54\nperimeter (worst):
50.41  251.2\narea (worst):                         185.2  4254.0\nsmoothness
(worst):                   0.071  0.223\ncompactness (worst):
0.027  1.058\nconcavity (worst):                    0.0    1.252\nconcave point
s (worst):              0.0    0.291\nsymmetry (worst):                     0.
156  0.664\nfractal dimension (worst):           0.055  0.208\n===============
===================== ====== ======\n\n:Missing Attribute Values: None\n\n:Cla
ss Distribution: 212 - Malignant, 357 - Benign\n\n:Creator:  Dr. William H. Wol
berg, W. Nick Street, Olvi L. Mangasarian\n\n:Donor: Nick Street\n\n:Date: Nove
mber, 1995\n\nThis is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) dat
asets.\nhttps://goo.gl/U2Uwz2\n\nFeatures are computed from a digitized image o
f a fine needle\naspirate (FNA) of a breast mass.  They describe\ncharacteristi
cs of the cell nuclei present in the image.\n\nSeparating plane described above
was obtained using\nMultisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision
Tree\nConstruction Via Linear Programming." Proceedings of the 4th\nMidwest Art
ificial Intelligence and Cognitive Science Society,\npp. 97-101, 1992], a class
ification method which uses linear\nprogramming to construct a decision tree.
Relevant features\nwere selected using an exhaustive search in the space of 1-4
\nfeatures and 1-3 separating planes.\n\nThe actual linear program used to obta
in the separating plane\nin the 3-dimensional space is that described in:\n[K.
P. Bennett and O. L. Mangasarian: "Robust Linear\nProgramming Discrimination of
Two Linearly Inseparable Sets",\nOptimization Methods and Software 1, 1992, 23-
34].\n\nThis database is also available through the UW CS ftp server:\n\nftp ft
p.cs.wisc.edu\ncd math-prog/cpo-dataset/machine-learn/WDBC/\n\n|details-start|
\n**References**\n|details-split|\n\n- W.N. Street, W.H. Wolberg and O.L. Manga
sarian. Nuclear feature extraction\n  for breast tumor diagnosis. IS&T/SPIE 199
3 International Symposium on\n  Electronic Imaging: Science and Technology, vol
ume 1905, pages 861-870,\n  San Jose, CA, 1993.\n- O.L. Mangasarian, W.N. Stree
t and W.H. Wolberg. Breast cancer diagnosis and\n  prognosis via linear program
ming. Operations Research, 43(4), pages 570-577,\n  July-August 1995.\n- W.H. W
olberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques\n  to di
agnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)\n  16
3-171.\n\n|details-end|\n',
 'feature_names': array(['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'], dtype='<U23'),
 'filename': 'breast_cancer.csv',
 'data_module': 'sklearn.datasets.data'}
```

In [8]:
```
x = data.data
y = data.target
```

```
df = pd.DataFrame(x,columns = data.feature_names)

df['target'] = y
```

Split the data into features and target by giving them a name of x and y. x have features and y have target values

In [10]:
```
df.head() # took the first 5 rows of the data
```

Out[10]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	s
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	

5 rows × 31 columns

In [11]:
```
df.info() # in info we can understand more about the data
```

```
<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 [12]:

```
# by using isnull function we can find out the null values from the data
df.isnull().sum() # there is no null values in the dataset
```

Out[12]:
```
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
```

In [13]:
```
# by using the function duplicated we can find out the duplicates values

df.duplicated().sum() # from the dataset we understood that there is no duplicat
```

