BREAST CANCER DETECTION MODEL

NAME: SOORAJ ARUN

OVERVIEW

One of the main causes of cancer-related mortality for women globally is breast cancer. Treatment results and survival rates can be greatly enhanced by early identification. This project's objective is to develop a machine learning model that, using pertinent characteristics extracted from clinical and diagnostic data, reliably classifies tumors as benign (non-cancerous) or malignant (cancerous).

OBJECTIVE

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

DATASET DESCRIPTION:

BREAST CANCER: SKLEARN LIBRARY

IMPORTING LIBRARIES

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.feature_selection import VarianceThreshold
from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

from sklearn.model_selection import GridSearchCV

import joblib
```

```python
import warnings
warnings.filterwarnings("ignore")
```

IMPORTING DATASET

```python
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
data
```

```
{'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) dataset\n--------------------------------------------\n\
n**Data Set Characteristics:**\n\n:Number of Instances: 569\n\n:Number
of Attributes: 30 numeric, predictive attributes and the class\n\
n:Attribute Information:\n    - radius (mean of distances from center
to points on the perimeter)\n    - texture (standard deviation of
gray-scale values)\n    - perimeter\n    - area\n    - smoothness
(local 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    - fractal dimension ("coastline
approximation" - 1)\n\n    The mean, standard error, and "worst" or
largest (mean of the three\n    worst/largest values) of these
features were computed for each image,\n    resulting in 30 features.
For instance, field 0 is Mean Radius, field\n    10 is Radius SE,
field 20 is Worst Radius.\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 (mean):                     43.79  188.5\narea
(mean):                          143.5  2501.0\nsmoothness (mean):
0.053  0.163\ncompactness (mean):                   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 (standard 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):           0.002  0.031\ncompactness
(standard error):         0.002  0.135\nconcavity (standard error):
0.0    0.396\nconcave points (standard error):      0.0    0.053\
nsymmetry (standard error):             0.008  0.079\nfractal dimension
(standard 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 points (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:Class Distribution: 212 - Malignant, 357 -
Benign\n\n:Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L.
Mangasarian\n\n:Donor: Nick Street\n\n:Date: November, 1995\n\nThis is
a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.\
nhttps://goo.gl/U2Uwz2\n\nFeatures are computed from a digitized image
of a fine needle\naspirate (FNA) of a breast mass.  They describe\
ncharacteristics 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 Artificial
Intelligence and Cognitive Science Society,\npp. 97-101, 1992], a
classification 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 obtain 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 ftp.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. Mangasarian. Nuclear feature
extraction\n  for breast tumor diagnosis. IS&T/SPIE 1993 International
Symposium on\n  Electronic Imaging: Science and Technology, volume
1905, pages 861-870,\n  San Jose, CA, 1993.\n- O.L. Mangasarian, W.N.
Street and W.H. Wolberg. Breast cancer diagnosis and\n  prognosis via
linear programming. Operations Research, 43(4), pages 570-577,\n
July-August 1995.\n- W.H. Wolberg, W.N. Street, and O.L. Mangasarian.
Machine learning techniques\n  to diagnose breast cancer from fine-
needle aspirates. Cancer Letters 77 (1994)\n  163-171.\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'}

# converting to dataframe
df = pd.DataFrame(data.data,  columns=data.feature_names)

df['target'] = data.target
df.head()

   mean radius  mean texture  mean perimeter  mean area  mean
smoothness  \
0        17.99         10.38          122.80     1001.0
0.11840
1        20.57         17.77          132.90     1326.0
0.08474
2        19.69         21.25          130.00     1203.0
0.10960
3        11.42         20.38           77.58      386.1
0.14250
4        20.29         14.34          135.10     1297.0
0.10030

   mean compactness  mean concavity  mean concave points  mean
symmetry  \
0           0.27760          0.3001              0.14710
0.2419
1           0.07864          0.0869              0.07017
0.1812
2           0.15990          0.1974              0.12790
0.2069
3           0.28390          0.2414              0.10520
0.2597
4           0.13280          0.1980              0.10430
0.1809
```

```
   mean fractal dimension  ...  worst texture  worst perimeter  worst
area  \
0                 0.07871  ...          17.33           184.60
2019.0
1                 0.05667  ...          23.41           158.80
1956.0
2                 0.05999  ...          25.53           152.50
1709.0
3                 0.09744  ...          26.50            98.87
567.7
4                 0.05883  ...          16.67           152.20
1575.0

   worst smoothness  worst compactness  worst concavity  worst concave
points  \
0            0.1622             0.6656           0.7119
0.2654
1            0.1238             0.1866           0.2416
0.1860
2            0.1444             0.4245           0.4504
0.2430
3            0.2098             0.8663           0.6869
0.2575
4            0.1374             0.2050           0.4000
0.1625

   worst symmetry  worst fractal dimension  target
0          0.4601                  0.11890       0
1          0.2750                  0.08902       0
2          0.3613                  0.08758       0
3          0.6638                  0.17300       0
4          0.2364                  0.07678       0

[5 rows x 31 columns]
```

```
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

df.describe()

       mean radius  mean texture  mean perimeter    mean area  \
count   569.000000    569.000000      569.000000   569.000000
mean     14.127292     19.289649       91.969033   654.889104
std       3.524049      4.301036       24.298981   351.914129
min       6.981000      9.710000       43.790000   143.500000
25%      11.700000     16.170000       75.170000   420.300000
50%      13.370000     18.840000       86.240000   551.100000
75%      15.780000     21.800000      104.100000   782.700000
max      28.110000     39.280000      188.500000  2501.000000

       mean smoothness  mean compactness  mean concavity  mean concave
points  \
count       569.000000        569.000000      569.000000
569.000000
mean          0.096360          0.104341        0.088799
0.048919
std           0.014064          0.052813        0.079720
0.038803
min           0.052630          0.019380        0.000000
0.000000
25%           0.086370          0.064920        0.029560
0.020310
```

```
50%           0.095870          0.092630       0.061540
0.033500
75%           0.105300          0.130400       0.130700
0.074000
max           0.163400          0.345400       0.426800
0.201200

       mean symmetry  mean fractal dimension  ...  worst texture  \
count    569.000000              569.000000  ...     569.000000
mean       0.181162                0.062798  ...      25.677223
std        0.027414                0.007060  ...       6.146258
min        0.106000                0.049960  ...      12.020000
25%        0.161900                0.057700  ...      21.080000
50%        0.179200                0.061540  ...      25.410000
75%        0.195700                0.066120  ...      29.720000
max        0.304000                0.097440  ...      49.540000

       worst perimeter   worst area  worst smoothness  worst
compactness  \
count       569.000000   569.000000        569.000000
569.000000
mean        107.261213   880.583128          0.132369
0.254265
std          33.602542   569.356993          0.022832
0.157336
min          50.410000   185.200000          0.071170
0.027290
25%          84.110000   515.300000          0.116600
0.147200
50%          97.660000   686.500000          0.131300
0.211900
75%         125.400000  1084.000000          0.146000
0.339100
max         251.200000  4254.000000          0.222600
1.058000

       worst concavity  worst concave points  worst symmetry  \
count       569.000000            569.000000      569.000000
mean          0.272188              0.114606        0.290076
std           0.208624              0.065732        0.061867
min           0.000000              0.000000        0.156500
25%           0.114500              0.064930        0.250400
50%           0.226700              0.099930        0.282200
75%           0.382900              0.161400        0.317900
max           1.252000              0.291000        0.663800

       worst fractal dimension      target
count               569.000000  569.000000
mean                  0.083946    0.627417
std                   0.018061    0.483918
```

```
min                      0.055040    0.000000
25%                      0.071460    0.000000
50%                      0.080040    1.000000
75%                      0.092080    1.000000
max                      0.207500    1.000000

[8 rows x 31 columns]
```

```
df.shape
```

```
(569, 31)
```

DATA CLEANING AND PRE PROCESSING

```
#checking for duplicated
df.duplicated()
```

```
0      False
1      False
2      False
3      False
4      False
       ...
564    False
565    False
566    False
567    False
568    False
Length: 569, dtype: bool
```

```
df.duplicated().sum()
```

```
0
```

Checking for missing values

```
df.isnull()
```

```
     mean radius  mean texture  mean perimeter  mean area  mean
smoothness  \
0          False         False           False      False
False
1          False         False           False      False
False
2          False         False           False      False
False
3          False         False           False      False
False
4          False         False           False      False
False
..           ...           ...             ...        ...
```

```
...
564        False         False           False      False
False
565        False         False           False      False
False
566        False         False           False      False
False
567        False         False           False      False
False
568        False         False           False      False
False

     mean compactness  mean concavity  mean concave points  mean
symmetry  \
0               False           False                False
False
1               False           False                False
False
2               False           False                False
False
3               False           False                False
False
4               False           False                False
False
..                ...             ...                  ...
...
564             False           False                False
False
565             False           False                False
False
566             False           False                False
False
567             False           False                False
False
568             False           False                False
False

     mean fractal dimension  ...  worst texture  worst perimeter
worst area  \
0                     False  ...          False            False
False
1                     False  ...          False            False
False
2                     False  ...          False            False
False
3                     False  ...          False            False
False
4                     False  ...          False            False
False
```

```
..                       ...  ...            ...              ...
...
564                    False  ...          False            False
False
565                    False  ...          False            False
False
566                    False  ...          False            False
False
567                    False  ...          False            False
False
568                    False  ...          False            False
False

     worst smoothness  worst compactness  worst concavity  \
0               False              False            False
1               False              False            False
2               False              False            False
3               False              False            False
4               False              False            False
..                ...                ...              ...
564             False              False            False
565             False              False            False
566             False              False            False
567             False              False            False
568             False              False            False

     worst concave points  worst symmetry  worst fractal dimension
target
0                   False           False                    False
False
1                   False           False                    False
False
2                   False           False                    False
False
3                   False           False                    False
False
4                   False           False                    False
False
..                    ...             ...                      ...
...
564                 False           False                    False
False
565                 False           False                    False
False
566                 False           False                    False
False
567                 False           False                    False
False
568                 False           False                    False
```

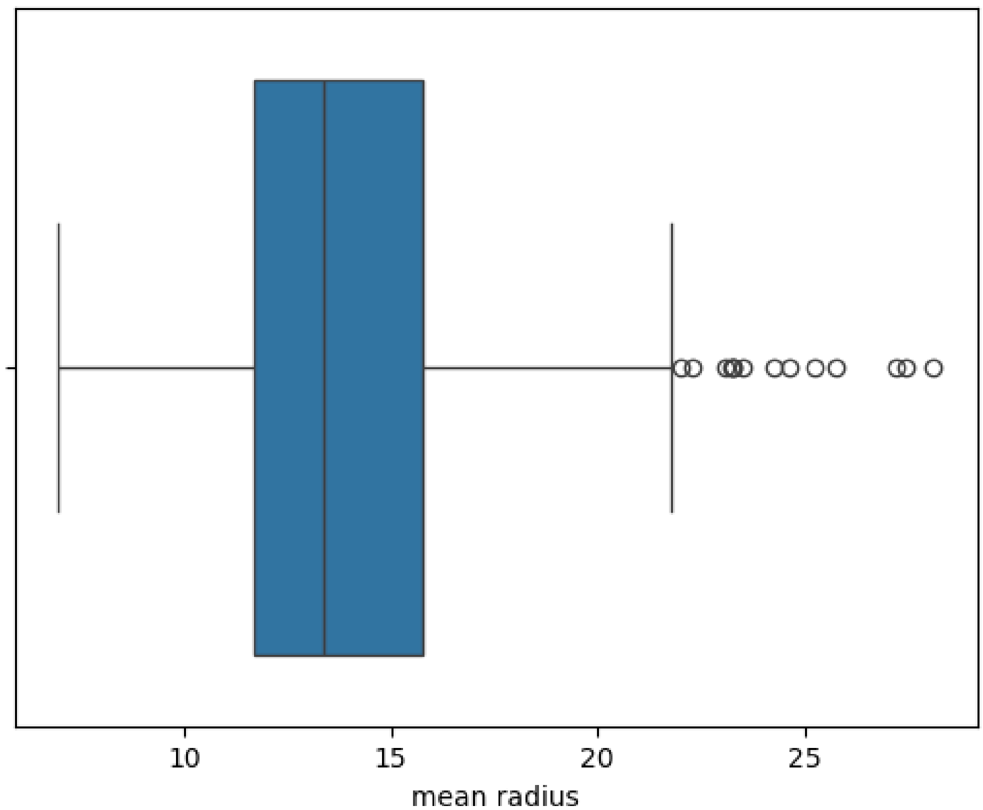
```
False