Out[13]:
```
0
```

In [14]:
```
df.describe() # gives the statistics of the data
```

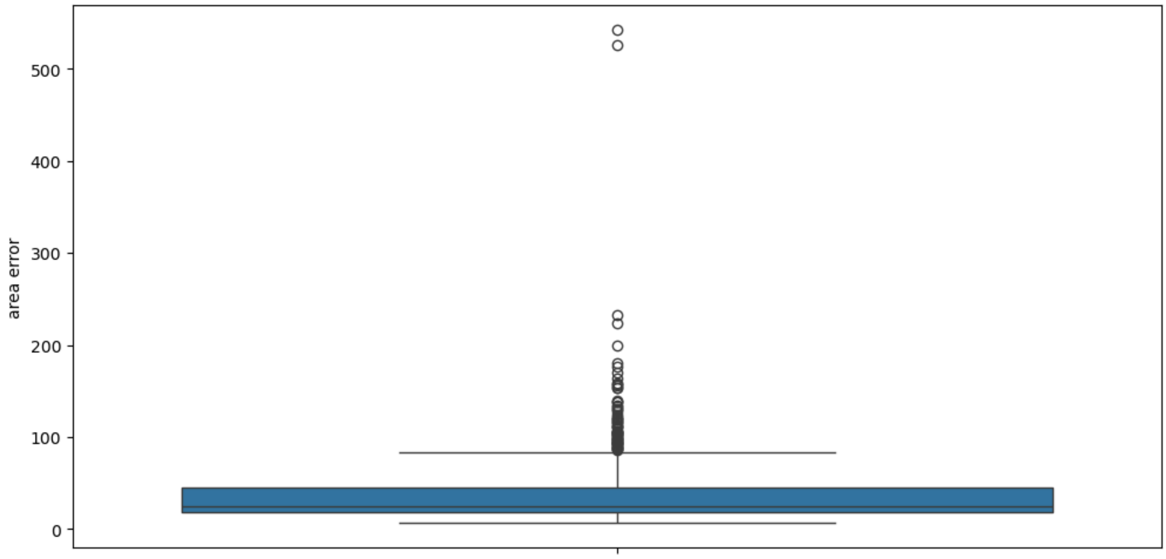
Out[14]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	co
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0

8 rows × 31 columns

In [15]:

```
# using boxplot we can find out the outliers of the data
plt.figure(figsize = (12,6))
sns.boxplot(data=df['area error'])
plt.show()
```



In [16]:

```
df.skew() # there is positive skewness in the data
```

Out[16]:
```
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 [17]:
```
# Standard scaling

from sklearn.preprocessing import StandardScaler

x = df.drop('target',axis = 1)
y = df['target']

x
```

Out[17]:

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

569 rows × 30 columns

In [18]:
```
scaler = StandardScaler()

x_scaled = scaler.fit_transform(x)
```

In [19]:
```
x_train,x_test,y_train,y_test = train_test_split(x_scaled,y)
```

In [20]:
```
x_train
```

Out[20]:
```
array([[ 1.22771051,  0.60977338,  1.16283917, ...,  0.35468496,
         0.33689357, -0.43478238],
       [ 1.43788102, -0.77948569,  1.41409939, ...,  0.96070354,
        -0.74054832, -1.18788333],
       [-0.87683468, -0.57237672, -0.8670139 , ..., -0.61357437,
         0.15731992, -0.28460551],
       ...,
       [-0.13271749, -0.03715128, -0.10334759, ...,  0.39731943,
        -0.25359635,  0.24627802],
       [ 0.23366083, -0.1209257 ,  0.24182629, ..., -0.46526731,
        -0.07887605,  0.45630397],
       [ 3.97128765, -0.19073771,  3.97612984, ...,  0.68357946,
        -2.0266839 , -1.59020217]])
```

In [21]:
```
y_train
```

```
Out[21]: 134    0
         161    0
         411    1
         291    1
         55     1
               ..
         117    0
         331    1
         423    1
         147    1
         212    0
         Name: target, Length: 426, dtype: int32
```

```
In [22]: column = df.select_dtypes(include = 'number')
         column
```

Out[22]:

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

569 rows × 31 columns