[569 rows x 31 columns]
```

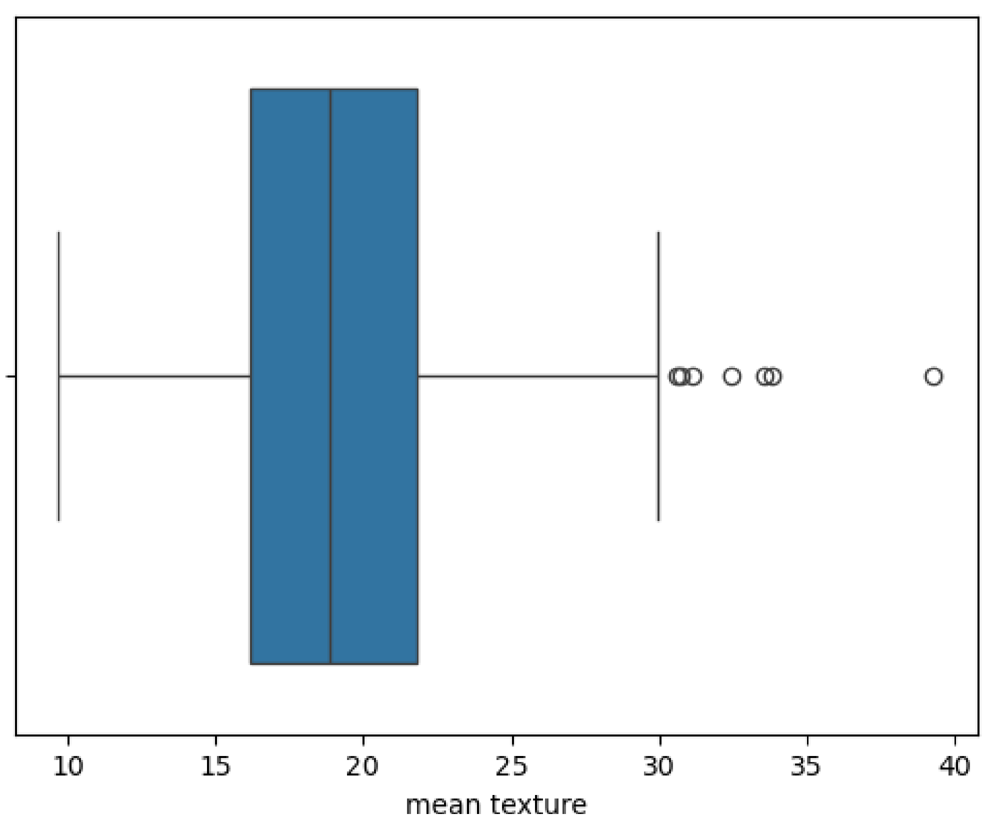
```
df.isnull().sum()
```

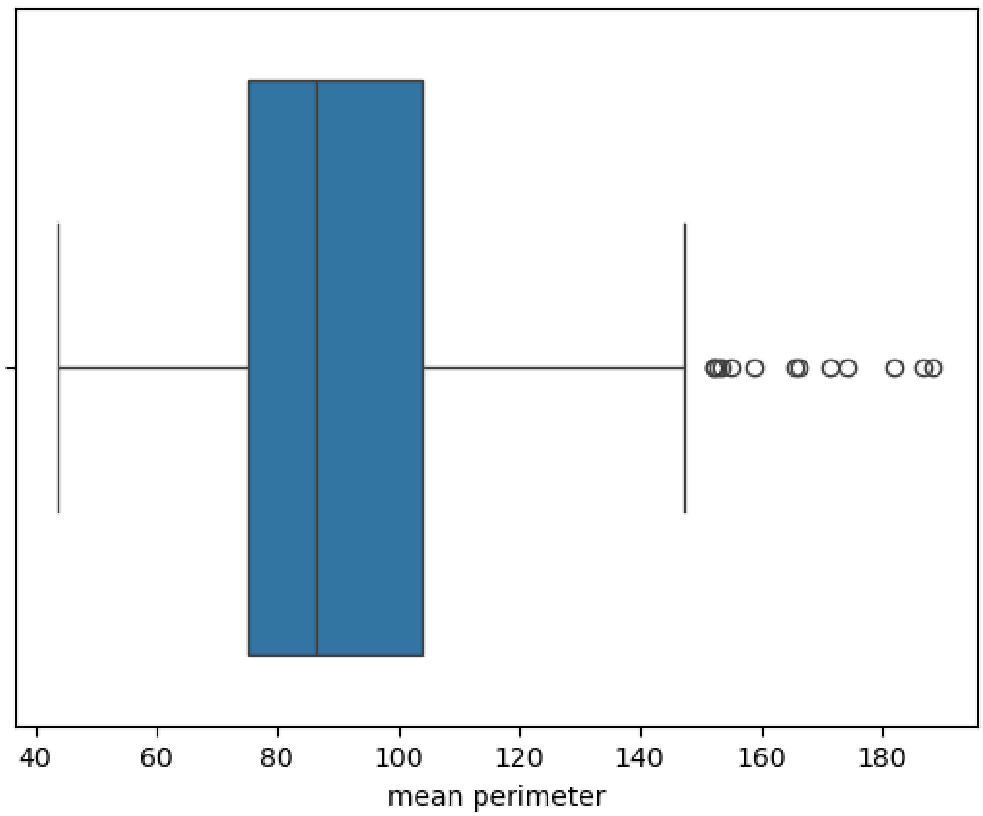
```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius 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
```

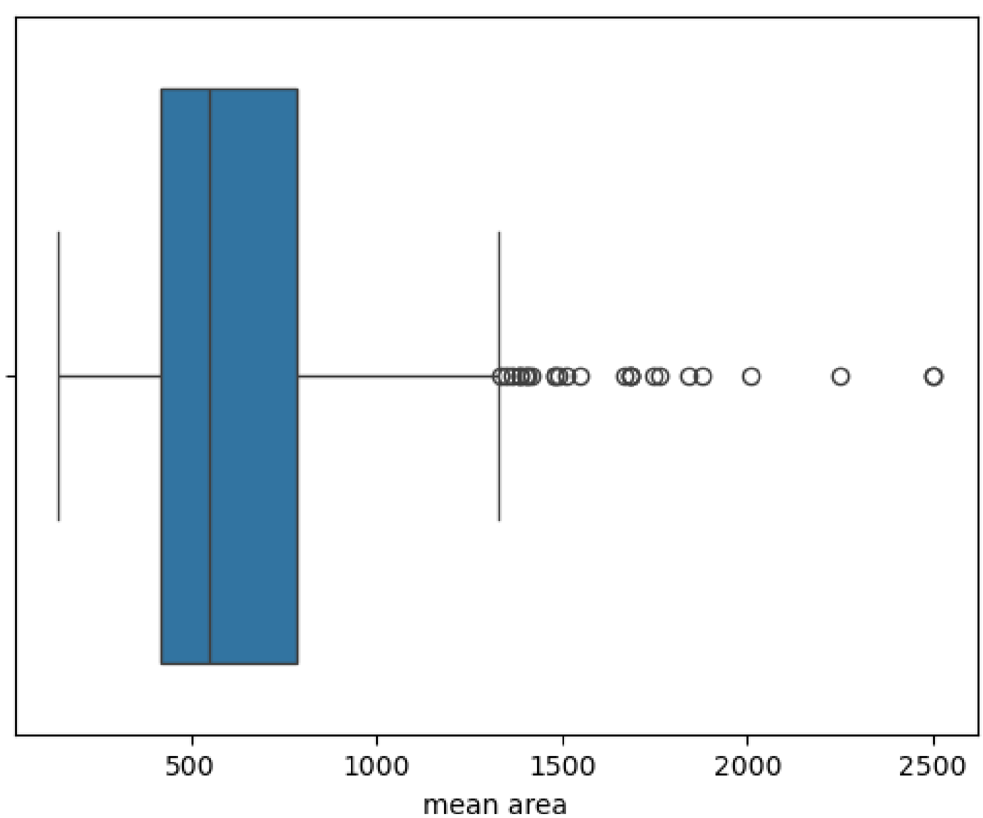
checking for Outliers

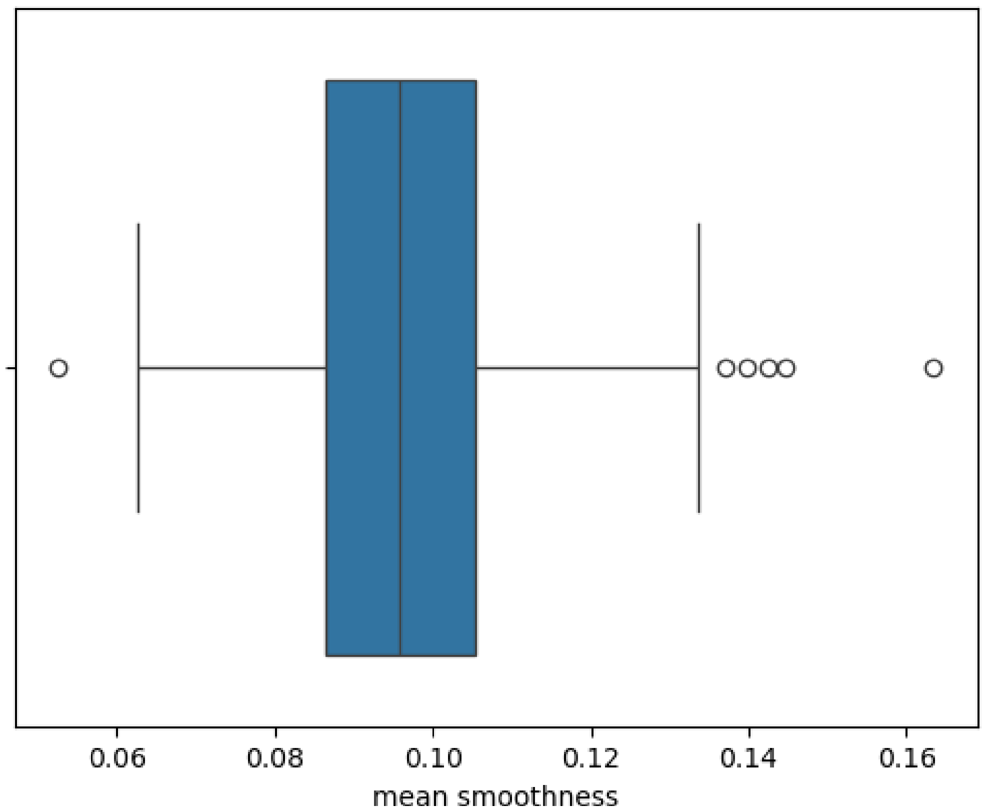
```
# Visualising outliers in each feature using boxplot method
for i in df.columns:
    sns.boxplot(data=df,x=i)
    plt.show()
```


10
15
20
25
mean radius

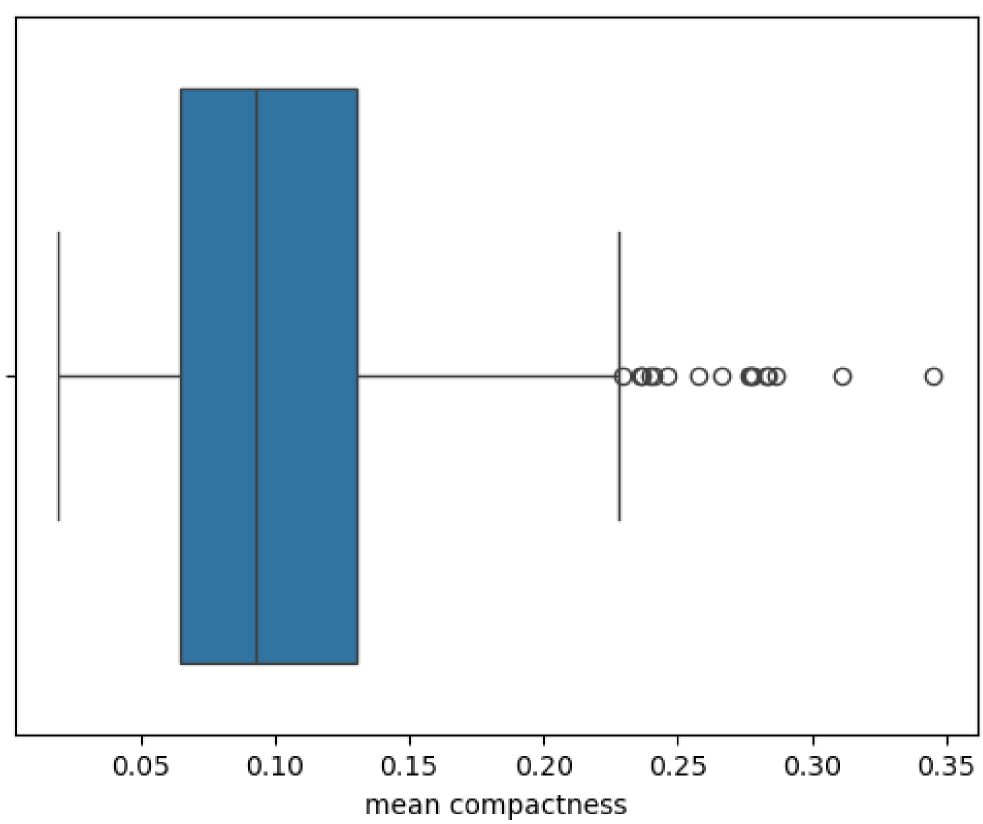

10
15
20
25
30
35
40
mean texture


40
60
80
100
120
140
160
180
mean perimeter

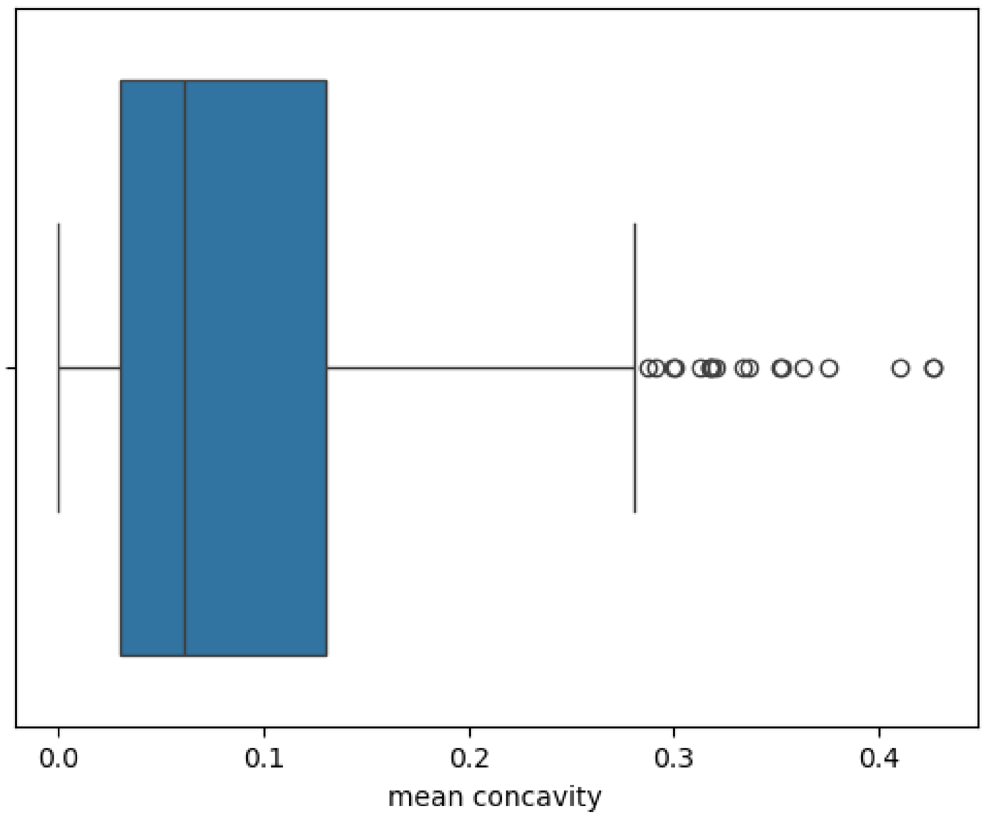

500
1000
1500
2000
2500
mean area

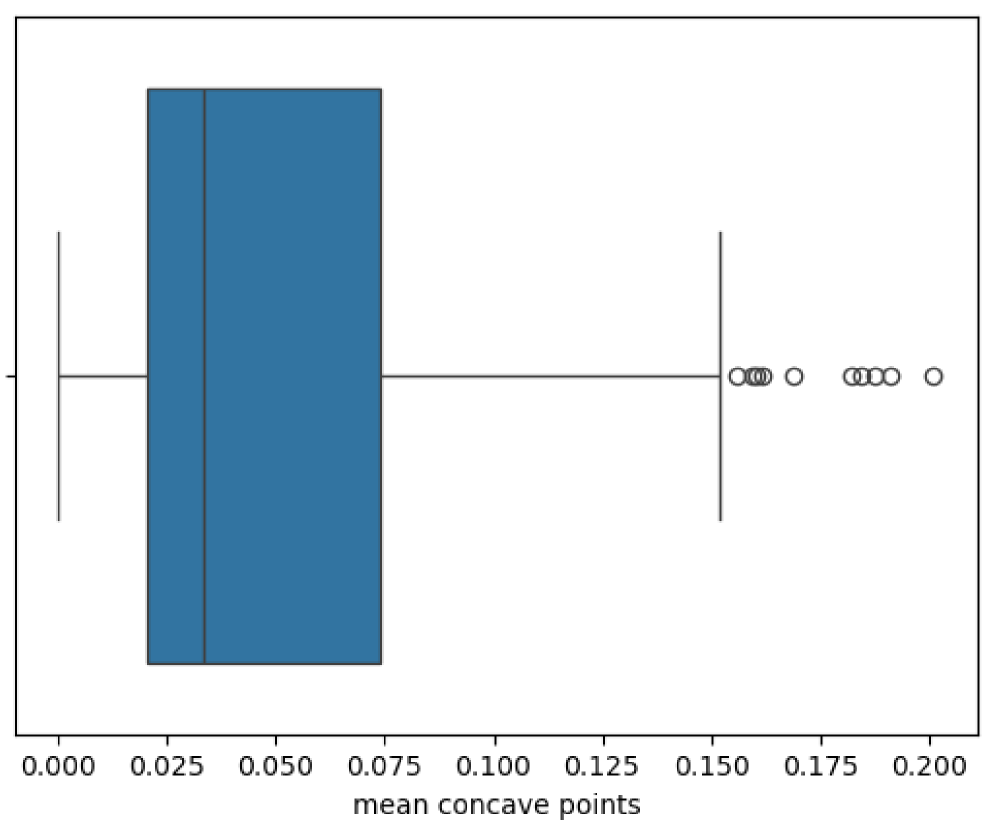


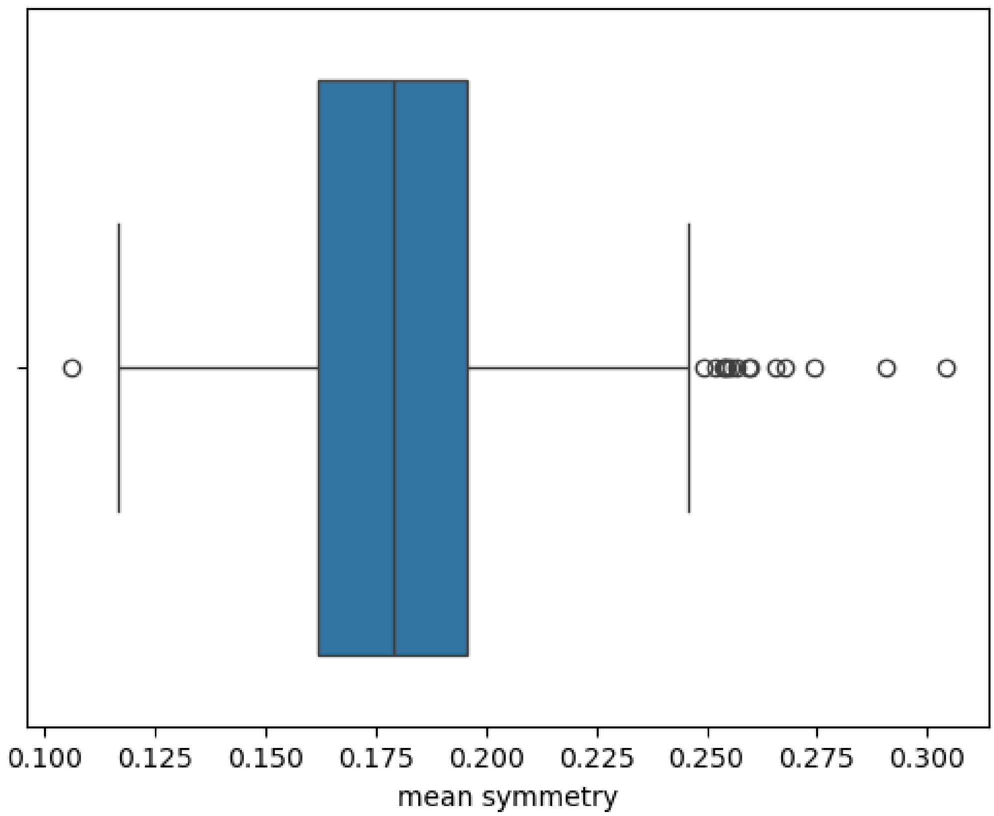
0.06
0.08
0.10
0.12
0.14
0.16
mean smoothness

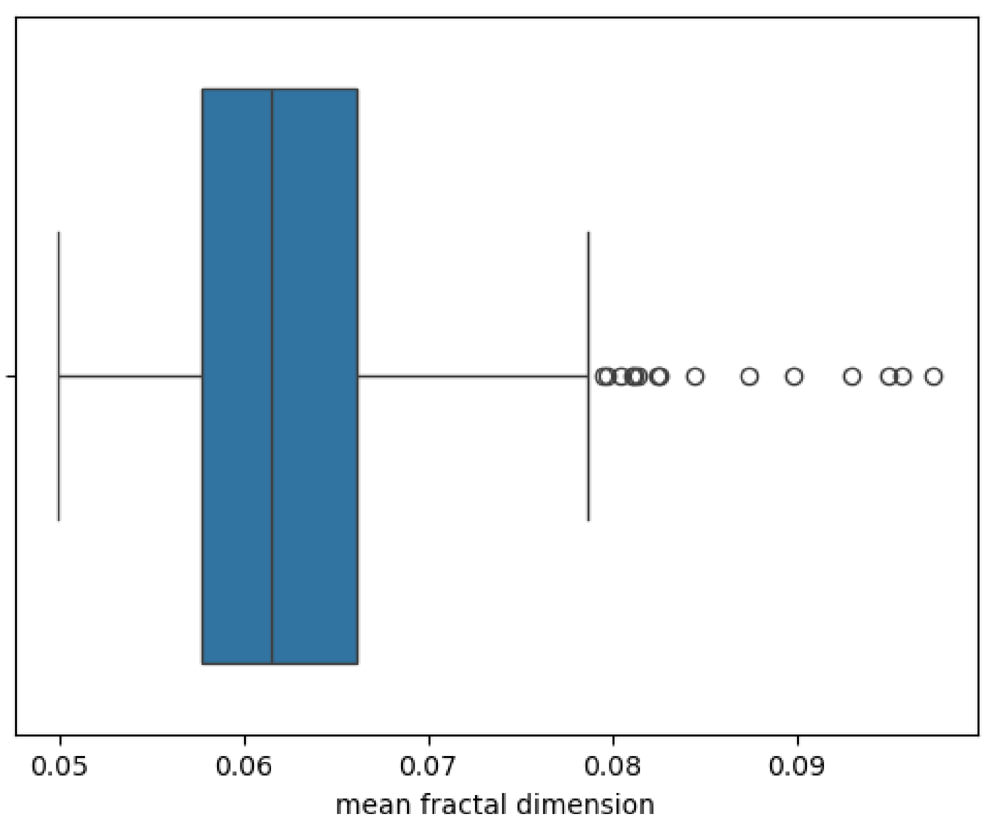


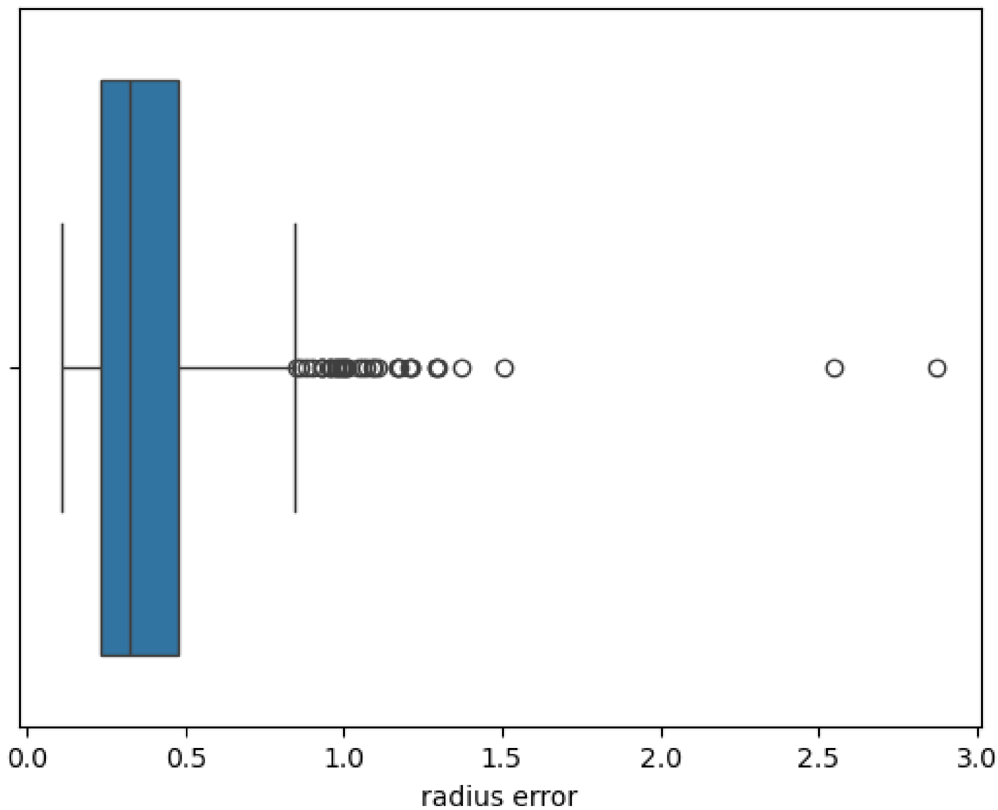
0.05
0.10
0.15
0.20
0.25
0.30
0.35
mean compactness

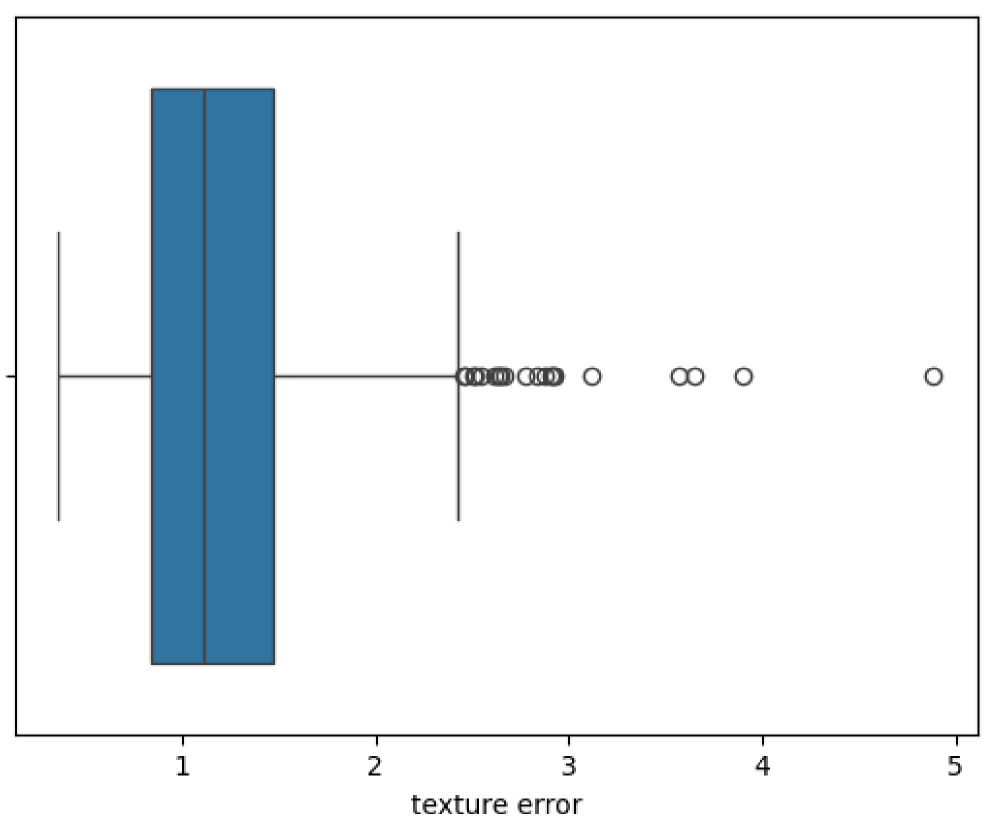

0.0
0.1
0.2
0.3
0.4
mean concavity

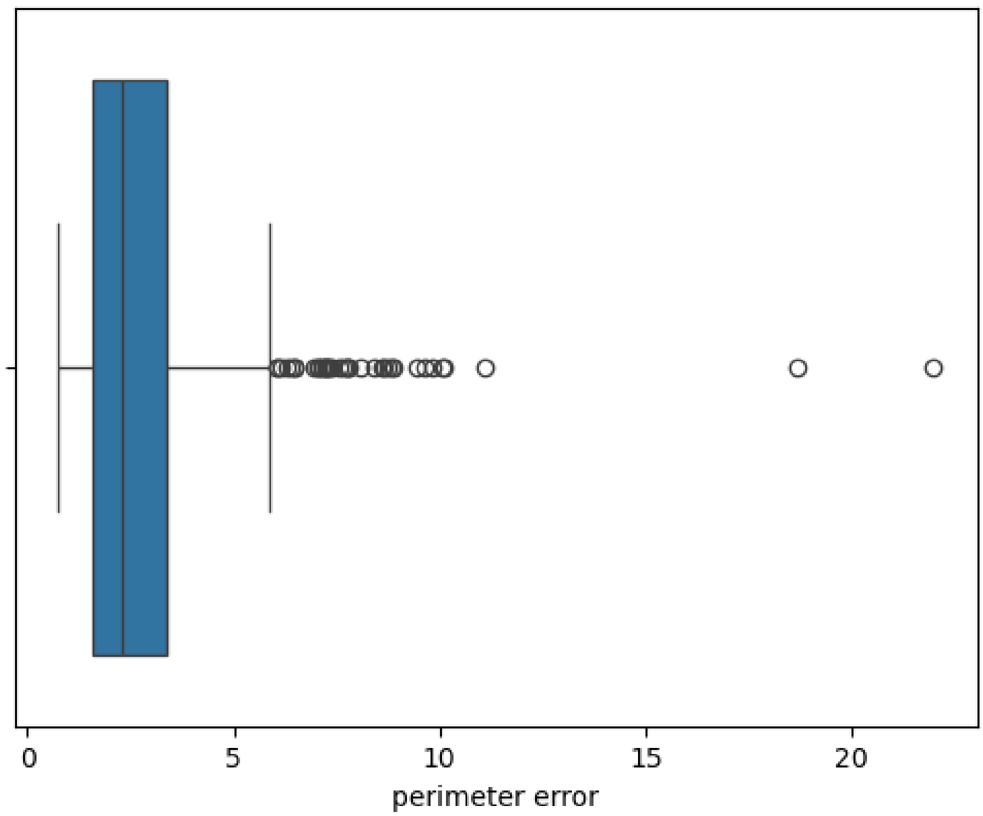

0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
mean concave points

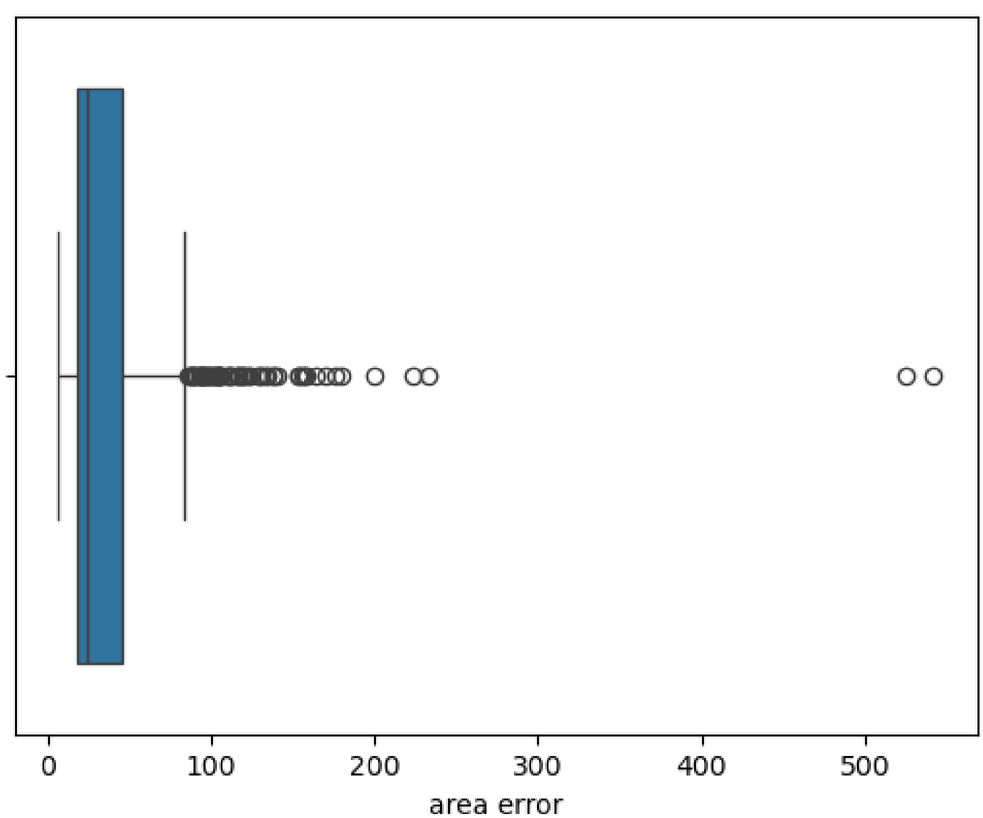

0.100
0.125
0.150
0.175
0.200
0.225
0.250
0.275
0.300
mean symmetry

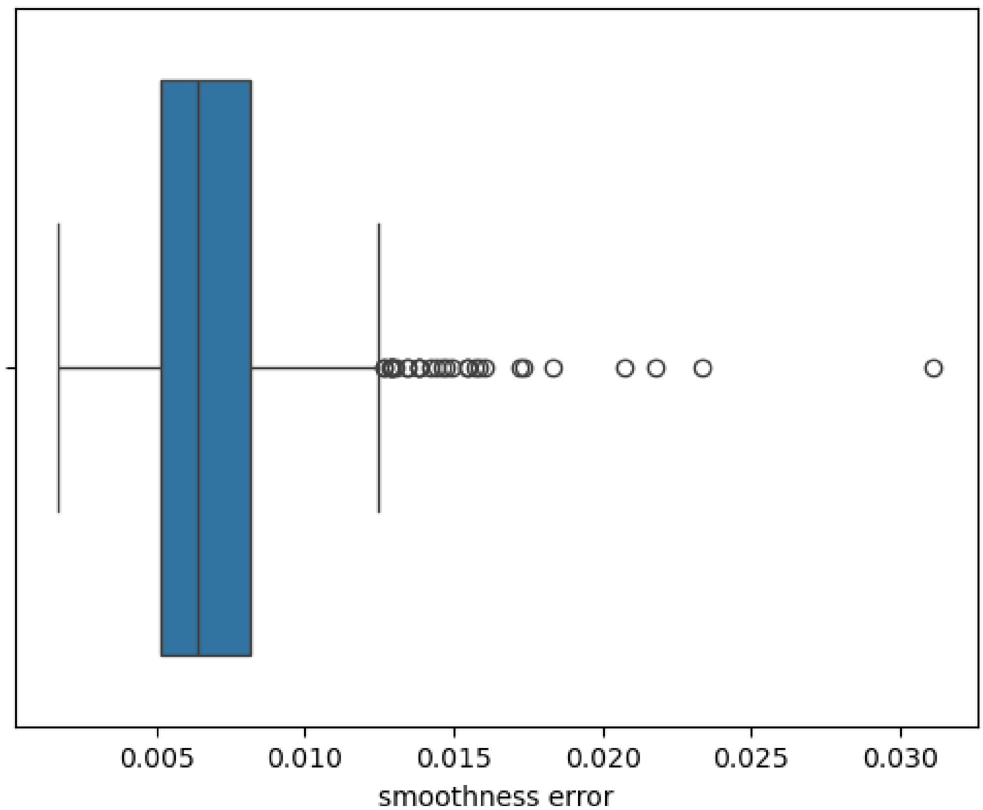

0.05
0.06
0.07
0.08
0.09
mean fractal dimension

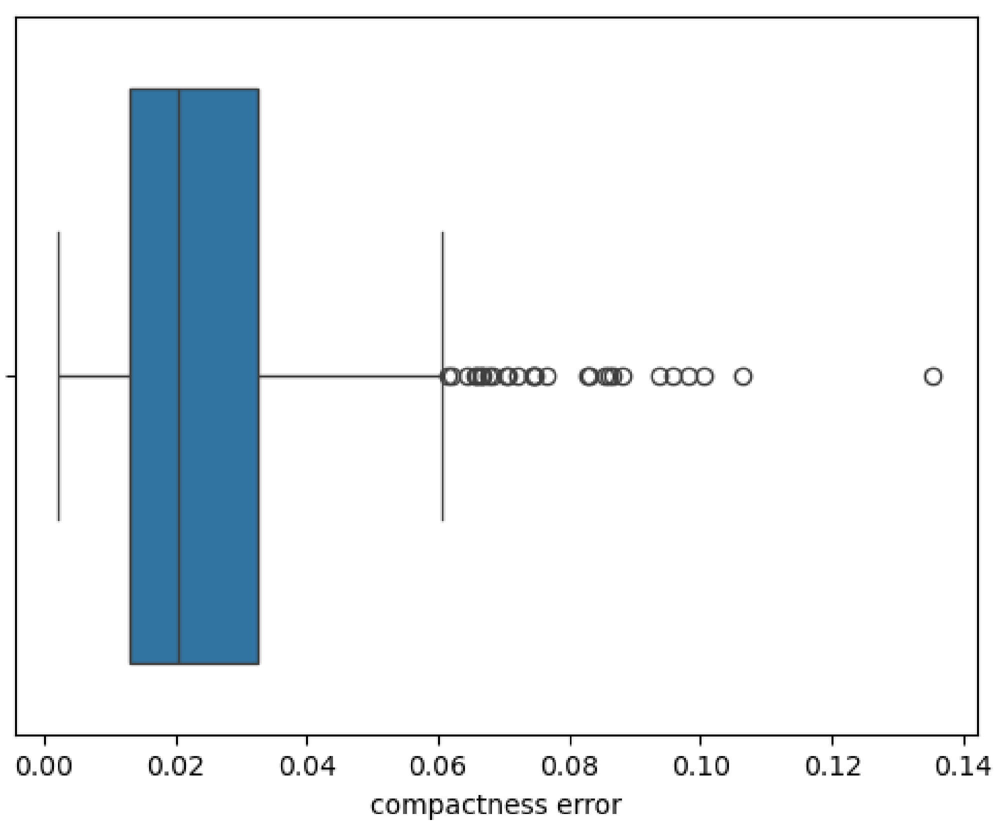

0.0
0.5
1.0
1.5
2.0
2.5
3.0
radius error

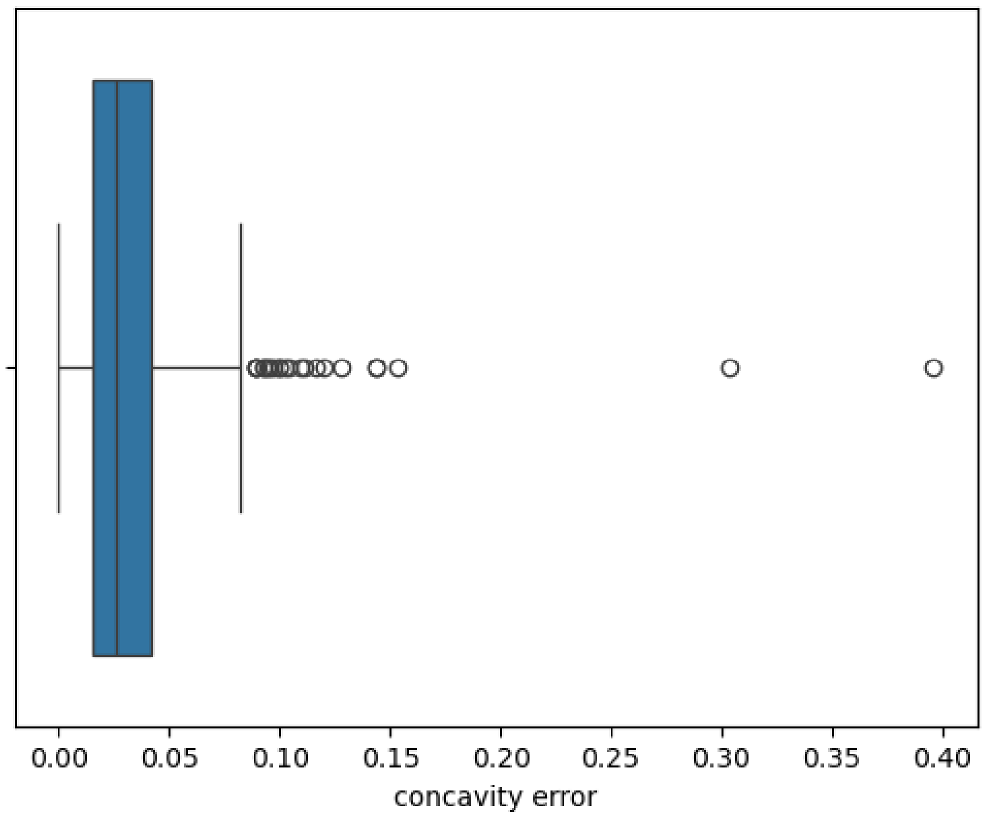

1
2
3
4
5
texture error