```
In [23]: # removing outliers by using IQR method

         def remove_outliers(df):
             df_filtered = df.copy()
             for col in df.columns:
                 # Calculate Q1, Q3 and IQR
                 q1 = df_filtered[col].quantile(0.25)
                 q3 = df_filtered[col].quantile(0.75)
                 IQR = q3 - q1

                 # Calculate the lower and upper bounds for outliers
                 lower_whisker = q1 - 1.5 * IQR
                 upper_whisker = q3 + 1.5 * IQR

                 # Filter the rows where the values are within the bounds
                 df_filtered = df_filtered[(df_filtered[col] >= lower_whisker) & (df_filt
```

```python
    return df_filtered
```

In [24]:
```python
dff= remove_outliers(df)
dff
```

Out[24]:

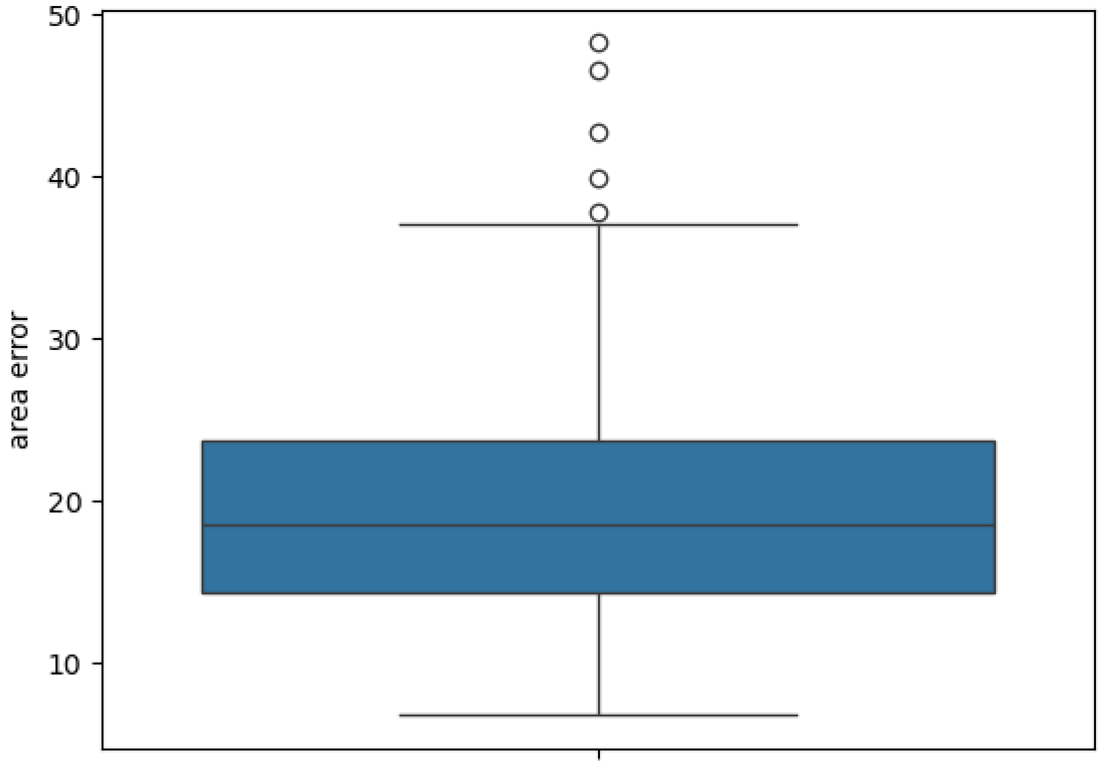
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points
19	13.540	14.36	87.46	566.3	0.09779	0.08129	0.06664	0.047810
20	13.080	15.71	85.63	520.0	0.10750	0.12700	0.04568	0.031100
21	9.504	12.44	60.34	273.9	0.10240	0.06492	0.02956	0.020760
37	13.030	18.42	82.61	523.8	0.08983	0.03766	0.02562	0.029230
46	8.196	16.84	51.71	201.9	0.08600	0.05943	0.01588	0.005917
...	...	...	...	...	...	...	...	...
551	11.130	22.44	71.49	378.4	0.09566	0.08194	0.04824	0.022570
552	12.770	29.43	81.35	507.9	0.08276	0.04234	0.01997	0.014990
554	12.880	28.92	82.50	514.3	0.08123	0.05824	0.06195	0.023430
555	10.290	27.61	65.67	321.4	0.09030	0.07658	0.05999	0.027380
560	14.050	27.15	91.38	600.4	0.09929	0.11260	0.04462	0.043040

255 rows × 31 columns

In [25]:
```python
print("orginal dataframe shape:",df.shape)
print("filtered datafram:",dff.shape)
```