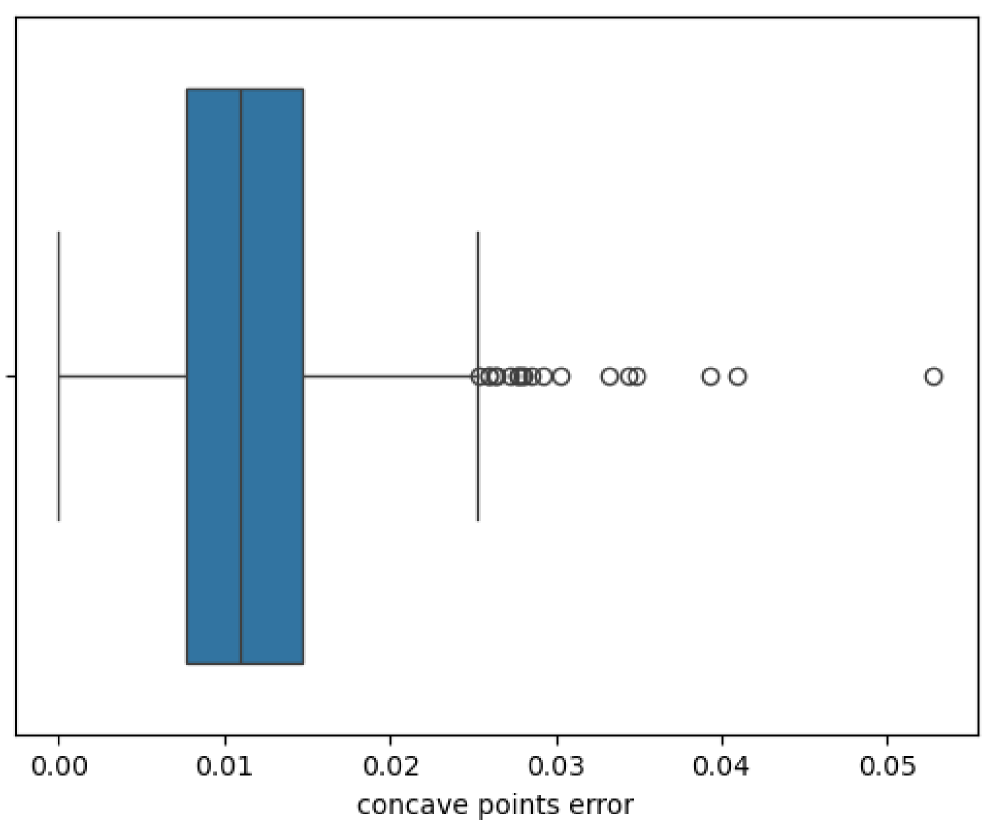

0
5
10
15
20
perimeter error

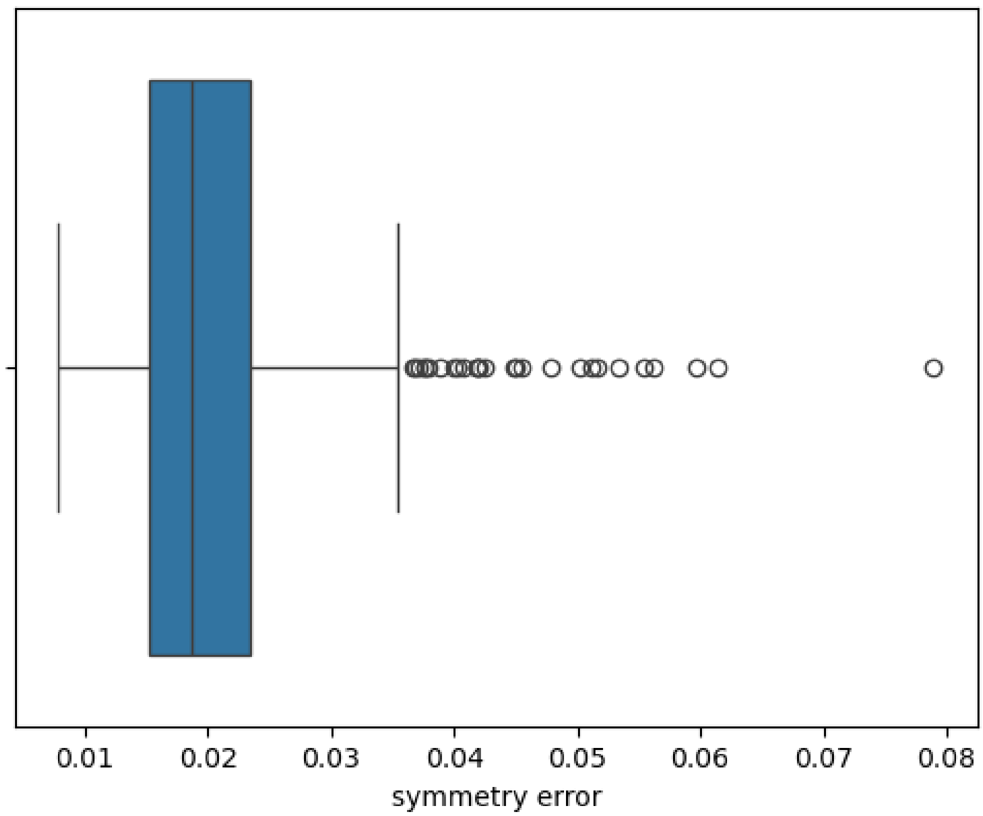

0
100
200
300
400
500
area error

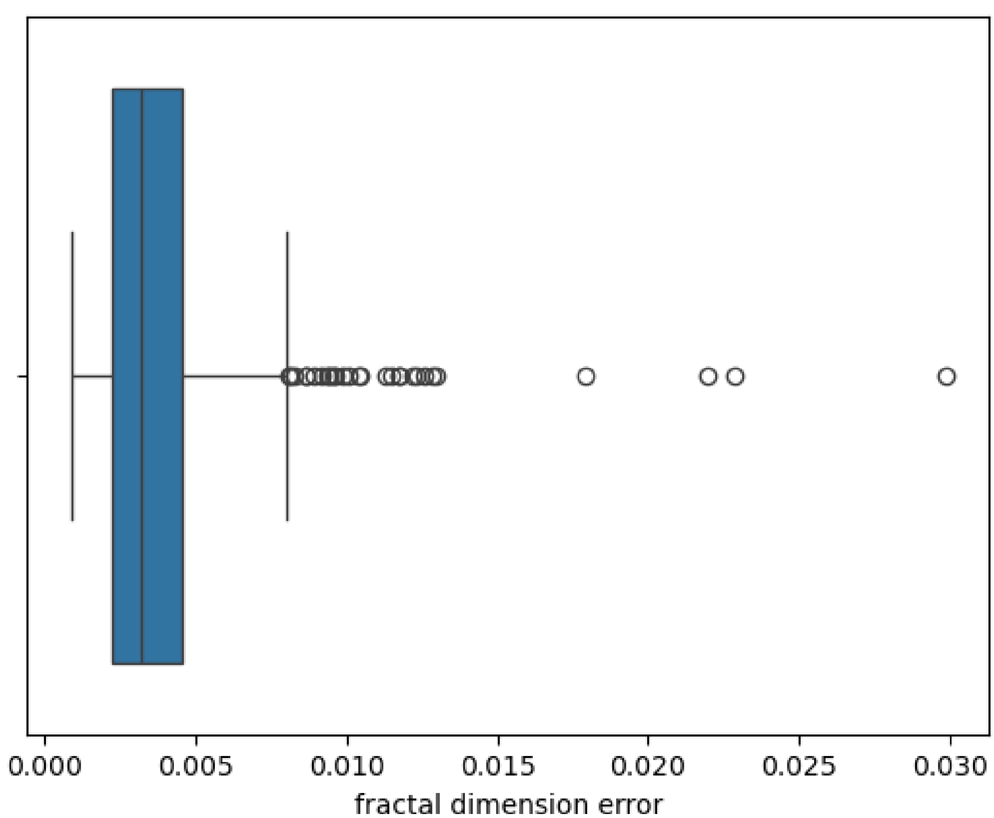

0.005
0.010
0.015
0.020
0.025
0.030
smoothness error

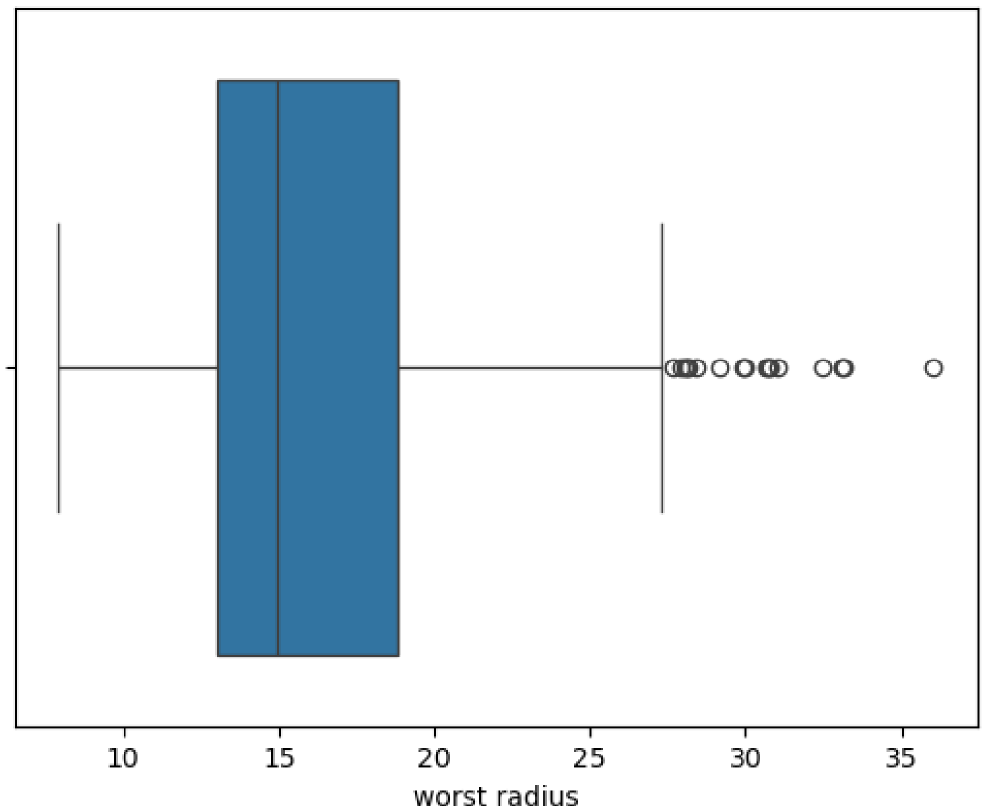

0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
compactness error

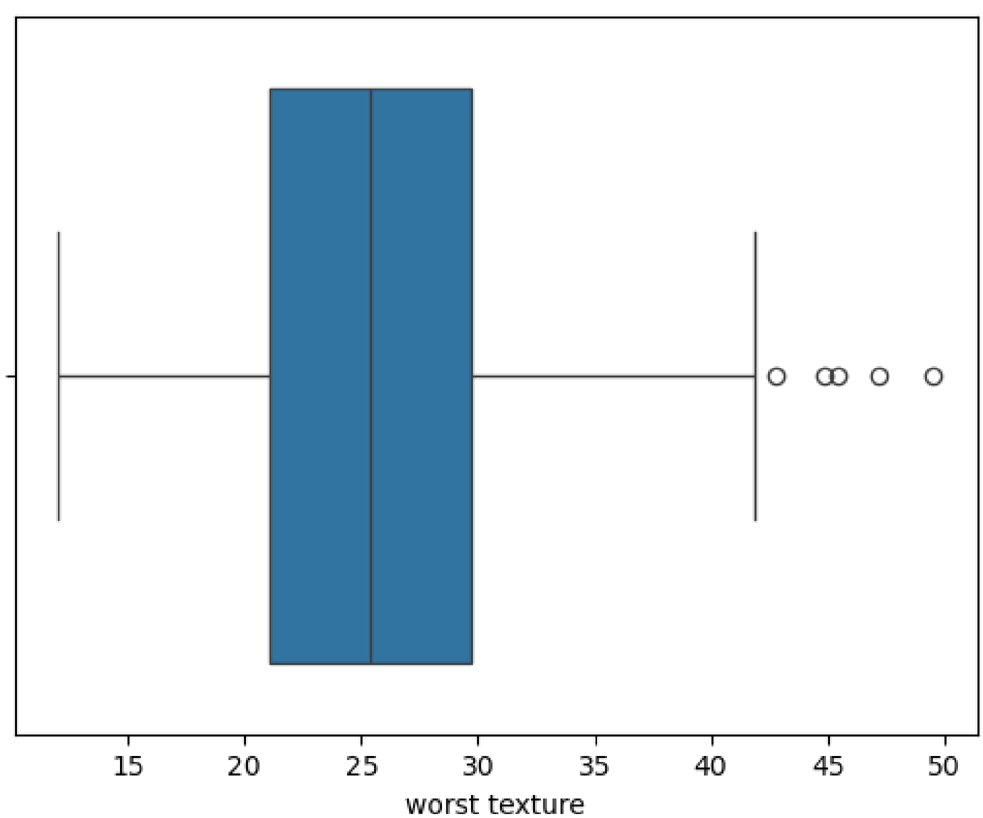

0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
concavity error

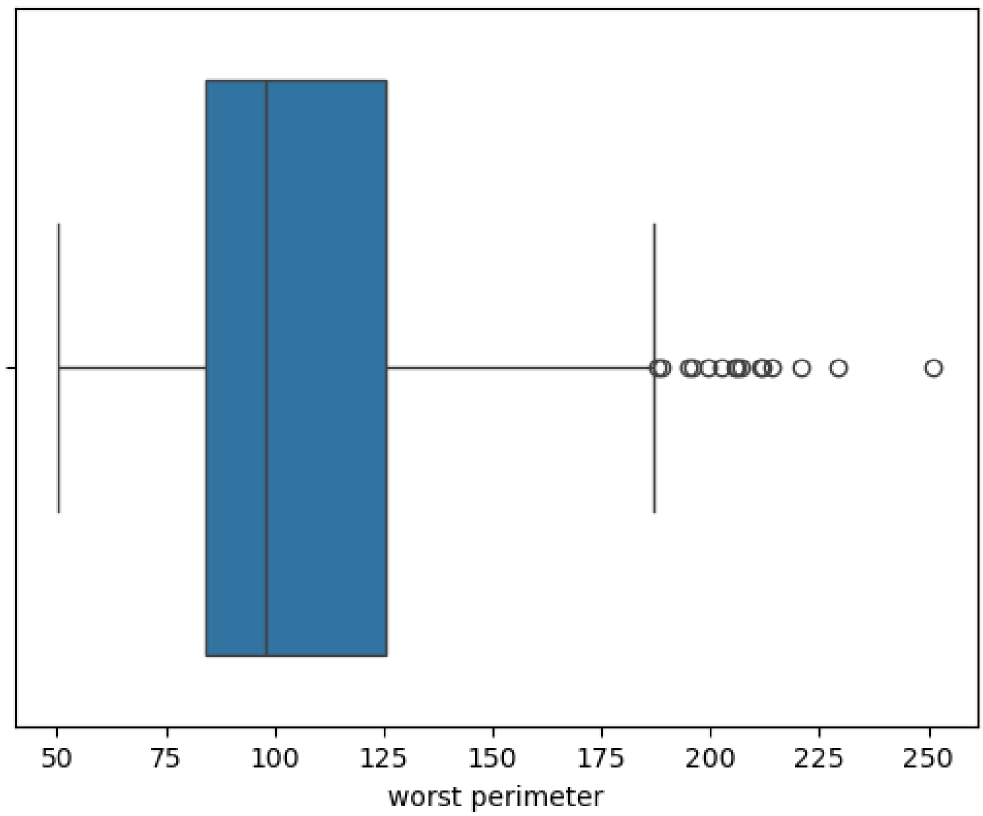

0.00
0.01
0.02
0.03
0.04
0.05
concave points error

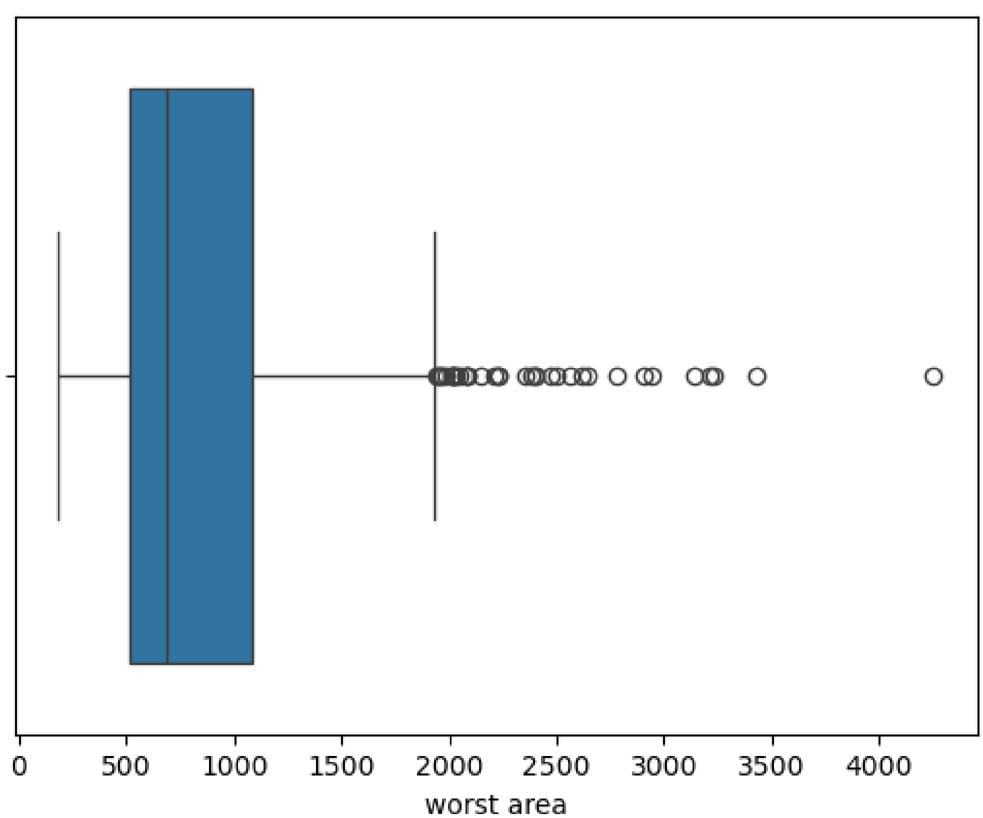

0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
symmetry error

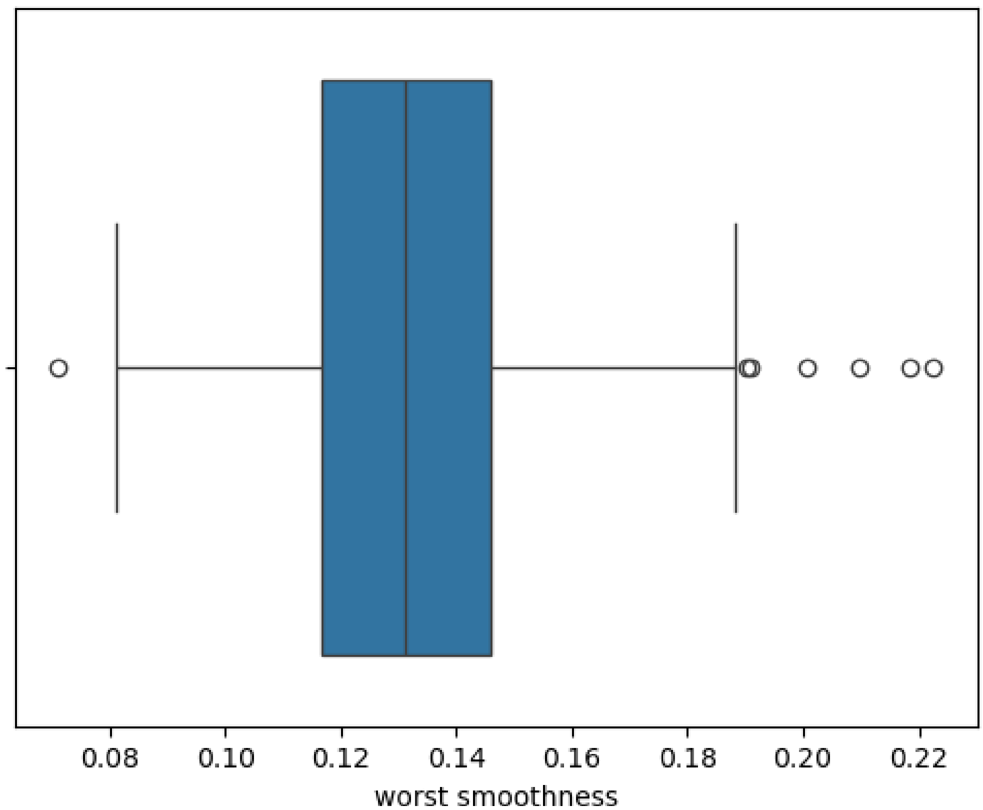

0.000
0.005
0.010
0.015
0.020
0.025
0.030
fractal dimension error

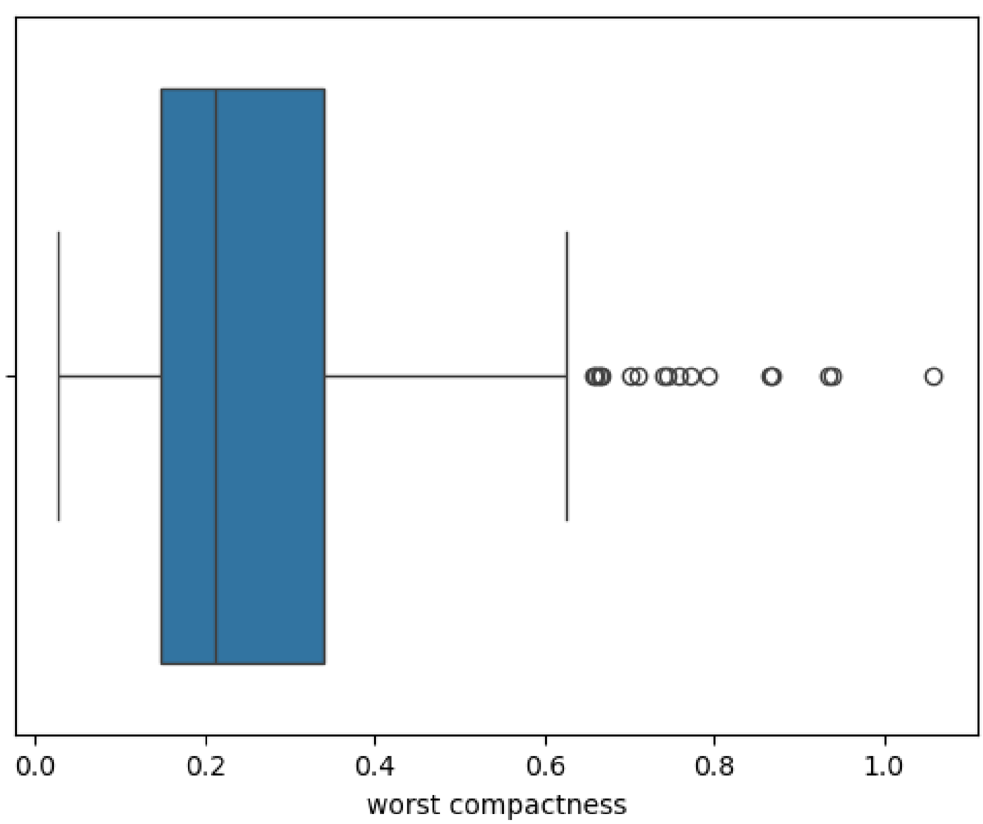

10
15
20
25
30
35
worst radius

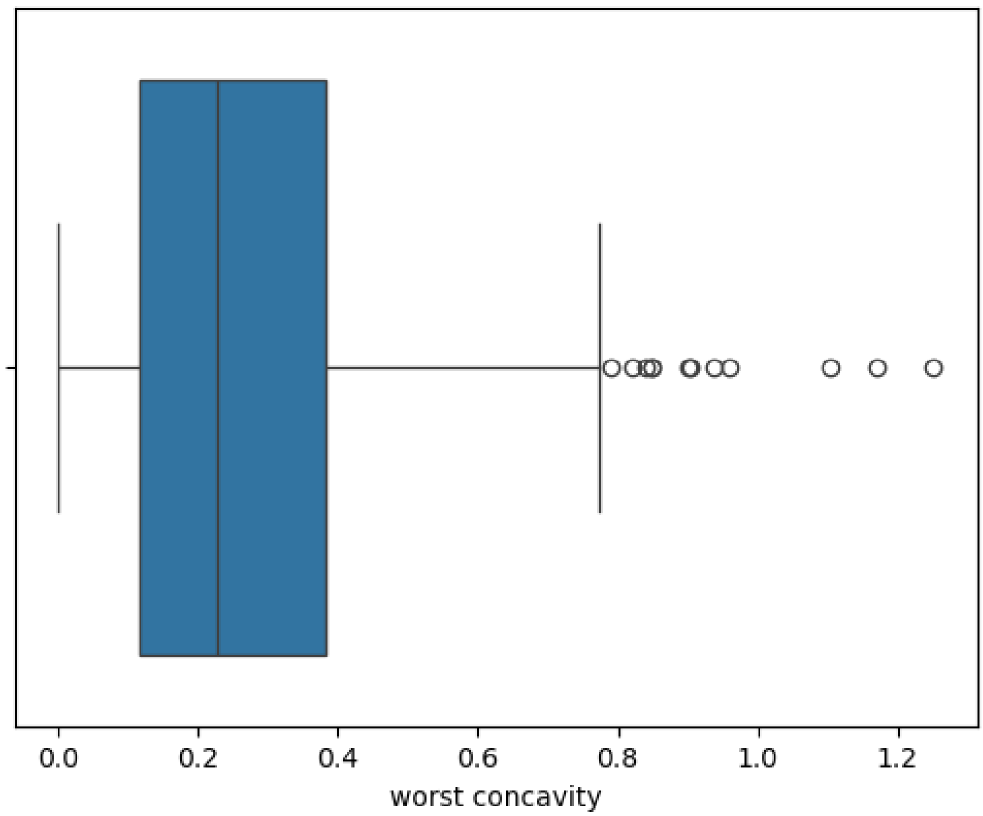

15
20
25
30
35
40
45
50
worst texture

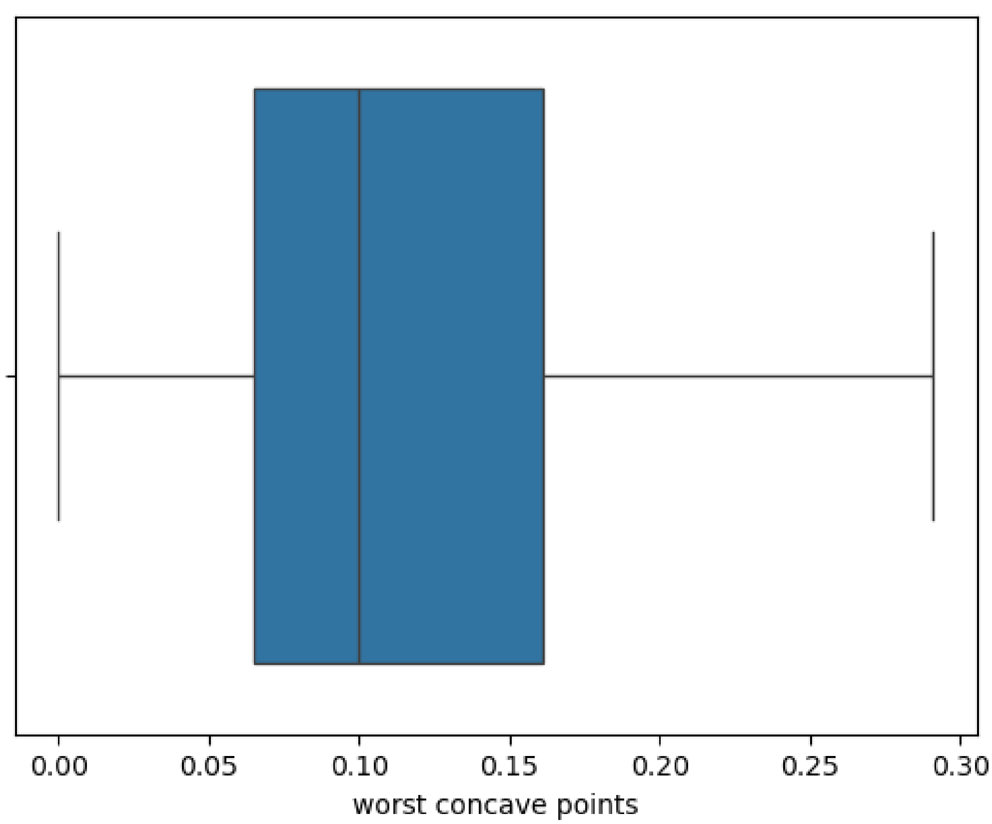

50
75
100
125
150
175
200
225
250
worst perimeter

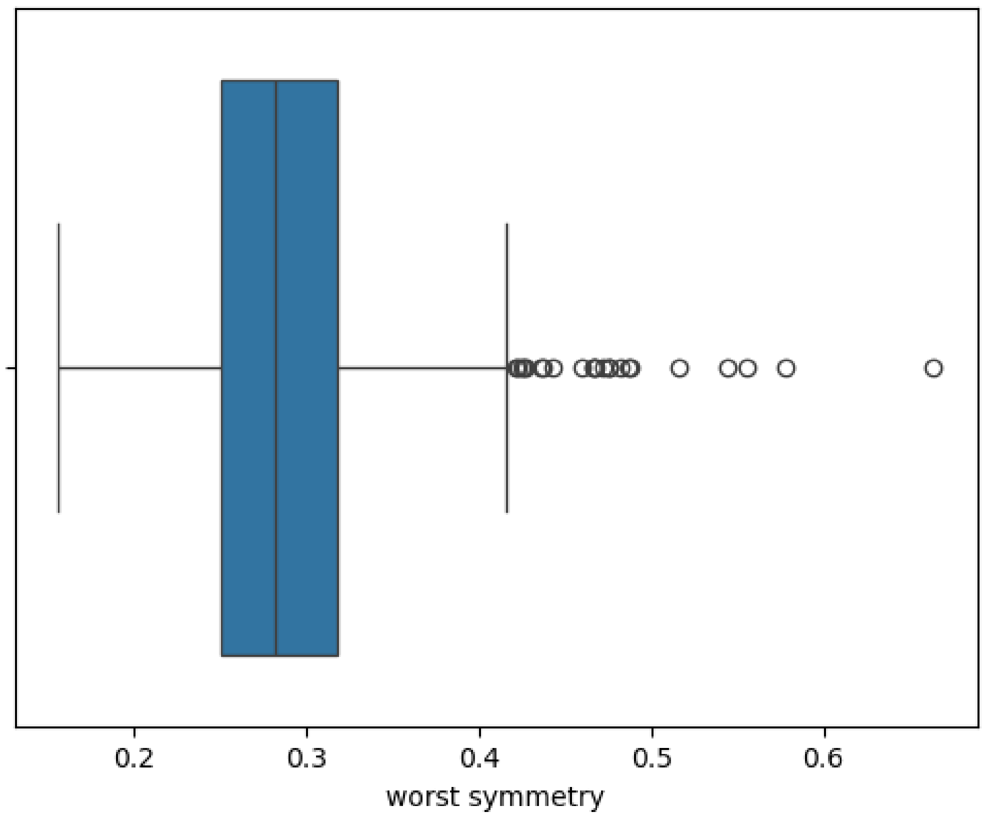

0
500
1000
1500
2000
2500
3000
3500
4000
worst area

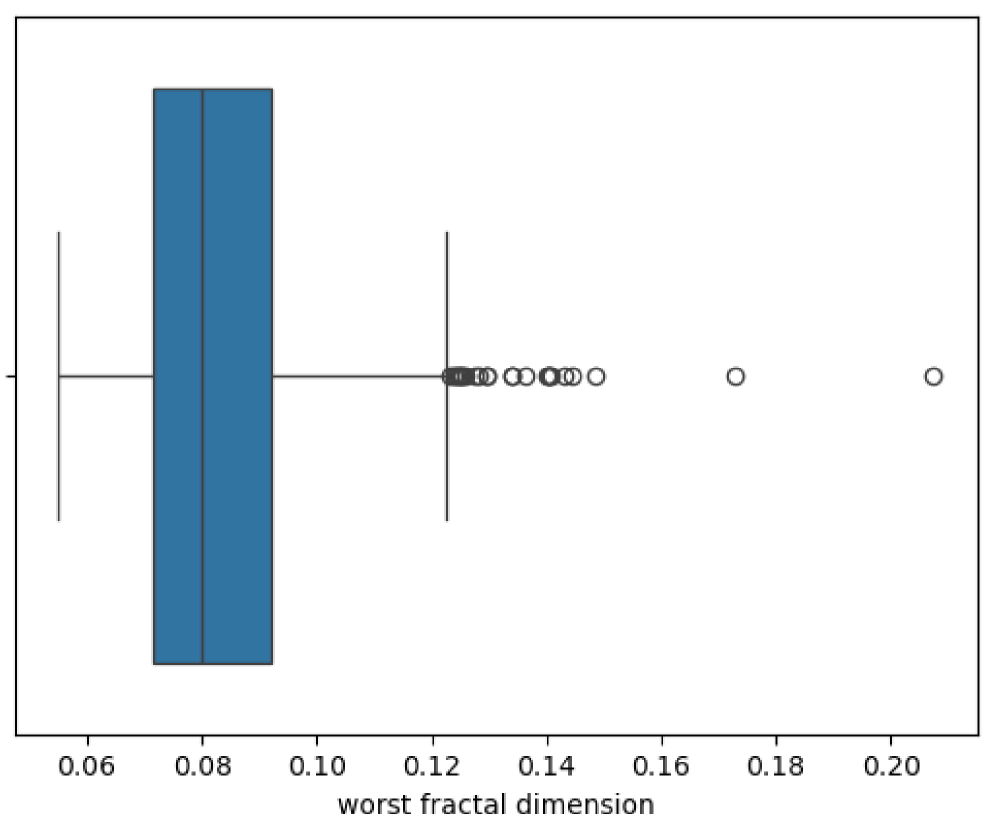

0.08
0.10
0.12
0.14
0.16
0.18
0.20
0.22
worst smoothness

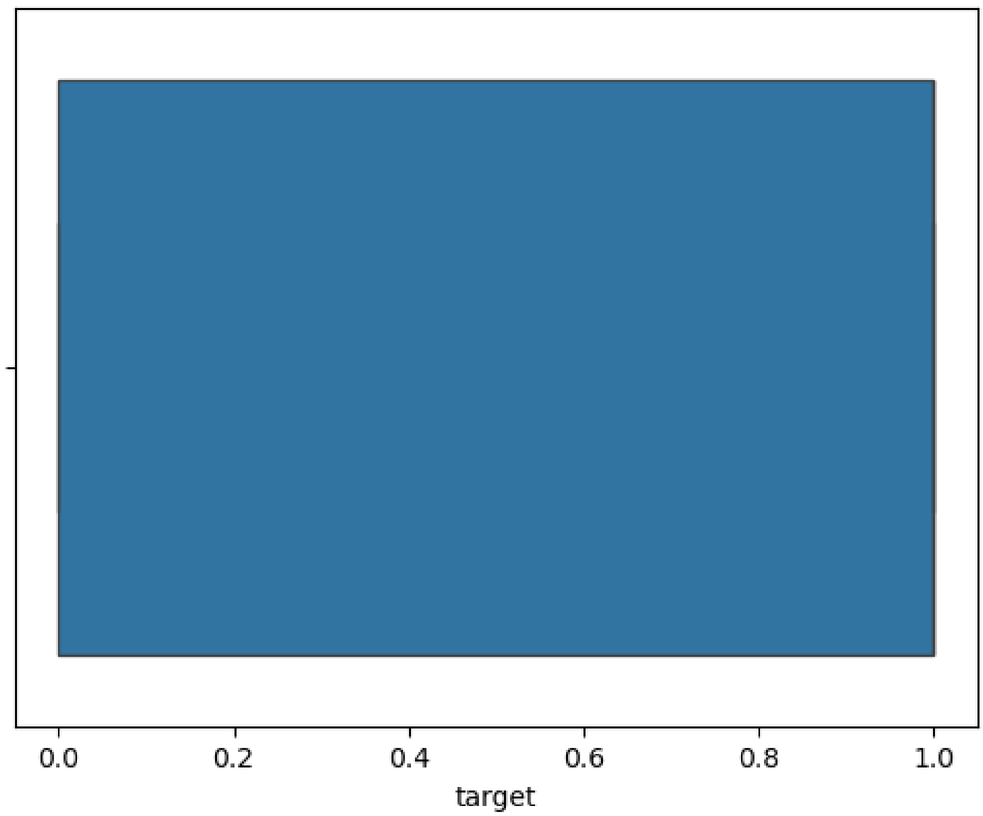

0.0
0.2
0.4
0.6
0.8
1.0
worst compactness


0.0
0.2
0.4
0.6
0.8
1.0
1.2
worst concavity


0.00
0.05
0.10
0.15
0.20
0.25
0.30
worst concave points


0.2
0.3
0.4
0.5
0.6
worst symmetry

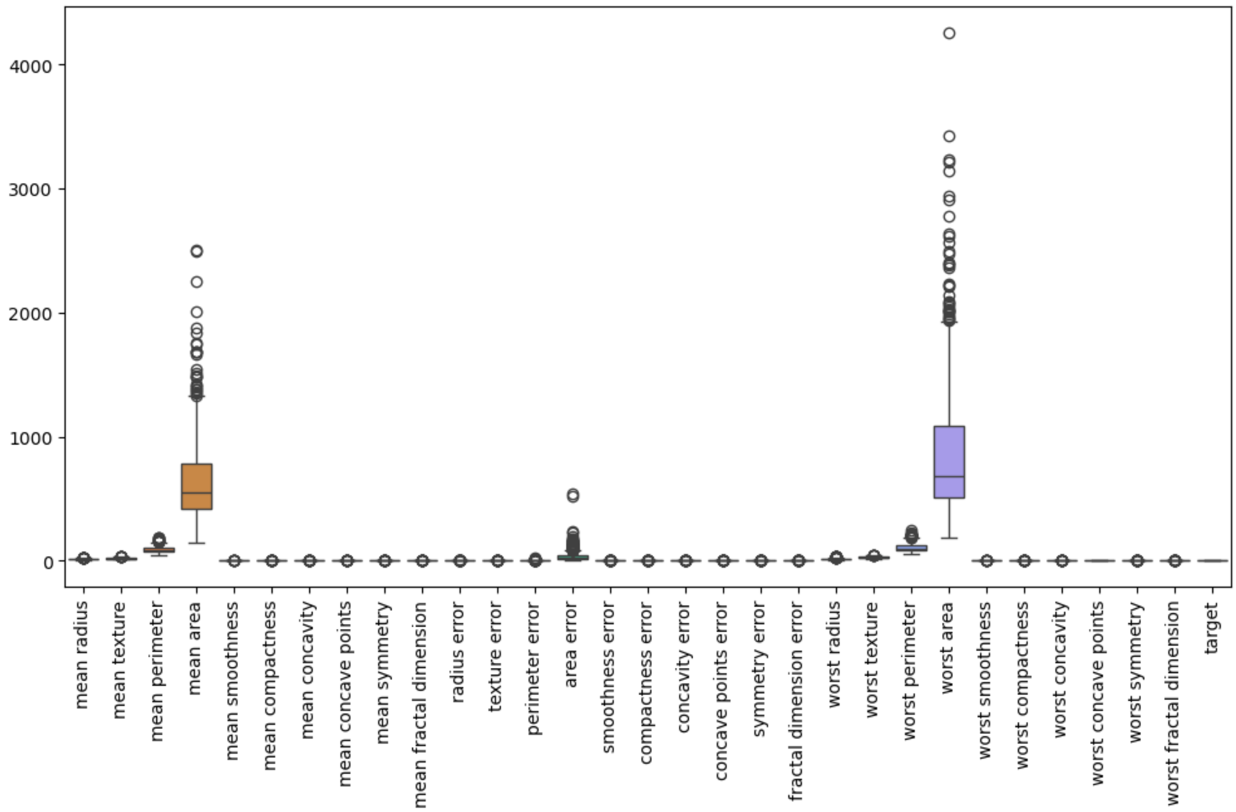

0.06
0.08
0.10
0.12
0.14
0.16
0.18
0.20
worst fractal dimension




```
# comparing the outliers
value = df.columns
plt.figure(figsize=(12,6))
sns.boxplot(data = df[value])
plt.xticks(rotation=90)
```

```
([0,
  1,
  2,
  3,
  4,
  5,
  6,
  7,
  8,
  9,
  10,
  11,
  12,
  13,
  14,
  15,
  16,
  17,
```

```
  18,
  19,
  20,
  21,
  22,
  23,
  24,
  25,
  26,
  27,
  28,
  29,
  30],
 [Text(0, 0, 'mean radius'),
  Text(1, 0, 'mean texture'),
  Text(2, 0, 'mean perimeter'),
  Text(3, 0, 'mean area'),
  Text(4, 0, 'mean smoothness'),
  Text(5, 0, 'mean compactness'),
  Text(6, 0, 'mean concavity'),
  Text(7, 0, 'mean concave points'),
  Text(8, 0, 'mean symmetry'),
  Text(9, 0, 'mean fractal dimension'),
  Text(10, 0, 'radius error'),
  Text(11, 0, 'texture error'),
  Text(12, 0, 'perimeter error'),
  Text(13, 0, 'area error'),
  Text(14, 0, 'smoothness error'),
  Text(15, 0, 'compactness error'),
  Text(16, 0, 'concavity error'),
  Text(17, 0, 'concave points error'),
  Text(18, 0, 'symmetry error'),
  Text(19, 0, 'fractal dimension error'),
  Text(20, 0, 'worst radius'),
  Text(21, 0, 'worst texture'),
  Text(22, 0, 'worst perimeter'),
  Text(23, 0, 'worst area'),
  Text(24, 0, 'worst smoothness'),
  Text(25, 0, 'worst compactness'),
  Text(26, 0, 'worst concavity'),
  Text(27, 0, 'worst concave points'),
  Text(28, 0, 'worst symmetry'),
  Text(29, 0, 'worst fractal dimension'),
  Text(30, 0, 'target')])
```




```
## Using IQR method to find outliers and capping it
columns_to_process = [col for col in df.columns if col != 'target']
#target doesn't have outliers

# Fix outliers using the IQR method
for col in columns_to_process:
    Q1 = df[col].quantile(0.25)  # First quartile
    Q3 = df[col].quantile(0.75)  # Third quartile
    IQR = Q3 - Q1                # Interquartile range

    lower_limit = Q1 - 1.5 * IQR
    upper_limit = Q3 + 1.5 * IQR

    # Clip values outside the bounds
    df[col] = df[col].clip(lower=lower_limit, upper=upper_limit)

df.head()
```

```
   mean radius  mean texture  mean perimeter  mean area  mean
smoothness  \
0        17.99         10.38          122.80     1001.0
0.118400
1        20.57         17.77          132.90     1326.0
0.084740
2        19.69         21.25          130.00     1203.0
```

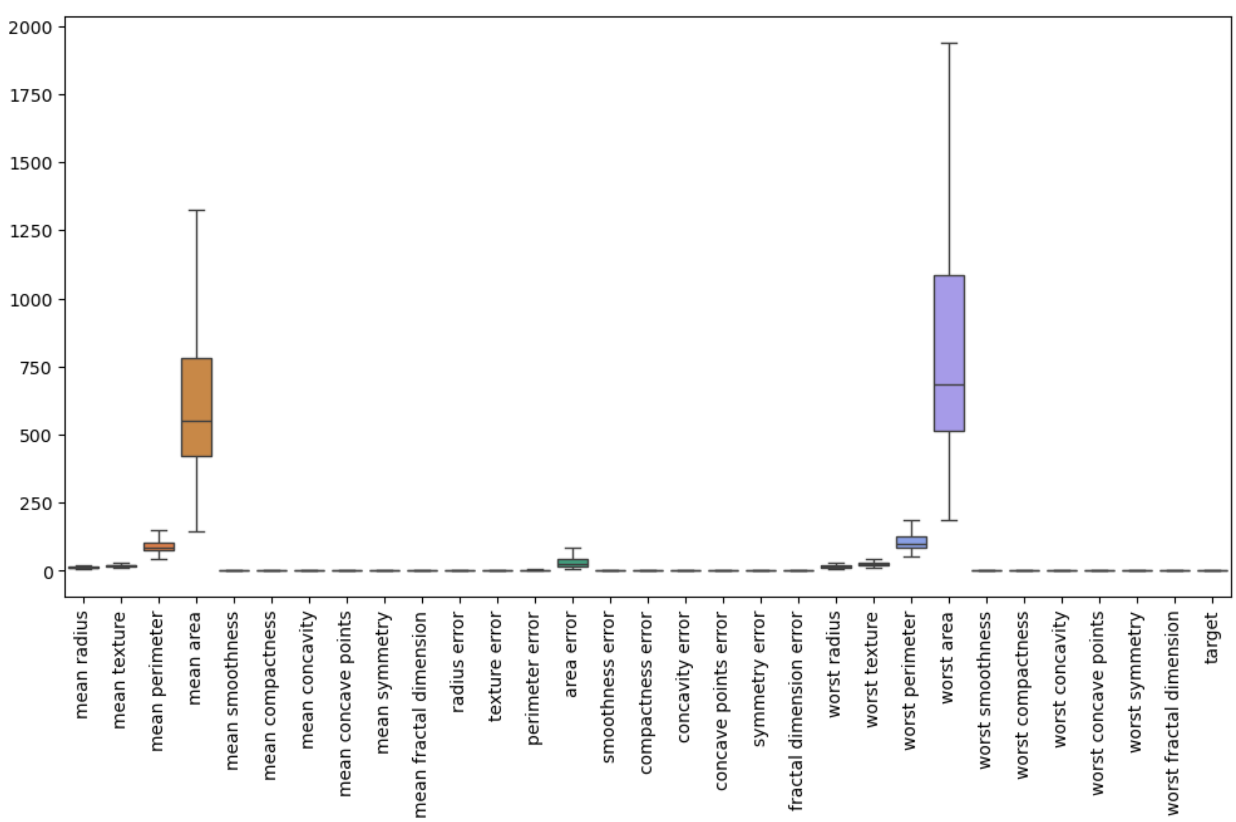
```
0.109600
3        11.42         20.38           77.58      386.1
0.133695
4        20.29         14.34          135.10     1297.0
0.100300

   mean compactness  mean concavity  mean concave points  mean
symmetry  \
0           0.22862         0.28241              0.14710
0.2419
1           0.07864         0.08690              0.07017
0.1812
2           0.15990         0.19740              0.12790
0.2069
3           0.22862         0.24140              0.10520
0.2464
4           0.13280         0.19800              0.10430
0.1809

   mean fractal dimension  ...  worst texture  worst perimeter  worst
area  \
0                 0.07871  ...          17.33           184.60
1937.05
1                 0.05667  ...          23.41           158.80
1937.05
2                 0.05999  ...          25.53           152.50
1709.00
3                 0.07875  ...          26.50            98.87
567.70
4                 0.05883  ...          16.67           152.20
1575.00

   worst smoothness  worst compactness  worst concavity  worst concave
points  \
0            0.1622            0.62695           0.7119
0.2654
1            0.1238            0.18660           0.2416
0.1860
2            0.1444            0.42450           0.4504
0.2430
3            0.1901            0.62695           0.6869
0.2575
4            0.1374            0.20500           0.4000
0.1625

   worst symmetry  worst fractal dimension  target
0         0.41915                  0.11890       0
1         0.27500                  0.08902       0
2         0.36130                  0.08758       0
3         0.41915                  0.12301       0
```