```
orginal dataframe shape: (569, 31)
filtered datafram: (255, 31)
```

In [26]:
```python
sns.boxplot(data=dff["area error"])
plt.show()
```



In [27]:
```
# after removing the outliers there is change in skewness
dff.skew()
```

Out[27]:
```
mean radius                 -0.111407
mean texture                 0.887321
mean perimeter              -0.089486
mean area                    0.235998
mean smoothness              0.206140
mean compactness             0.619329
mean concavity               0.771774
mean concave points          0.783523
mean symmetry                0.176187
mean fractal dimension       0.492844
radius error                 0.963516
texture error                0.692455
perimeter error              1.086510
area error                   0.942534
smoothness error             0.648031
compactness error            0.933022
concavity error              0.816751
concave points error         0.377017
symmetry error               0.531354
fractal dimension error      0.811067
worst radius                -0.120318
worst texture                0.586718
worst perimeter             -0.053960
worst area                   0.221604
worst smoothness             0.172241
worst compactness            0.435714
worst concavity              0.602731
worst concave points         0.107525
worst symmetry               0.241687
worst fractal dimension      0.284154
target                       0.000000
dtype: float64
```

In [28]:
```python
# LOGISTICS REGRESSION

log_reg = LogisticRegression()

log_reg.fit(x_train,y_train)
```

Out[28]:
```
▾ LogisticRegression
LogisticRegression()
```

In [29]:
```python
y_pred = log_reg.predict(x_test)
y_pred
```

Out[29]:
```
array([1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0,
       1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1,
       0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0,
       0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1])
```

In [30]:
```python
log_accuracy = accuracy_score(y_test,y_pred)
log_accuracy
```

Out[30]:
```
0.972027972027972
```

In [31]:
```python
log_cr = classification_report(y_test,y_pred)
print(log_cr)
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        50
           1       0.98      0.98      0.98        93

    accuracy                           0.97       143
   macro avg       0.97      0.97      0.97       143
weighted avg       0.97      0.97      0.97       143
```

In [32]:
```python
log_cm = confusion_matrix(y_test,y_pred)
print(log_cm)
```

```
[[48  2]
 [ 2 91]]
```

logistic regression is mainly used to classify data into two categories. our data is a binary classification so logistic regression is suitable for this. Here we spilt the data in two and trained and after predict them.

In [34]:
```python
# Decision tree classifier

from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()

dt.fit(x_train,y_train)
```

Out[34]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [35]:
```python
dt_y_pred = dt.predict(x_test)
dt_y_pred
```

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

In [36]:
```python
dt_accuracy_score = accuracy_score(y_test,dt_y_pred)
dt_accuracy_score
```

Out[36]:
```
0.9090909090909091
```

Decision tree is used for both classification and regression tasks.It split the data into subset and make predictions

In [38]:
```python
# Random forest

from sklearn.ensemble import RandomForestClassifier

random_fc = RandomForestClassifier()

random_fc.fit(x_train,y_train)
```

Out[38]:

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [39]:
```python
rfc_y_pred = random_fc.predict(x_test)

rfc_y_pred
```

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

In [40]:
```python
rfc_accuracy = accuracy_score(y_test,rfc_y_pred)
rfc_accuracy
```

Out[40]:
```
0.9440559440559441
```

Random forest is a collection of decision tree. it combines multiple models to create a stronger model.

In [42]:
```
# Support machine vector

from sklearn.svm import SVC

SVM_classifier = SVC()