```
4         0.23640                        0.07678       0

[5 rows x 31 columns]
```

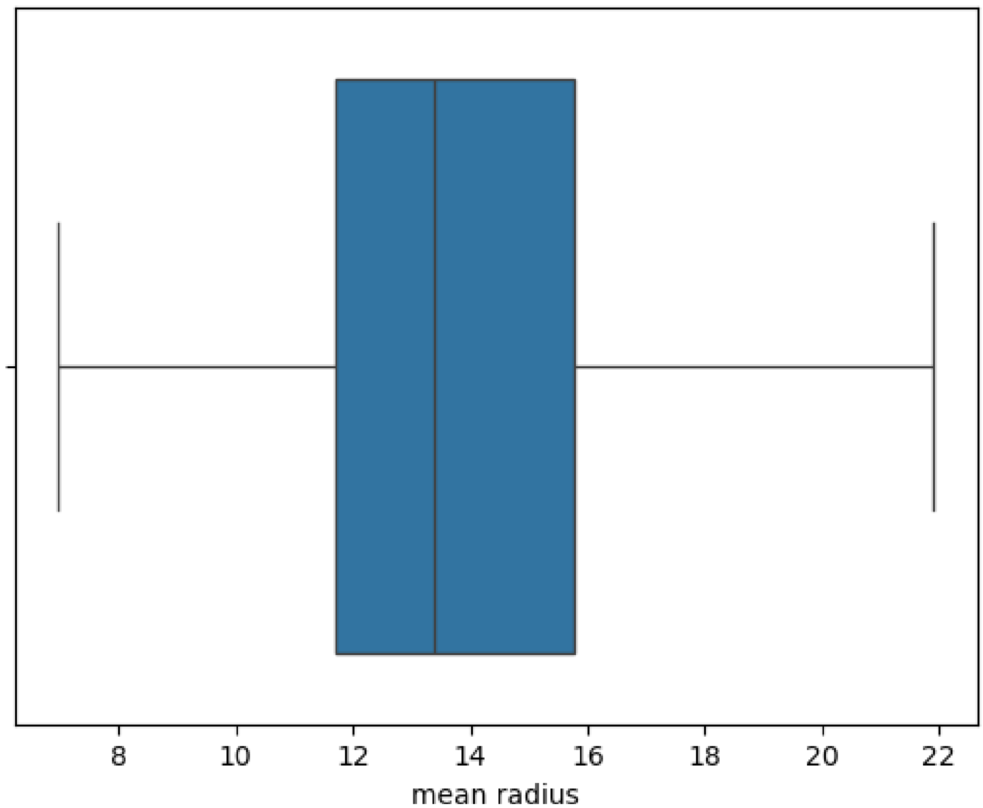
```
# visualising outliers after fixing
value = df.columns
plt.figure(figsize=(12,6))
sns.boxplot(data = df[value])
plt.xticks(rotation=90)
```

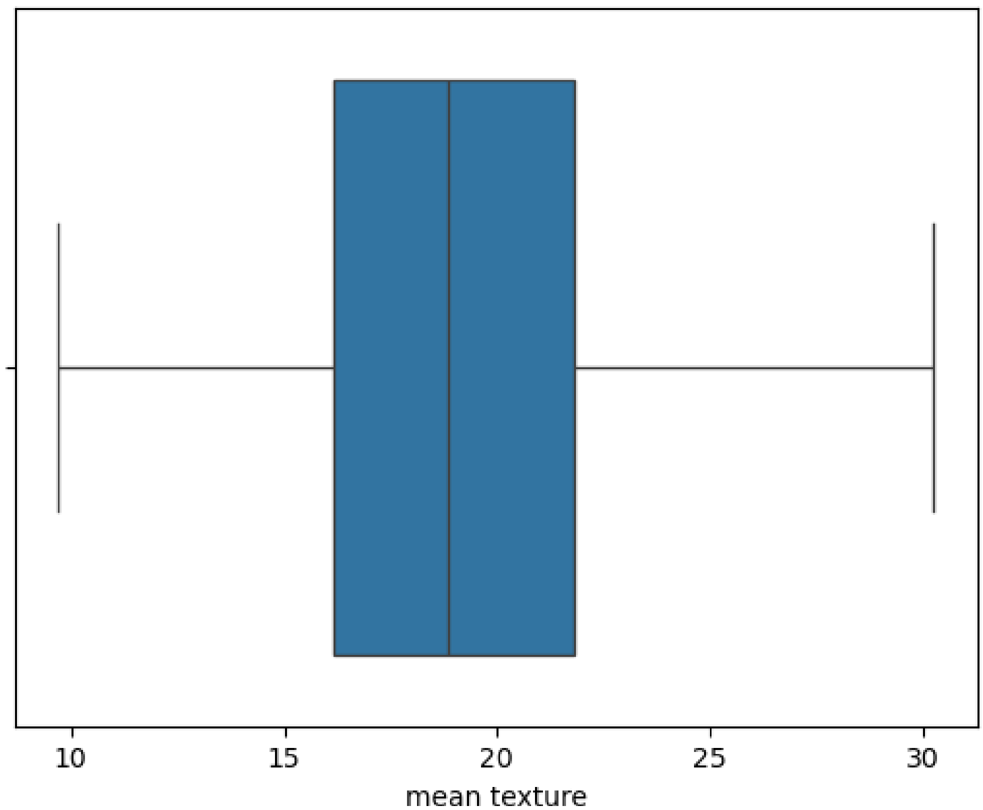
```
([0,
  1,
  2,
  3,
  4,
  5,
  6,
  7,
  8,
  9,
  10,
  11,
  12,
  13,
  14,
  15,
  16,
  17,
  18,
  19,
  20,
  21,
  22,
  23,
  24,
  25,
  26,
  27,
  28,
  29,
  30],
 [Text(0, 0, 'mean radius'),
  Text(1, 0, 'mean texture'),
  Text(2, 0, 'mean perimeter'),
  Text(3, 0, 'mean area'),
  Text(4, 0, 'mean smoothness'),
  Text(5, 0, 'mean compactness'),
  Text(6, 0, 'mean concavity'),
  Text(7, 0, 'mean concave points'),
  Text(8, 0, 'mean symmetry'),
```

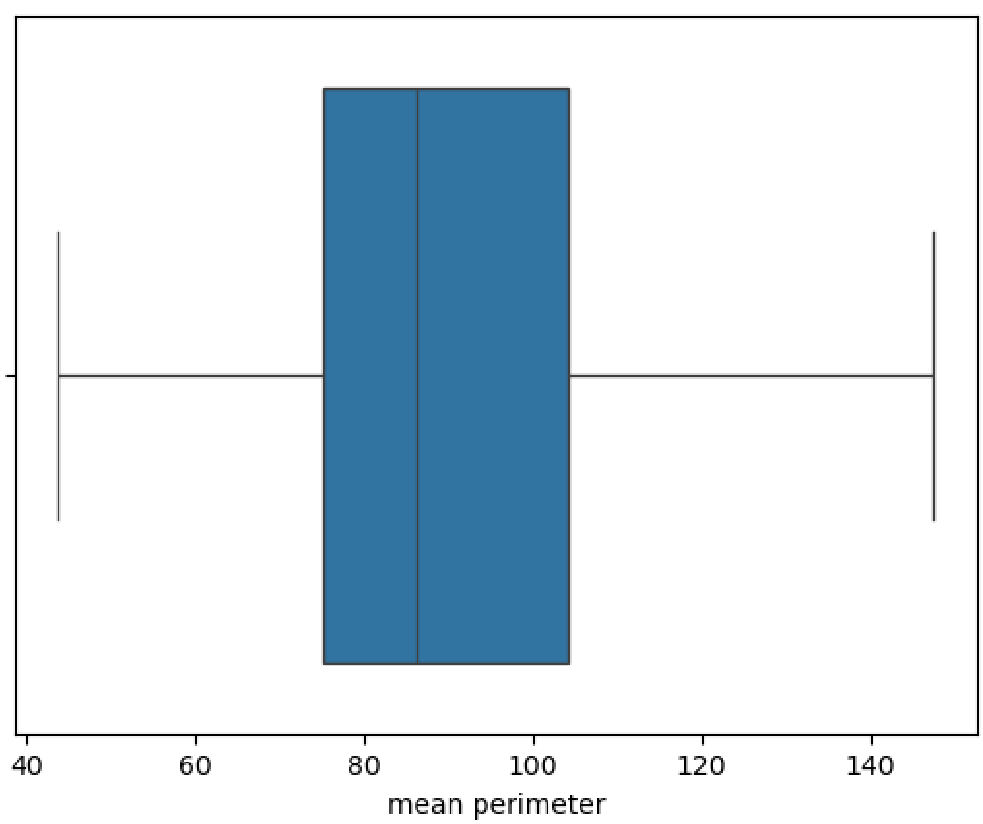
```
 Text(9, 0, 'mean fractal dimension'),
 Text(10, 0, 'radius error'),
 Text(11, 0, 'texture error'),
 Text(12, 0, 'perimeter error'),
 Text(13, 0, 'area error'),
 Text(14, 0, 'smoothness error'),
 Text(15, 0, 'compactness error'),
 Text(16, 0, 'concavity error'),
 Text(17, 0, 'concave points error'),
 Text(18, 0, 'symmetry error'),
 Text(19, 0, 'fractal dimension error'),
 Text(20, 0, 'worst radius'),
 Text(21, 0, 'worst texture'),
 Text(22, 0, 'worst perimeter'),
 Text(23, 0, 'worst area'),
 Text(24, 0, 'worst smoothness'),
 Text(25, 0, 'worst compactness'),
 Text(26, 0, 'worst concavity'),
 Text(27, 0, 'worst concave points'),
 Text(28, 0, 'worst symmetry'),
 Text(29, 0, 'worst fractal dimension'),
 Text(30, 0, 'target')])
```

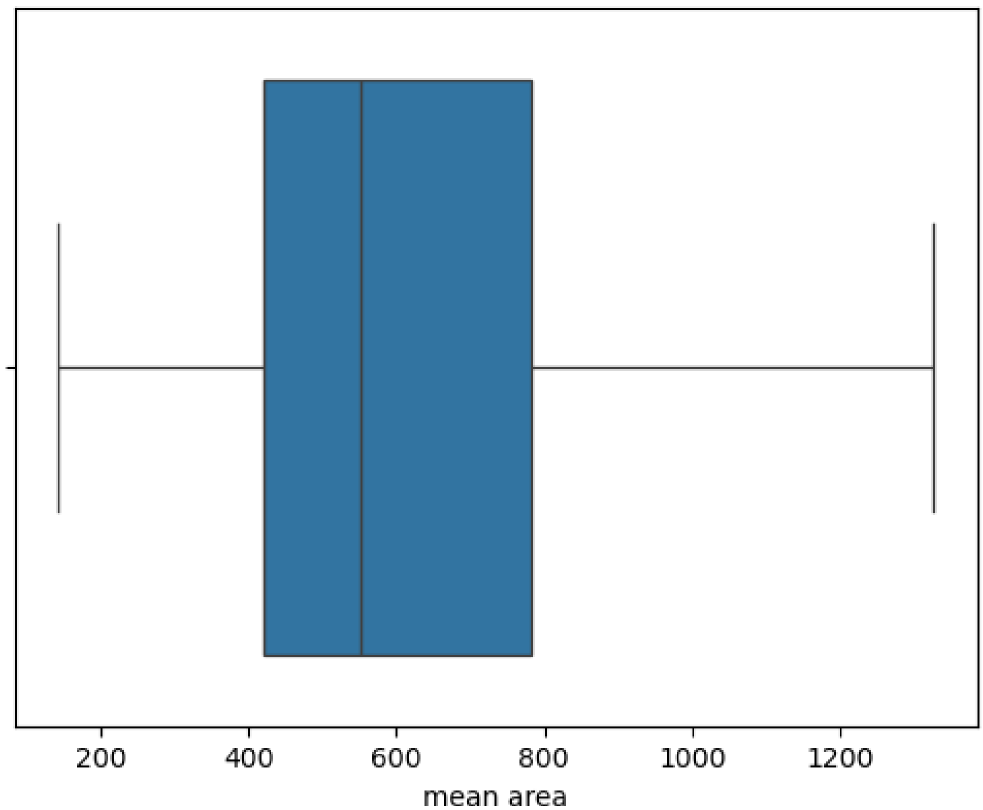


```python
# Box plot for each column after fixing outliers
for i in df.columns:
    sns.boxplot(data=df,x=i)
    plt.show()
```

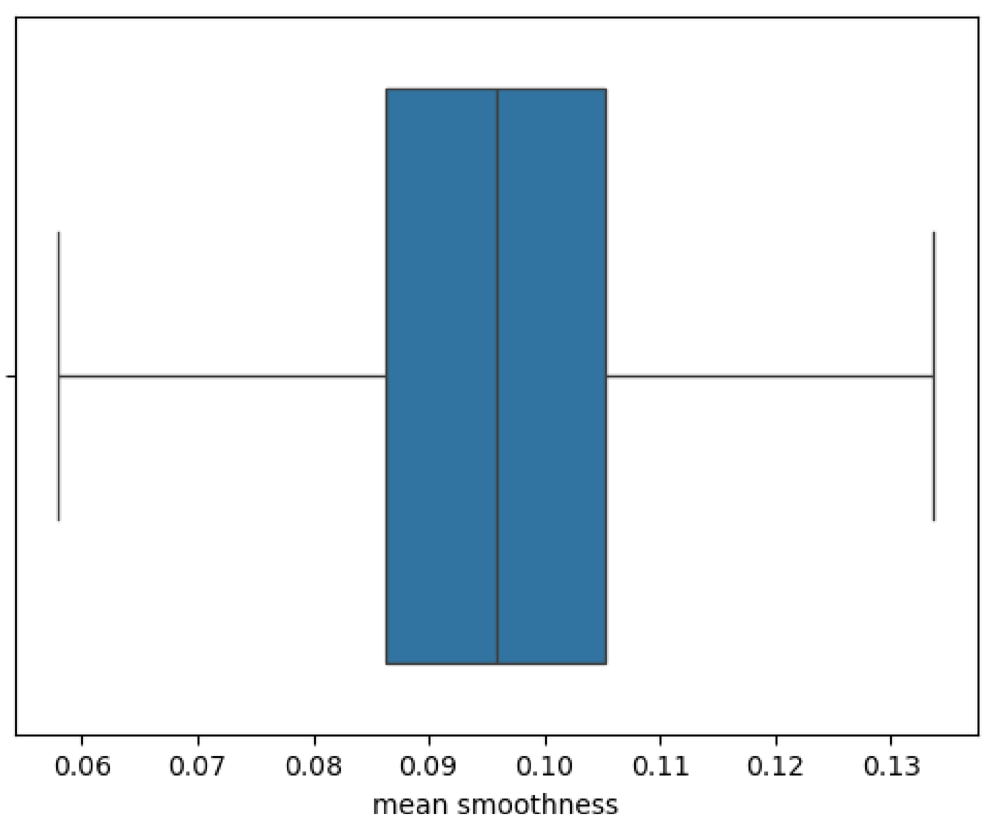



10
15
20
25
30
mean texture

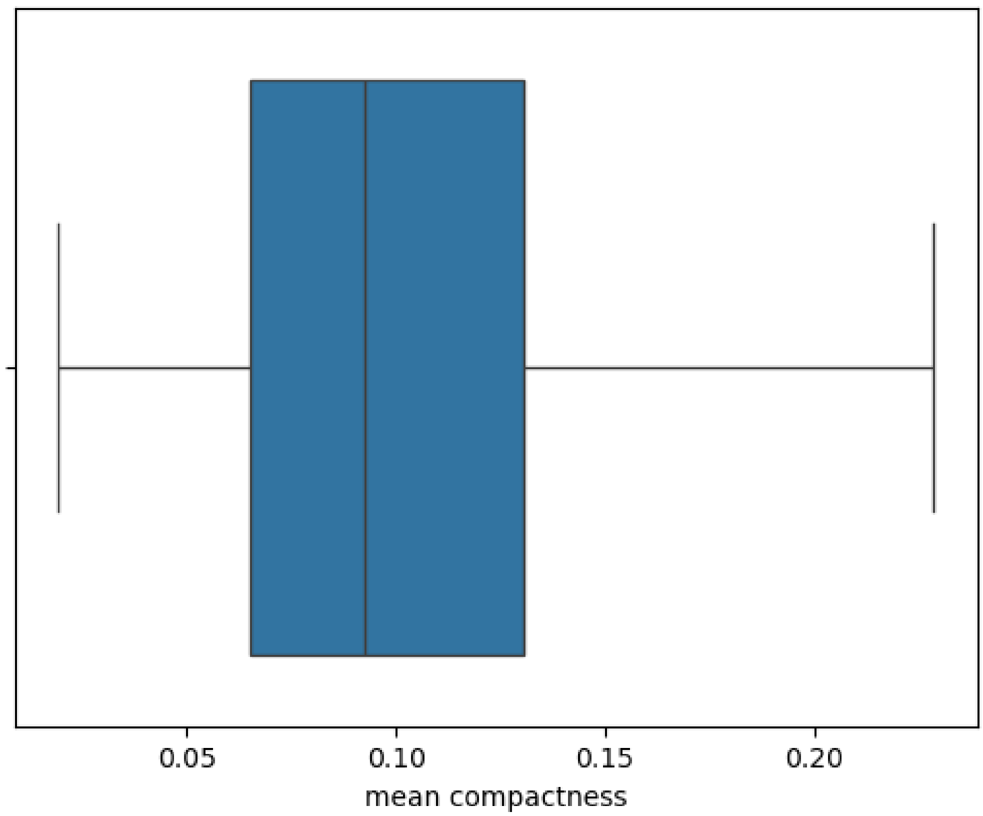

40
60
80
100
120
140
mean perimeter

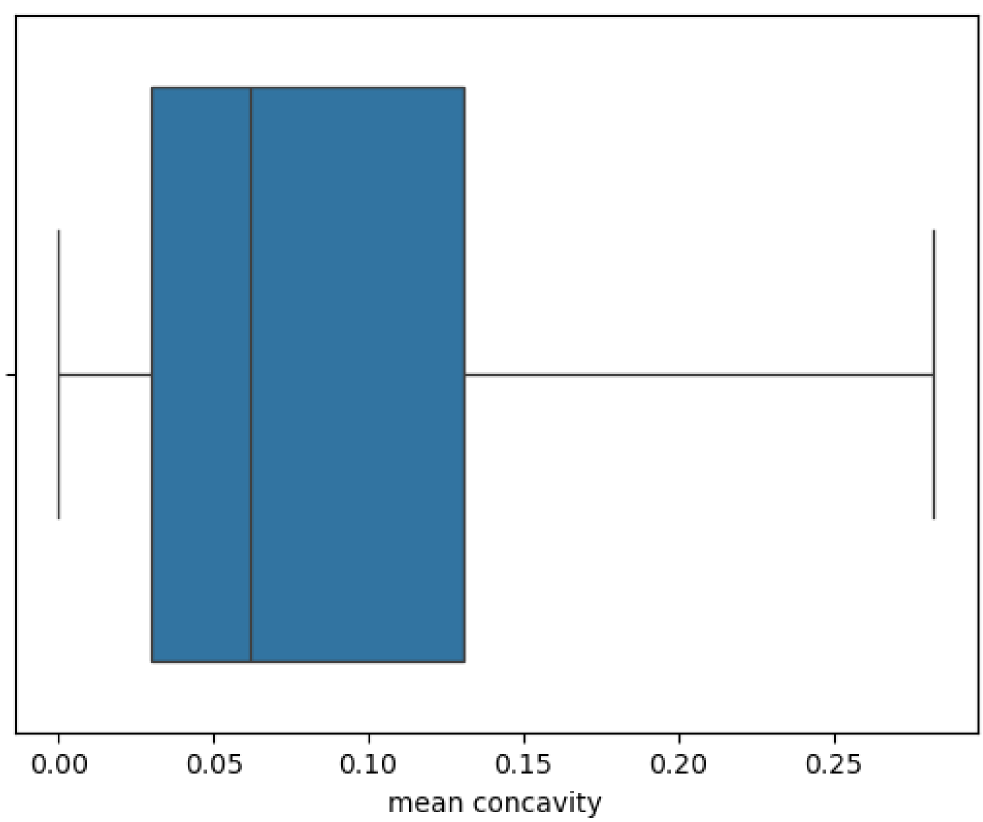


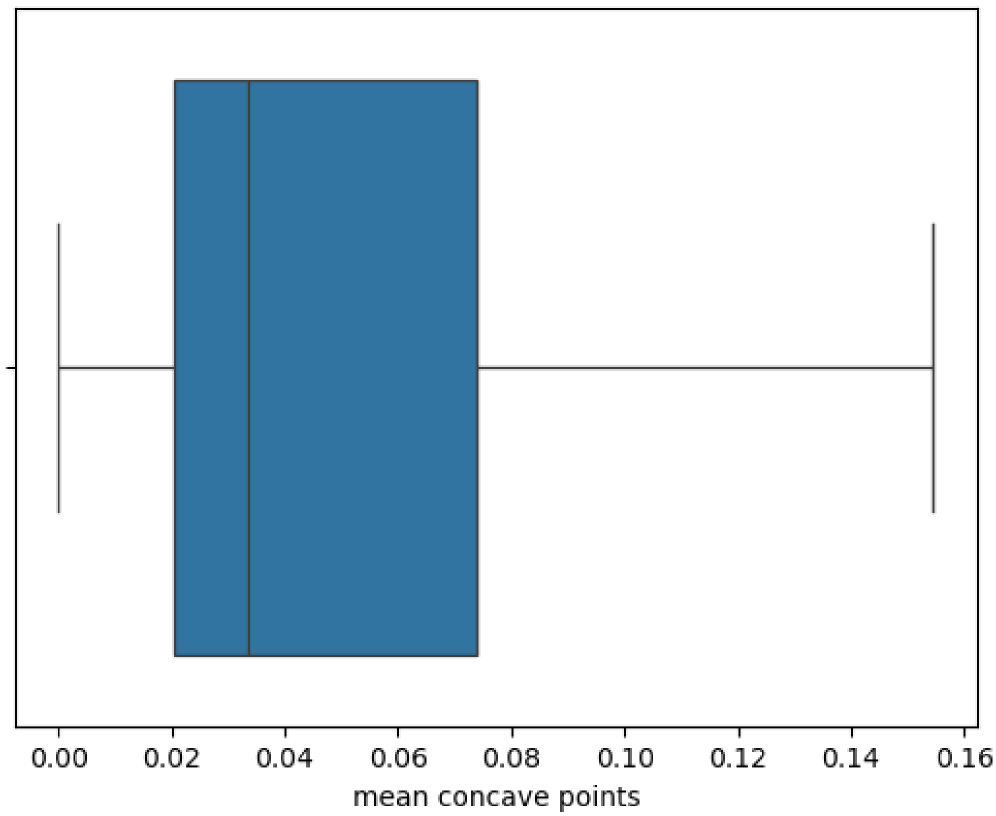
200
400
600
800
1000
1200
mean area

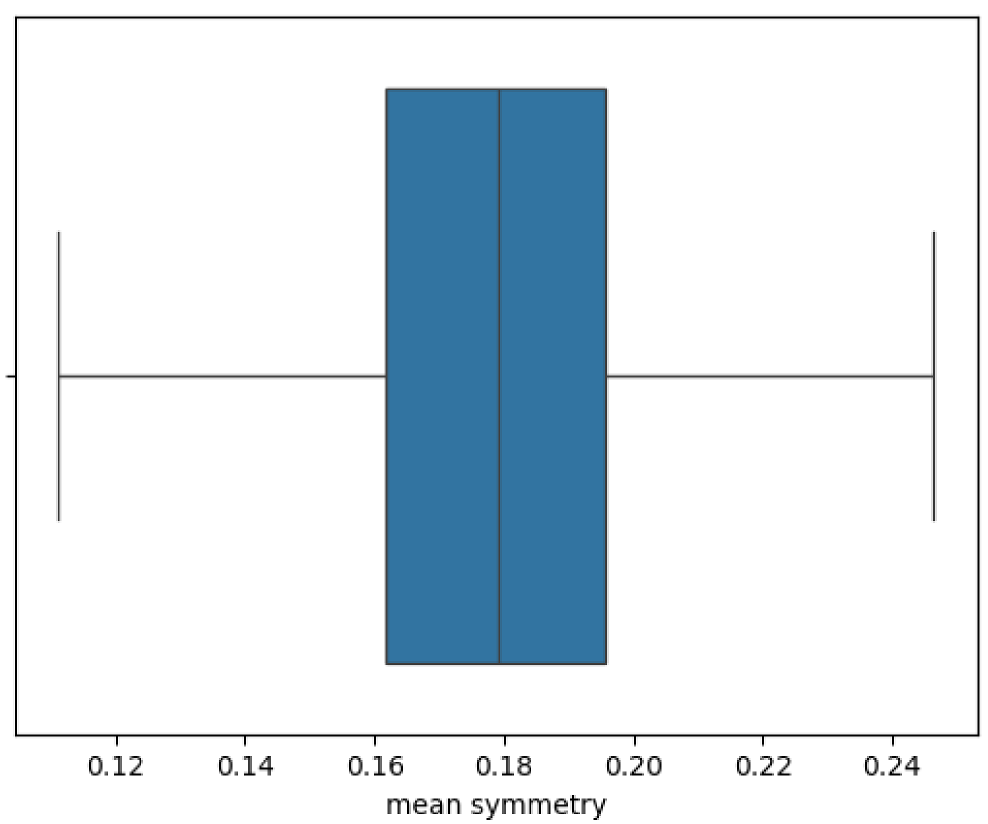


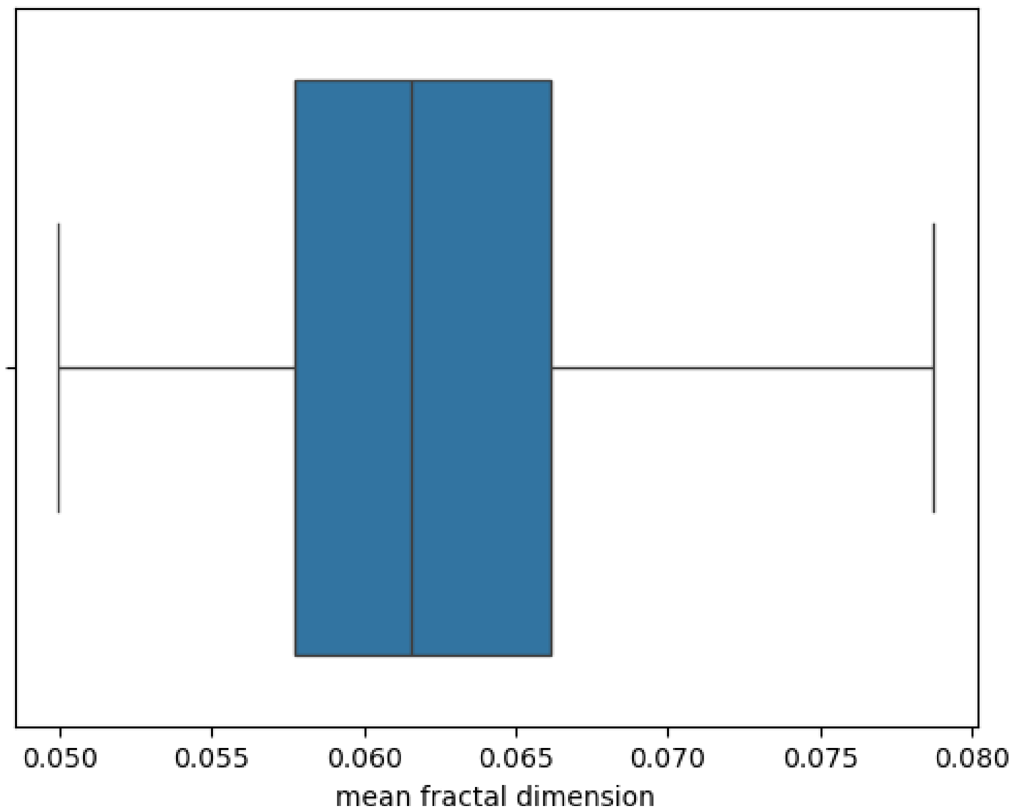
0.06
0.07
0.08
0.09
0.10
0.11
0.12
0.13
mean smoothness

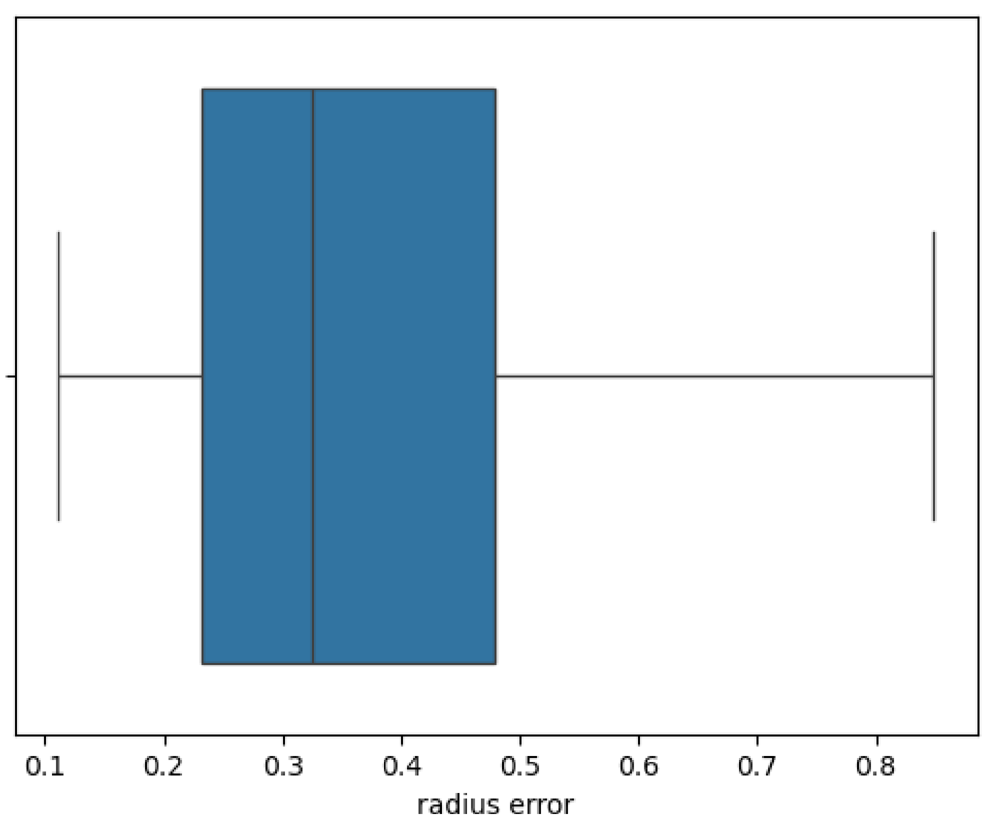

0.05
0.10
0.15
0.20
mean compactness

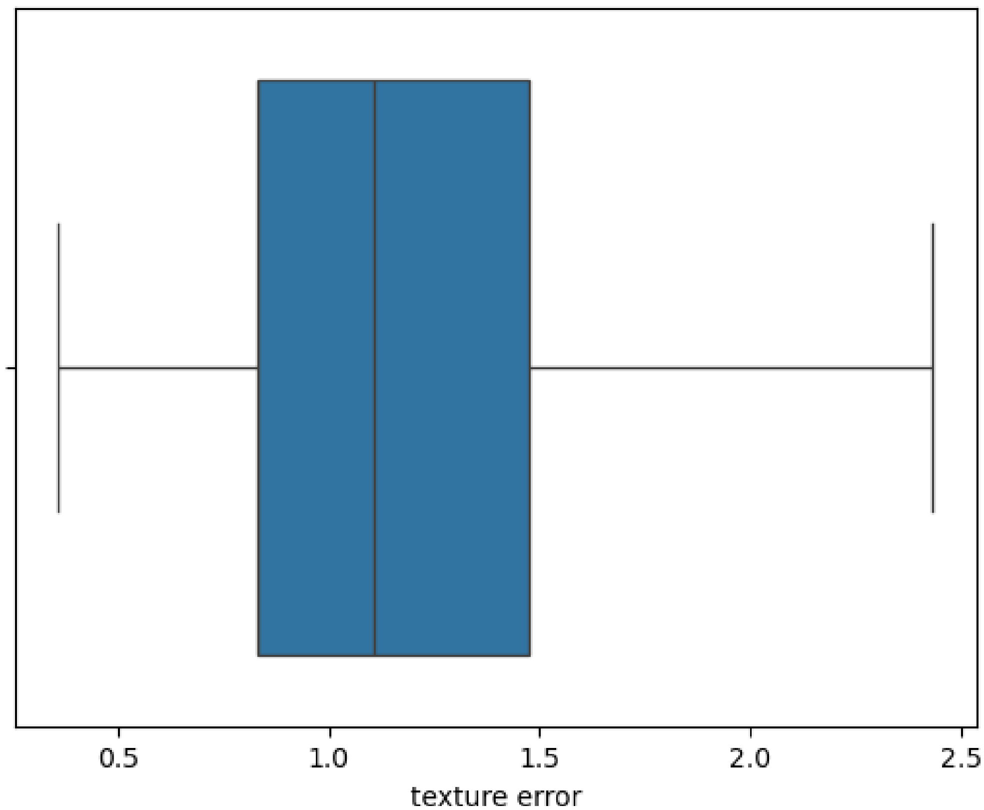

0.00
0.05
0.10
0.15
0.20
0.25
mean concavity

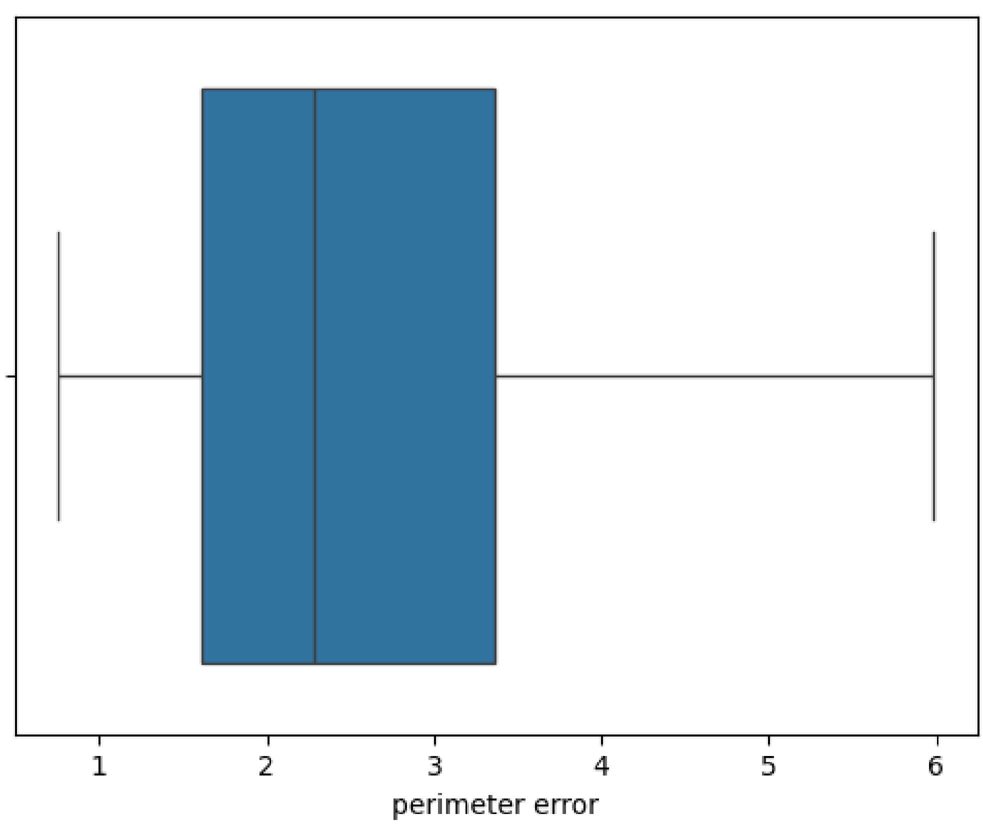

0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
mean concave points

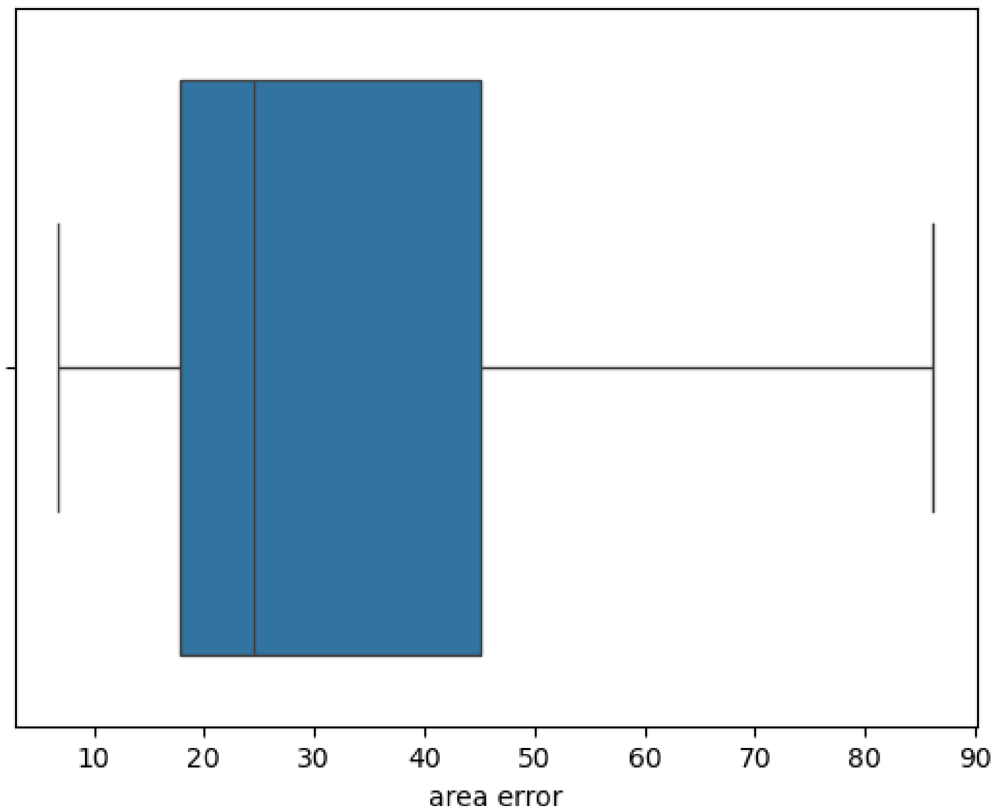

0.12
0.14
0.16
0.18
0.20
0.22
0.24
mean symmetry

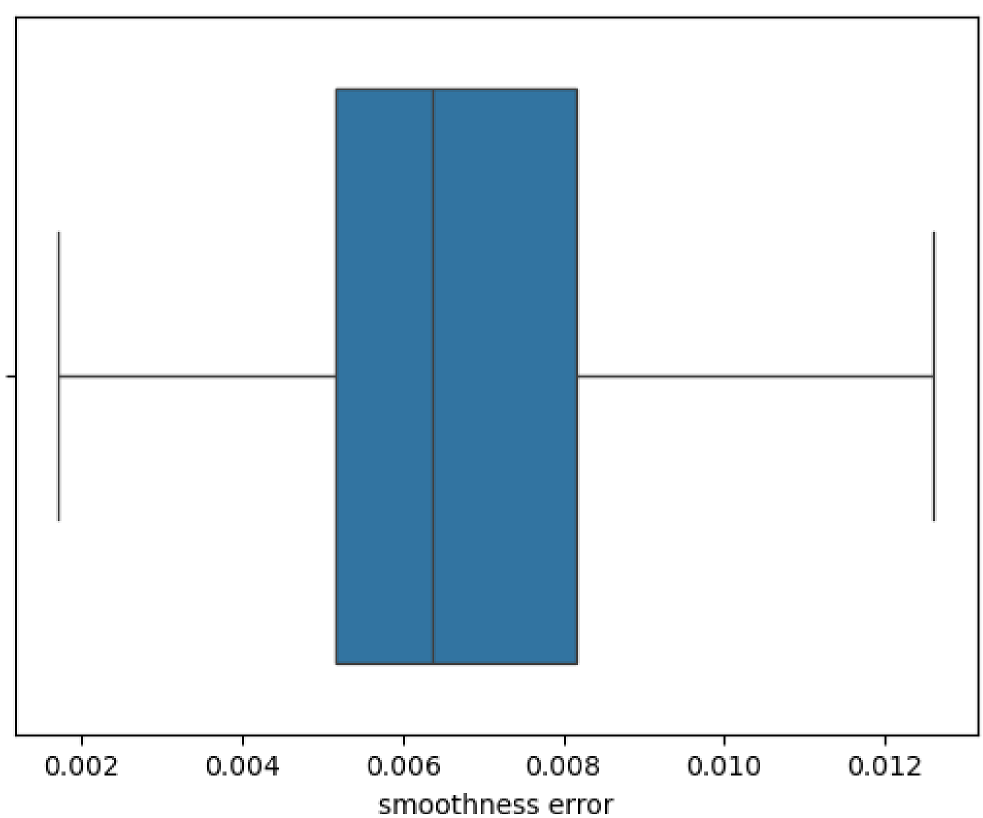

0.050
0.055
0.060
0.065
0.070
0.075
0.080
mean fractal dimension

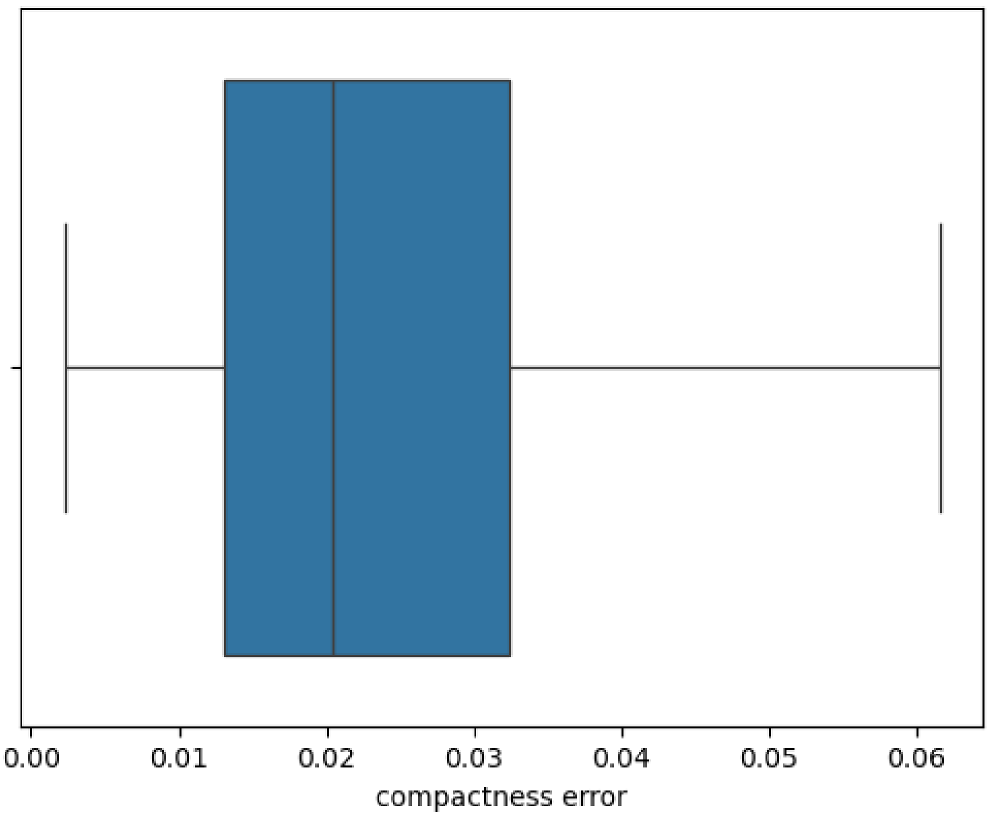

0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
radius error

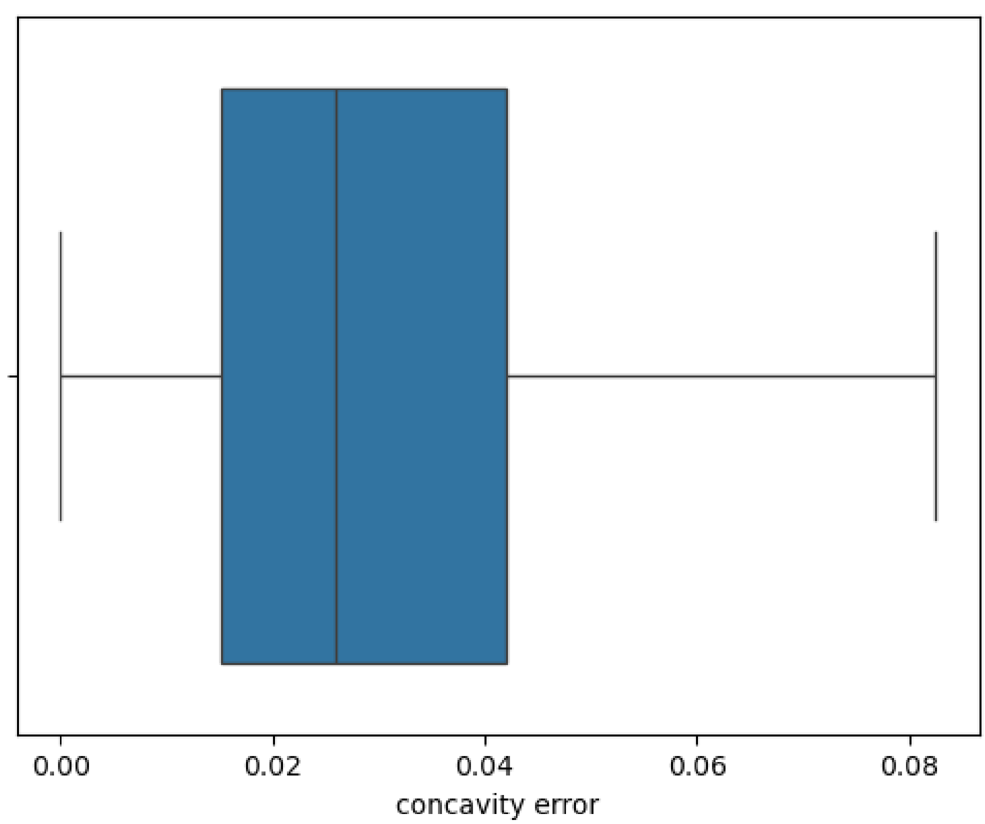

0.5
1.0
1.5
2.0
2.5
texture error

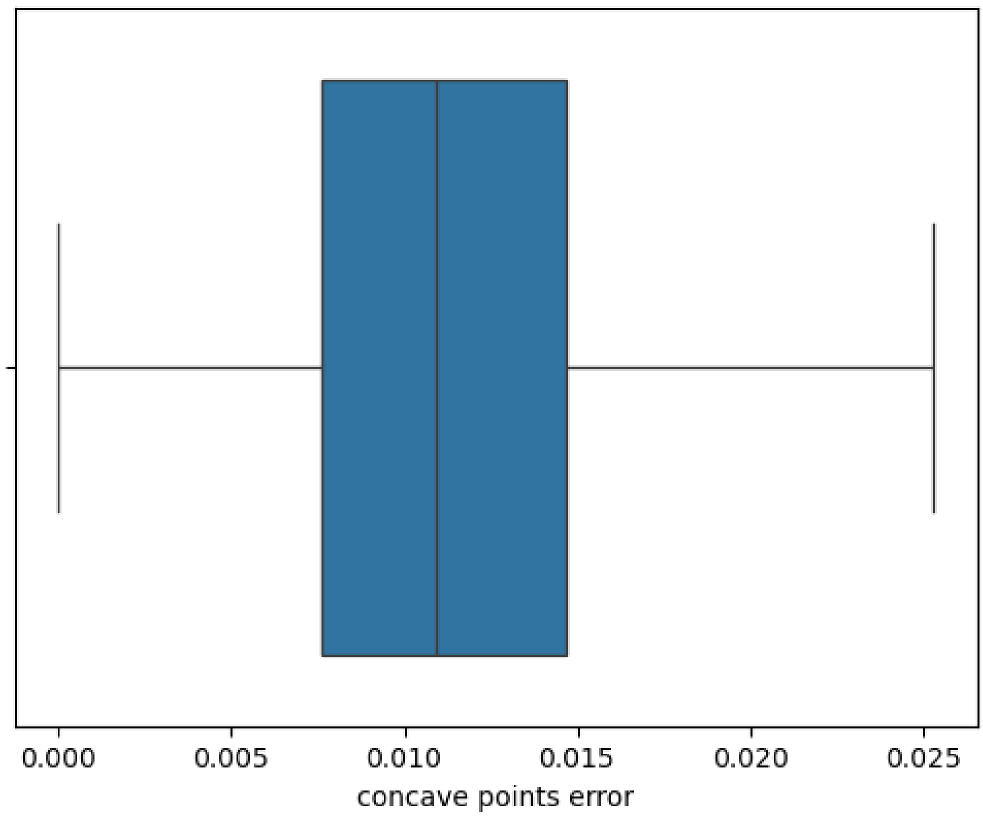

1
2
3
4
5
6
perimeter error

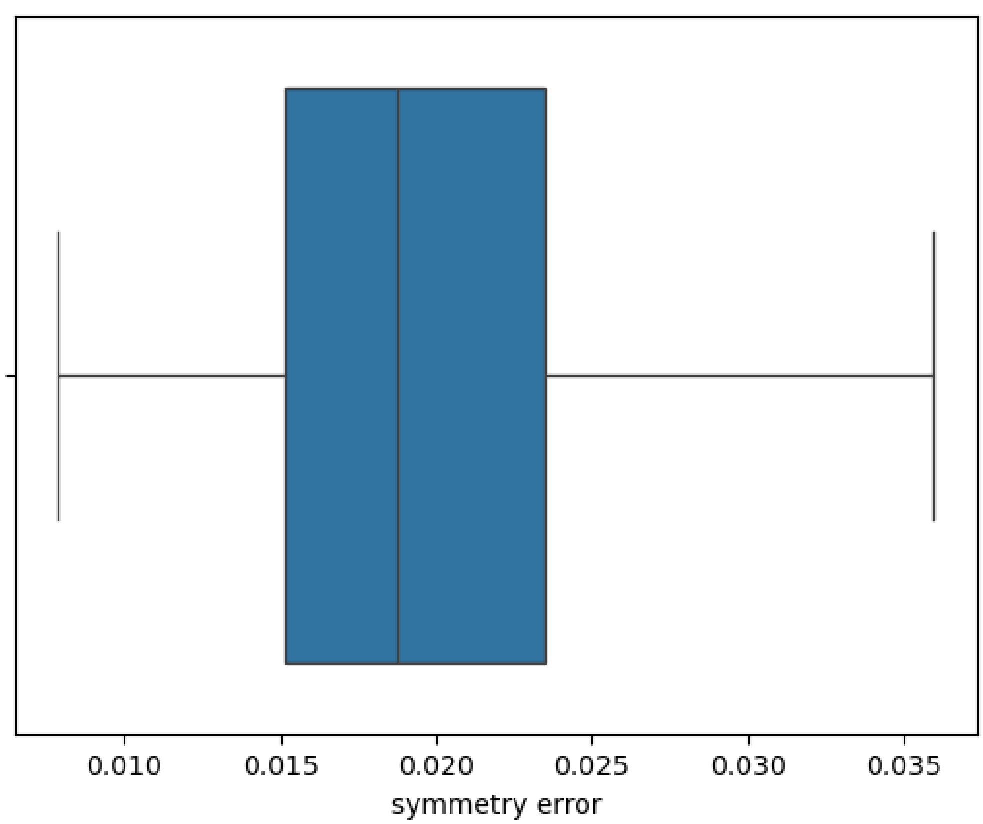

10
20
30
40
50
60
70
80
90
area error

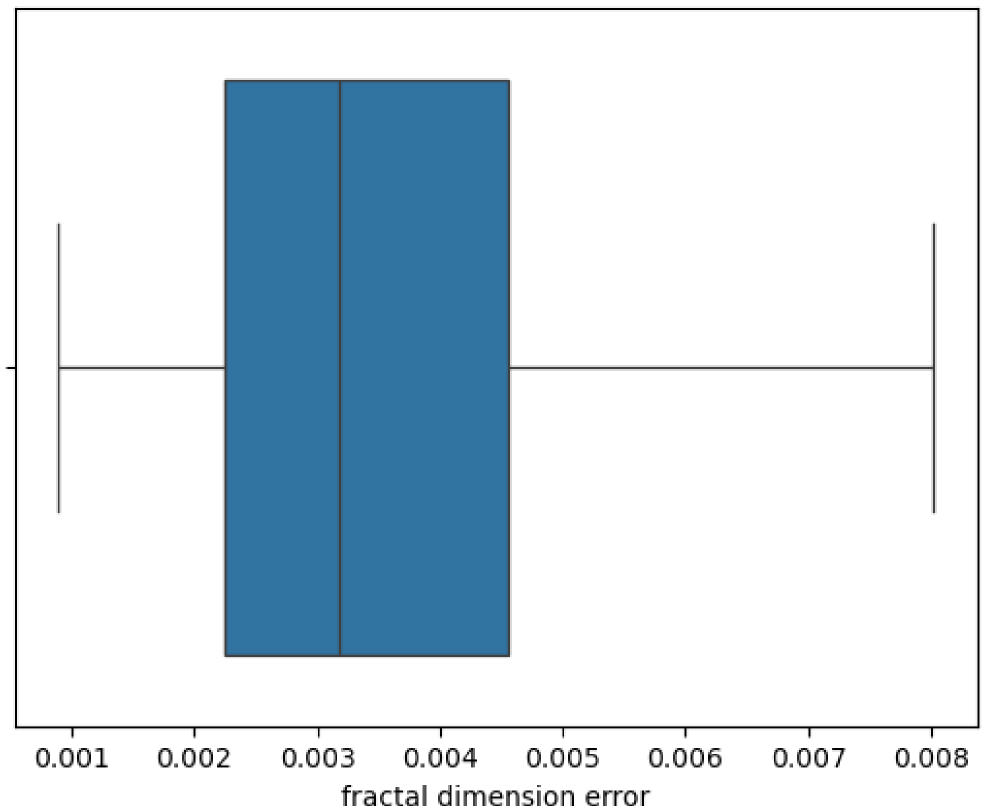