SVM_classifier.fit(x_train,y_train)

SVM_y_pred = SVM_classifier.predict(x_test)
SVM_y_pred

SVM_accuracy = accuracy_score(y_test,SVM_y_pred)
SVM_accuracy
```

Out[42]:
```
0.958041958041958
```

SVC is commonly used in medical diagnosis,text classification and image recogination. It works like seprate the data into 2 classes having a boundary

In [44]:
```
# K-NN

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(x_train,y_train)
```

Out[44]:
```
KNeighborsClassifier

KNeighborsClassifier(n_neighbors=3)
```

In [45]:
```
knn_y_pred = knn.predict(x_test)
knn_y_pred

knn_accuracy = accuracy_score(y_test,knn_y_pred)
knn_accuracy
```

Out[45]:
```
0.951048951048951
```

KNN is to predict the label or values of a data point based on the labels or values of its k-nearest neighbors in the feature space.

In [47]:
```
predictions = {
    "log_reg" : y_pred,
    "Decision Tree" : dt_y_pred,
    "Random Forest" : rfc_y_pred,
    "Support vector machine" : SVM_y_pred,
    "KNN" : knn_y_pred
}

results = {
    "Model" : [],
    "Accuracy" : [],
    "precision" : [],
    "Recall" : [],
    "F1-score" : []
}
```

```python
for model_name,y_pred in predictions.items():
    results["Model"].append(model_name)
    results["Accuracy"].append(accuracy_score(y_test,y_pred))
    results["precision"].append(precision_score(y_test,y_pred))
    results["Recall"].append(recall_score(y_test,y_pred))
    results["F1-score"].append(f1_score(y_test,y_pred))

results
```

Out[47]:

```
{'Model': ['log_reg',
  'Decision Tree',
  'Random Forest',
  'Support vector machine',
  'KNN'],
 'Accuracy': [0.972027972027972,
  0.9090909090909091,
  0.9440559440559441,
  0.958041958041958,
  0.951048951048951],
 'precision': [0.978494623655914,
  0.9347826086956522,
  0.9473684210526315,
  0.967741935483871,
  0.9479166666666666],
 'Recall': [0.978494623655914,
  0.9247311827956989,
  0.967741935483871,
  0.967741935483871,
  0.978494623655914],
 'F1-score': [0.978494623655914,
  0.9297297297297298,
  0.9574468085106383,
  0.967741935483871,
  0.9629629629629629]}
```

In [48]:

```python
# converting the model into a DataFrame
results_df = pd.DataFrame(results)

print(results_df)
```

```
                    Model  Accuracy  precision    Recall  F1-score
0                 log_reg  0.972028   0.978495  0.978495  0.978495
1           Decision Tree  0.909091   0.934783  0.924731  0.929730
2           Random Forest  0.944056   0.947368  0.967742  0.957447
3  Support vector machine  0.958042   0.967742  0.967742  0.967742
4                     KNN  0.951049   0.947917  0.978495  0.962963
```

In [49]:

```python
# sorting by F1-SCORE to identify the best model and worst model

results_df = results_df.sort_values(by="F1-score",ascending = False)
print(results_df)
```

```
                    Model  Accuracy  precision    Recall  F1-score
0                 log_reg  0.972028   0.978495  0.978495  0.978495
3  Support vector machine  0.958042   0.967742  0.967742  0.967742
4                     KNN  0.951049   0.947917  0.978495  0.962963
2           Random Forest  0.944056   0.947368  0.967742  0.957447
1           Decision Tree  0.909091   0.934783  0.924731  0.929730
```

From the above we can conclude that the best model is Support vector machine and the worst one is Decision tree

In [ ]:
```

```