0.002
0.004
0.006
0.008
0.010
0.012
smoothness error

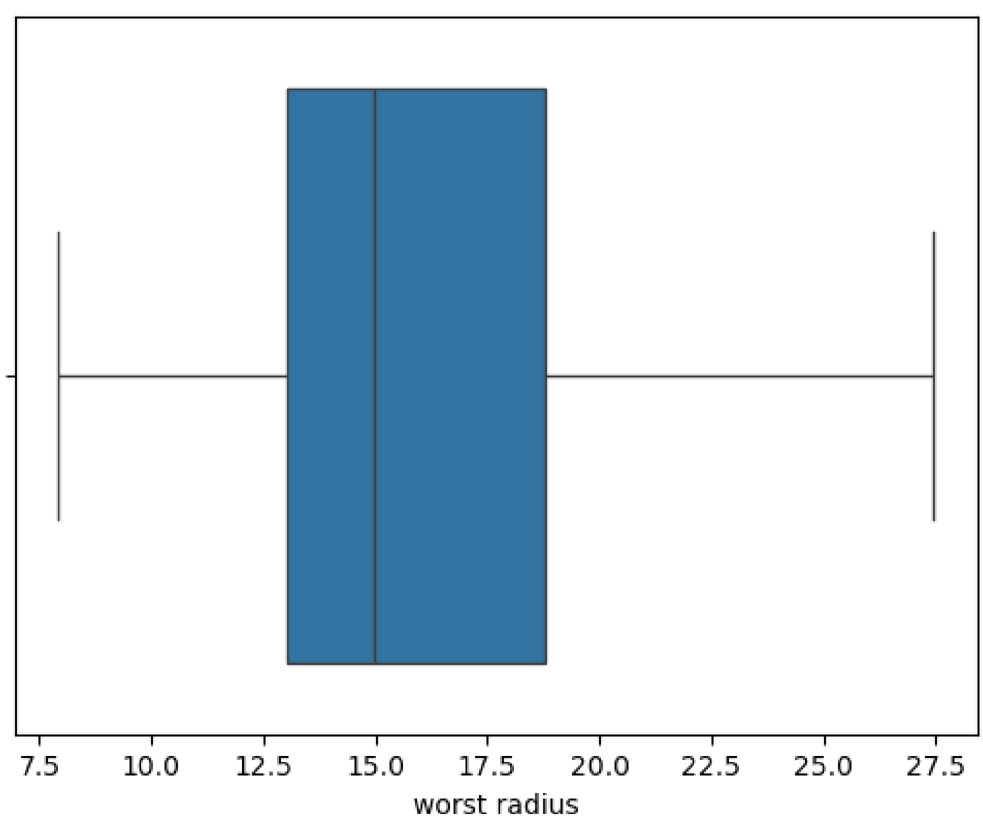

0.00
0.01
0.02
0.03
0.04
0.05
0.06
compactness error

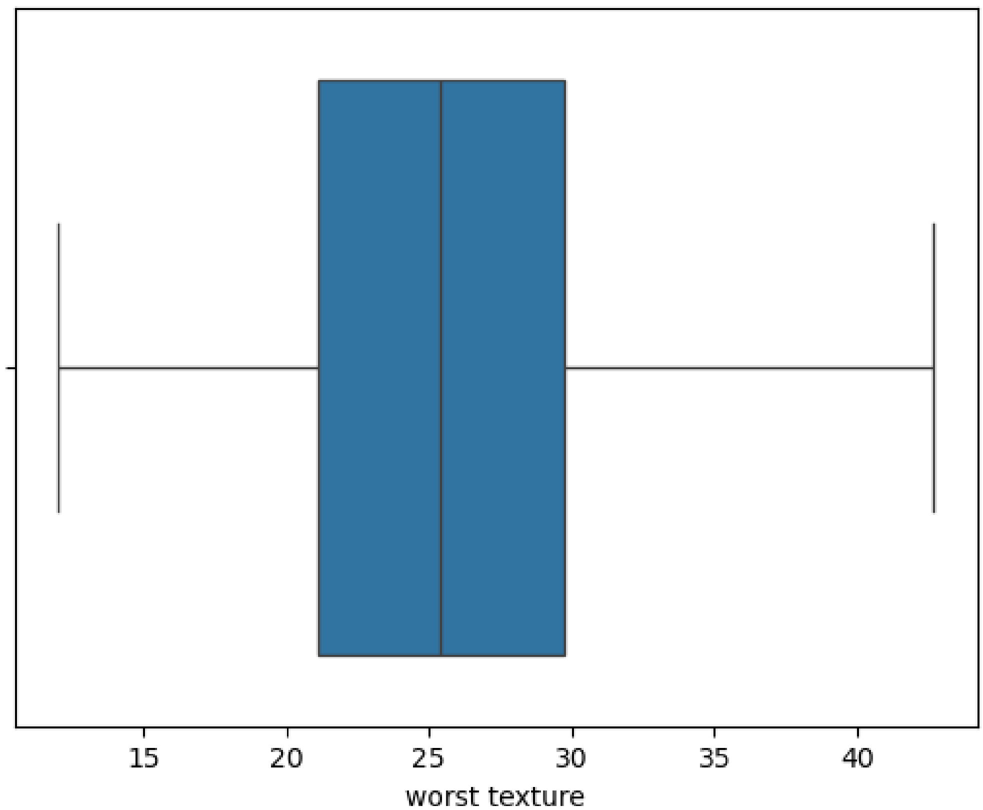

0.00
0.02
0.04
0.06
0.08
concavity error

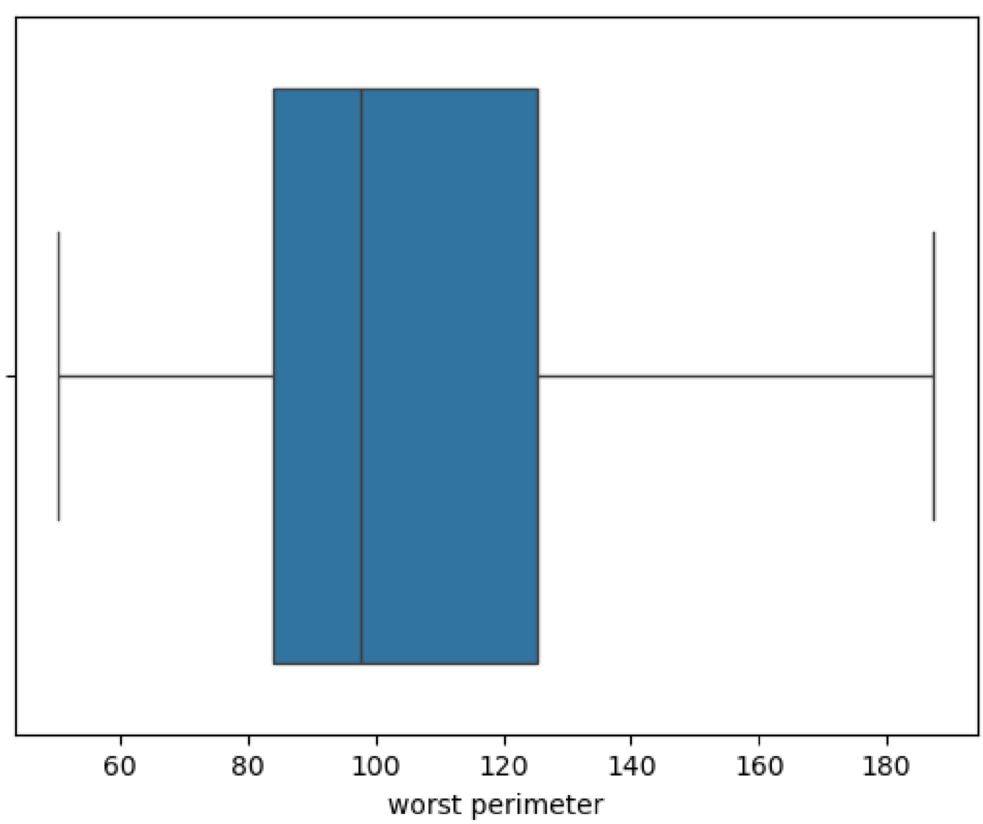

0.000
0.005
0.010
0.015
0.020
0.025
concave points error

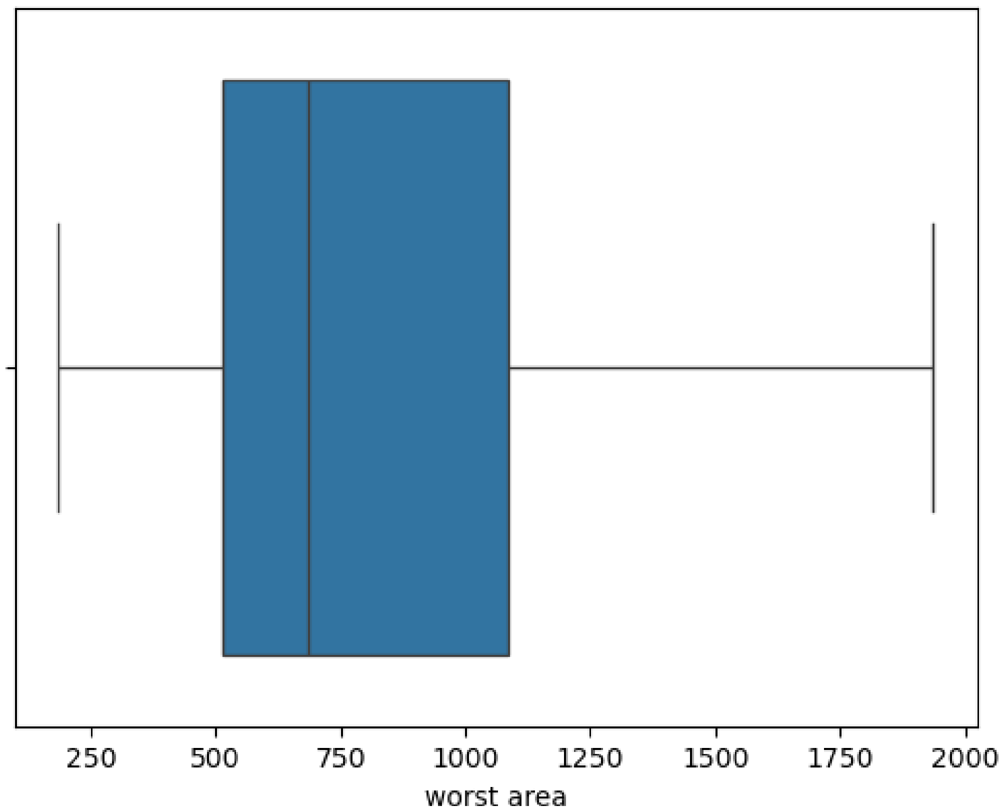

0.010
0.015
0.020
0.025
0.030
0.035
symmetry error

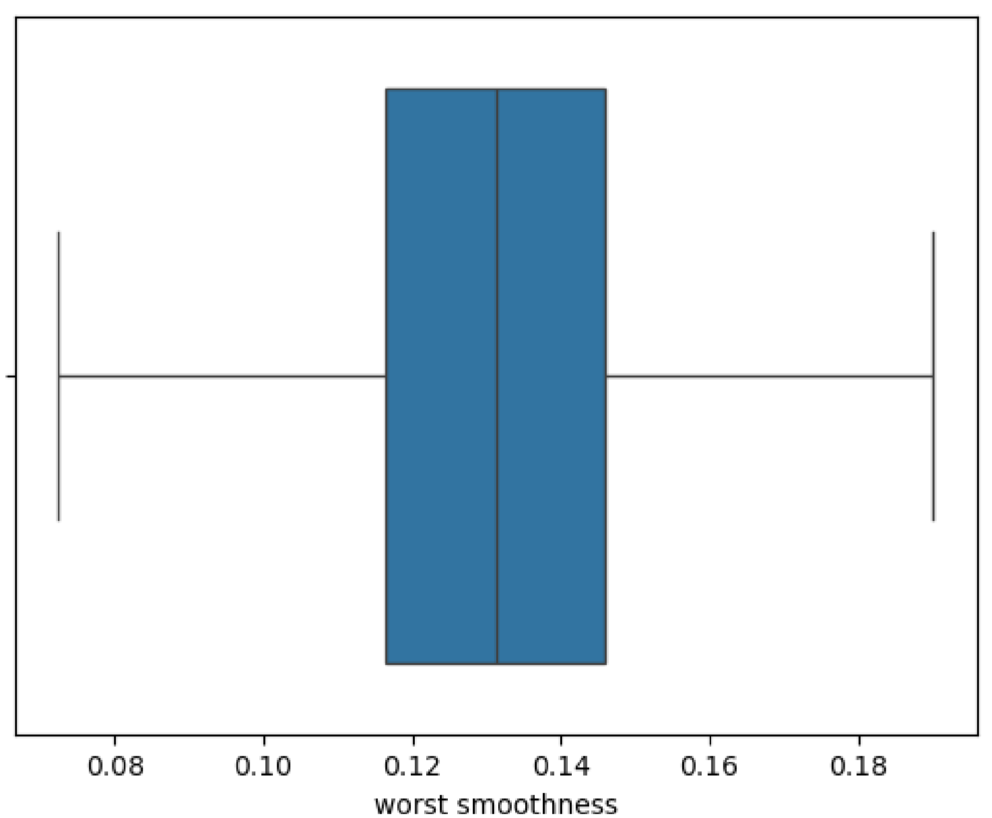

0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
fractal dimension error

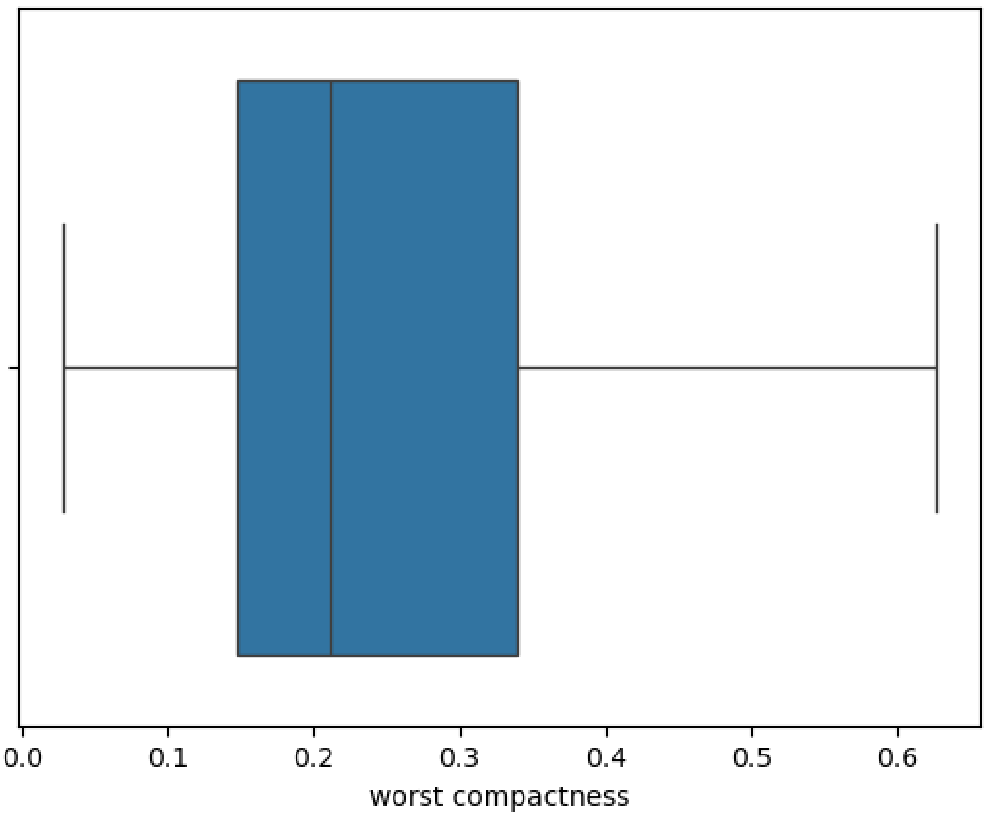

7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
27.5
worst radius


15
20
25
30
35
40
worst texture

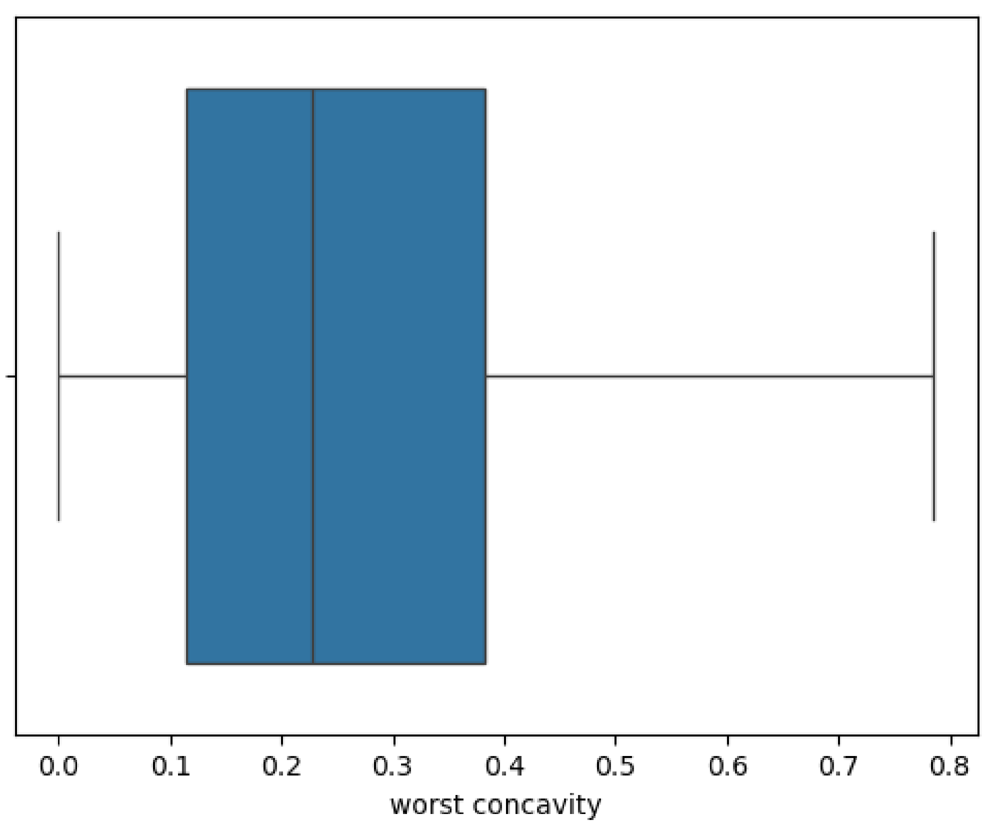

60
80
100
120
140
160
180
worst perimeter


250
500
750
1000
1250
1500
1750
2000
worst area

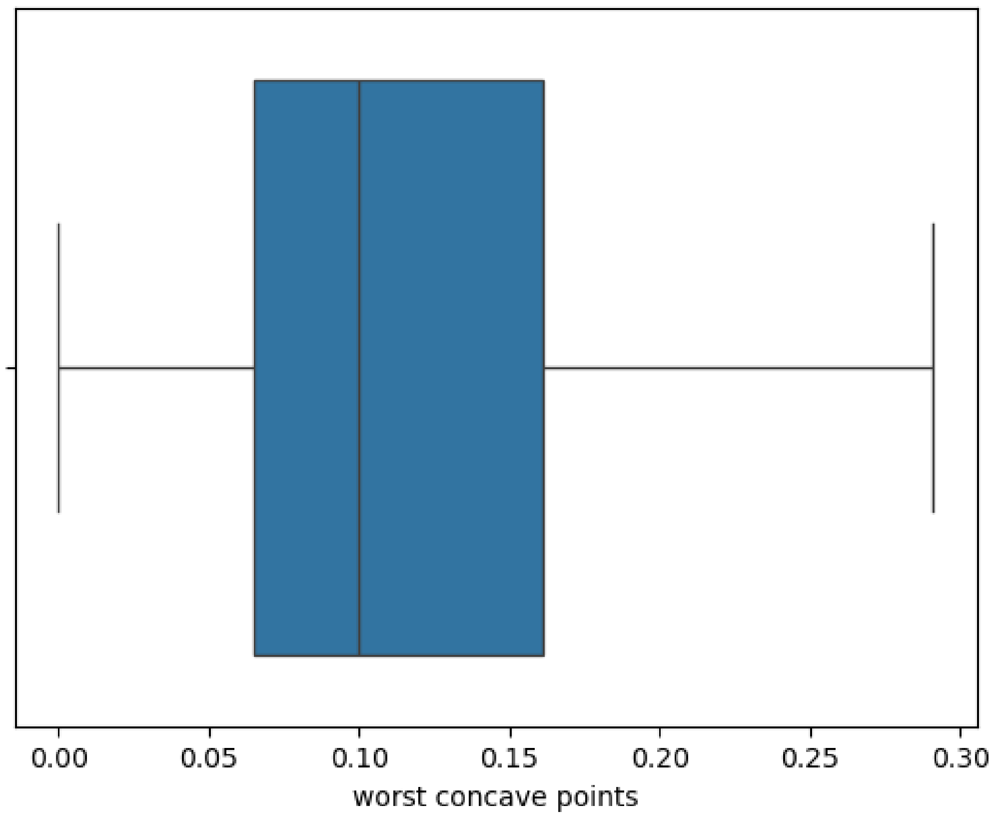

0.08
0.10
0.12
0.14
0.16
0.18
worst smoothness

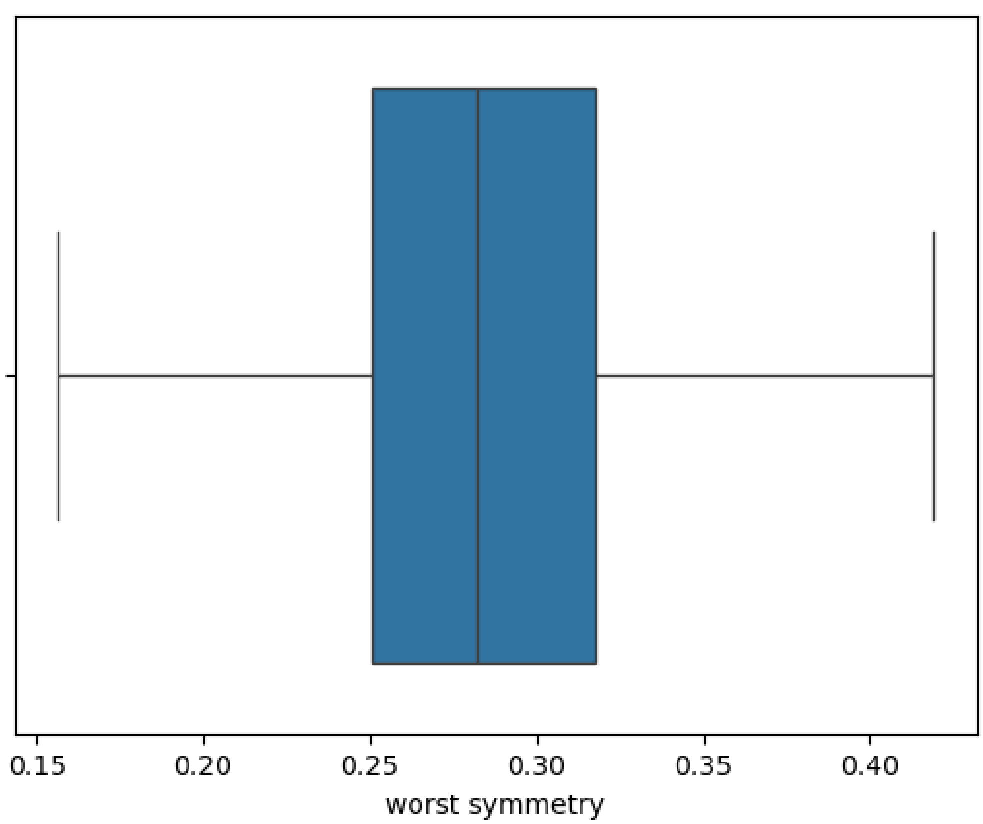


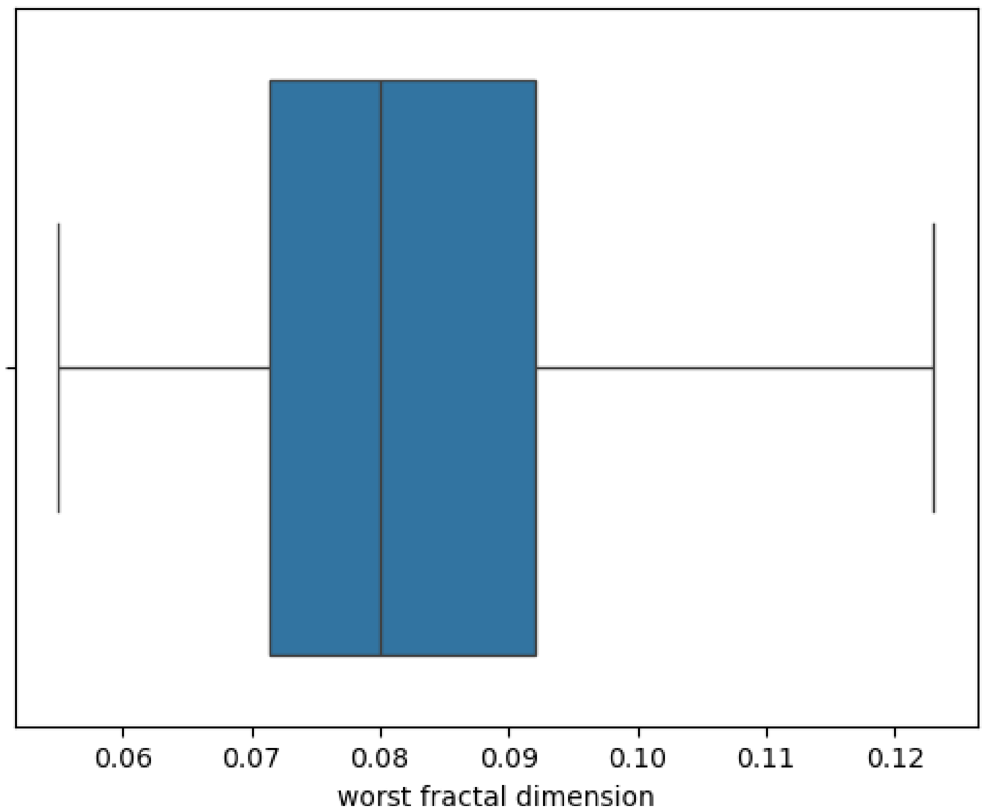
0.0
0.1
0.2
0.3
0.4
0.5
0.6
worst compactness

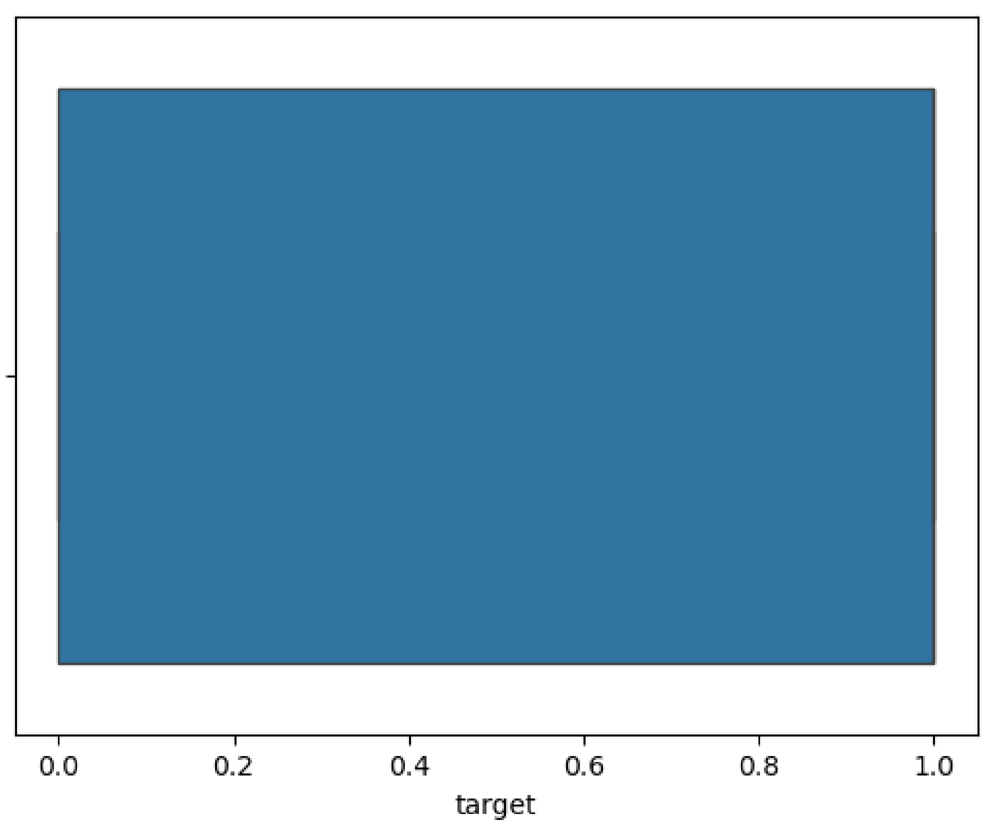


0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
worst concavity


0.00
0.05
0.10
0.15
0.20
0.25
0.30
worst concave points


0.15
0.20
0.25
0.30
0.35
0.40
worst symmetry


0.06
0.07
0.08
0.09
0.10
0.11
0.12
worst fractal dimension


0.0
0.2
0.4
0.6
0.8
1.0
target

```python
# checking skewness
df.skew()
```

```
mean radius                0.655953
mean texture               0.449700
mean perimeter             0.701081
mean area                  0.922884
mean smoothness            0.257712
mean compactness           0.826755
mean concavity             1.023859
mean concave points        1.004049
mean symmetry              0.403621
mean fractal dimension     0.682430
radius error               1.025031
texture error              0.740987
perimeter error            1.034389
area error                 1.130940
smoothness error           0.780923
compactness error          0.990285
concavity error            0.916740
concave points error       0.539571
symmetry error             0.869297
fractal dimension error    0.979344
worst radius               0.849779
worst texture              0.386858
worst perimeter            0.874870
worst area                 1.048970
worst smoothness           0.247199
worst compactness          0.915295
worst concavity            0.809174
worst concave points       0.492616
worst symmetry             0.521772
worst fractal dimension    0.831581
target                    -0.528461
dtype: float64
```

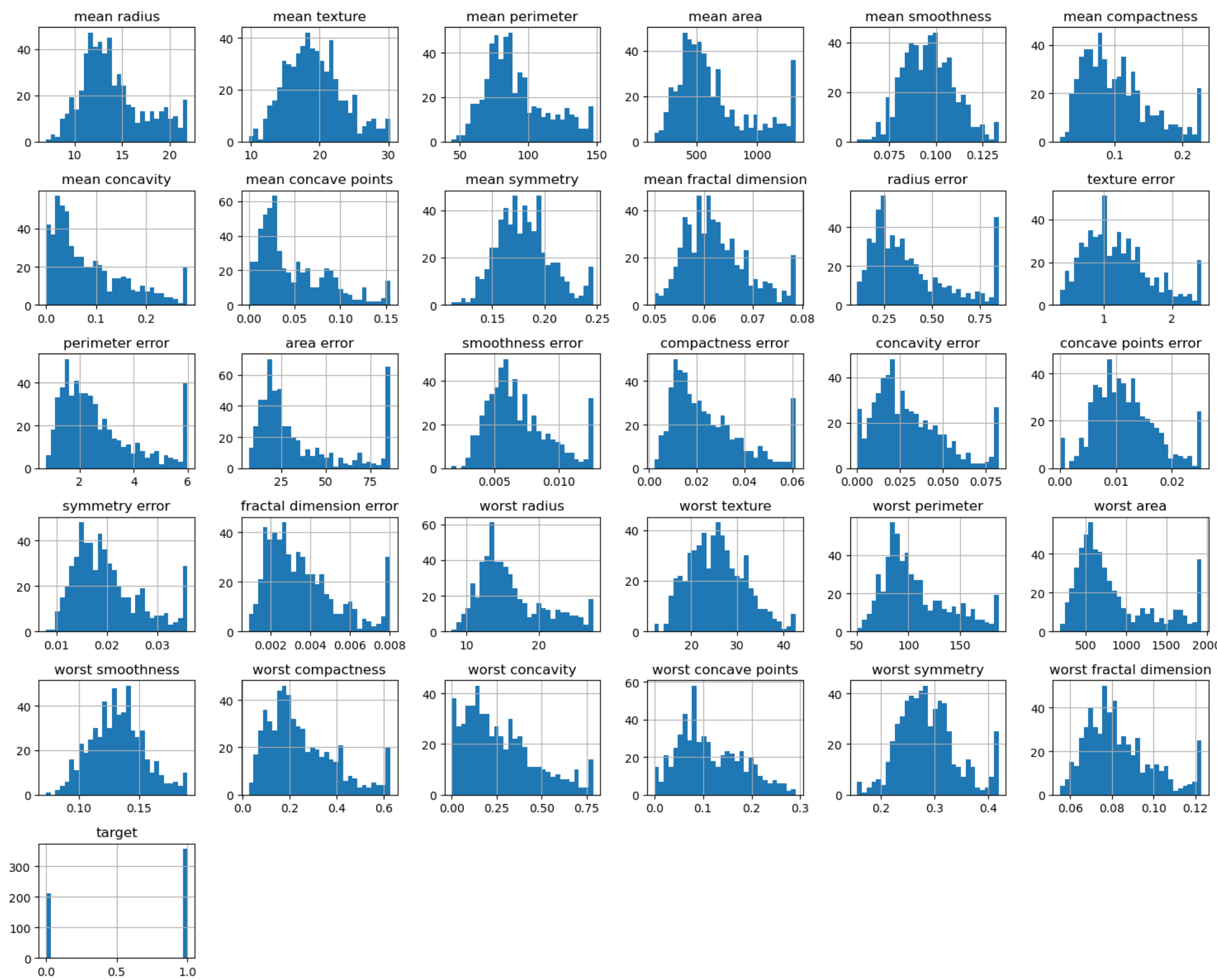
EDA

```python
df1 = df.copy()
df1.head()
```

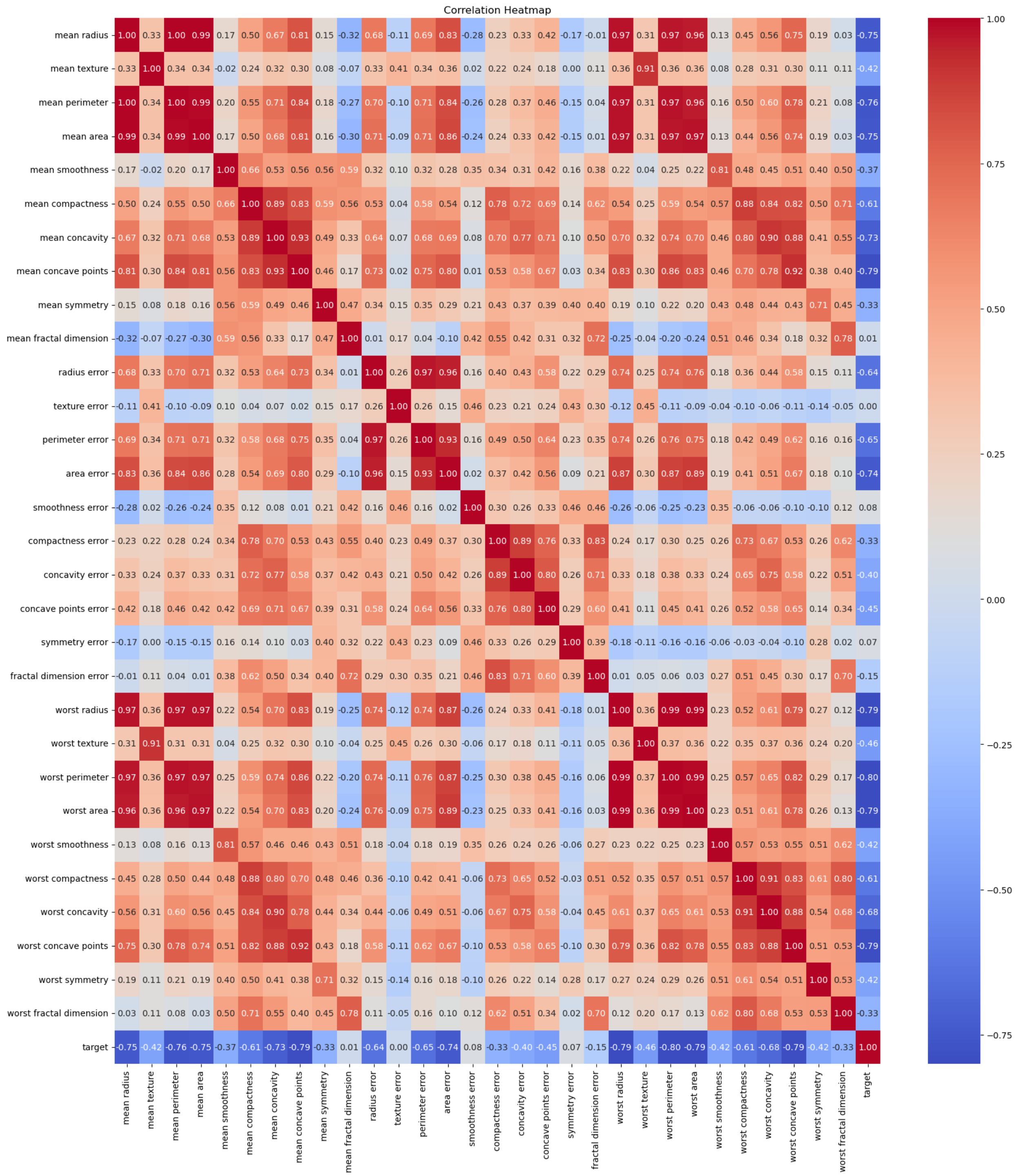
```
   mean radius  mean texture  mean perimeter  mean area  mean 
smoothness  \
0        17.99         10.38          122.80     1001.0         
0.118400   
1        20.57         17.77          132.90     1326.0         
0.084740   
2        19.69         21.25          130.00     1203.0         
0.109600   
```

```
3         11.42         20.38           77.58      386.1
0.133695
4         20.29         14.34          135.10     1297.0
0.100300

   mean compactness  mean concavity  mean concave points  mean
symmetry  \
0           0.22862         0.28241              0.14710
0.2419
1           0.07864         0.08690              0.07017
0.1812
2           0.15990         0.19740              0.12790
0.2069
3           0.22862         0.24140              0.10520
0.2464
4           0.13280         0.19800              0.10430
0.1809

   mean fractal dimension  ...  worst texture  worst perimeter  worst
area  \
0                 0.07871  ...          17.33           184.60
1937.05
1                 0.05667  ...          23.41           158.80
1937.05
2                 0.05999  ...          25.53           152.50
1709.00
3                 0.07875  ...          26.50            98.87
567.70
4                 0.05883  ...          16.67           152.20
1575.00

   worst smoothness  worst compactness  worst concavity  worst concave
points  \
0            0.1622            0.62695           0.7119
0.2654
1            0.1238            0.18660           0.2416
0.1860
2            0.1444            0.42450           0.4504
0.2430
3            0.1901            0.62695           0.6869
0.2575
4            0.1374            0.20500           0.4000
0.1625

   worst symmetry  worst fractal dimension  target
0         0.41915                  0.11890       0
1         0.27500                  0.08902       0
2         0.36130                  0.08758       0
3         0.41915                  0.12301       0
4         0.23640                  0.07678       0
```

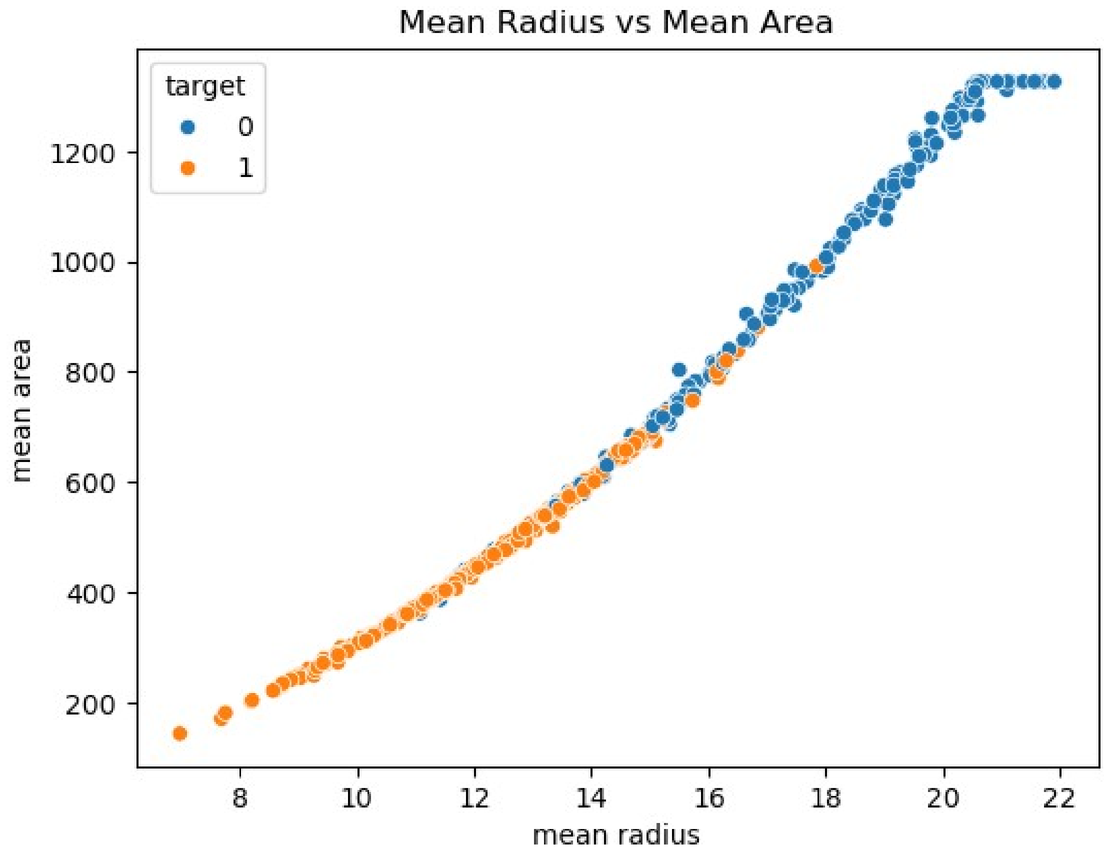
```
[5 rows x 31 columns]

# Histogram
df1.hist(bins=30, figsize=(15, 12))
plt.tight_layout()
plt.show()
```




```
# Compute correlation matrix
corr_matrix = df1.corr()

# Heatmap
plt.figure(figsize=(20, 22))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```




```python
# Scatterplot
sns.scatterplot(x="mean radius", y="mean area", hue="target", data=df)
plt.title("Mean Radius vs Mean Area")
plt.show()
```



PREPROCESSING STEPS AND EXPLAINATION

1.Fetched basic details of the dataset with info(), describe(), shape and dtypes

2.Checked for duplicate values and null values. The data dont have any null/ duplicated values

3.Added boxplot for every column to visualise if there is any outliers present also added a single box plot chart combining every feature to compare the outliers

4.Used IQR method to find the outliers of all features

5.Used capping method further to fix the outliers

6.Target feature doesn't have any outliers so dropped it for outlier fixing

7.After Capping added box plot again to see the changes after outlier fixation

8.Checked for the skew value for every feature . skew was in a good range.

9.Drawn a histogram for every feature

10.Added a correlation heatmap to see the relationship

11.Added a scattershot for mean radius vs mean area

FEATURE SELECTION

```
# Using correlation matrix

# Compute correlation matrix
corr_matrix = df1.corr().abs()

# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape),
k=1).astype(bool))

# Find features with correlation > 0.9
to_drop = [column for column in upper.columns if any(upper[column] >
0.9)]

print(f"Features to drop due to high correlation: {to_drop}")

# Drop features
df1_reduced = df1.drop(columns=to_drop)
```

```
Features to drop due to high correlation: ['mean perimeter', 'mean
area', 'mean concave points', 'perimeter error', 'area error', 'worst
radius', 'worst texture', 'worst perimeter', 'worst area', 'worst
concavity', 'worst concave points']
```

```
df1_reduced.head()
```

```
   mean radius  mean texture  mean smoothness  mean compactness  \
0        17.99         10.38         0.118400           0.22862
1        20.57         17.77         0.084740           0.07864
2        19.69         21.25         0.109600           0.15990
3        11.42         20.38         0.133695           0.22862
4        20.29         14.34         0.100300           0.13280

   mean concavity  mean symmetry  mean fractal dimension  radius error
\
0         0.28241         0.2419                 0.07871       0.84865

1         0.08690         0.1812                 0.05667       0.54350

2         0.19740         0.2069                 0.05999       0.74560

3         0.24140         0.2464                 0.07875       0.49560

4         0.19800         0.1809                 0.05883       0.75720


   texture error  smoothness error  compactness error  concavity error
\
0         0.9053          0.006399           0.049040          0.05373

1         0.7339          0.005225           0.013080          0.01860
```

```
2         0.7869          0.006150           0.040060          0.03832

3         1.1560          0.009110           0.061505          0.05661

4         0.7813          0.011490           0.024610          0.05688


   concave points error  symmetry error  fractal dimension error  \
0              0.01587         0.03003                 0.006193
1              0.01340         0.01389                 0.003532
2              0.02058         0.02250                 0.004571
3              0.01867         0.03596                 0.008023
4              0.01885         0.01756                 0.005115

   worst smoothness  worst compactness  worst symmetry  \
0            0.1622            0.62695         0.41915
1            0.1238            0.18660         0.27500
2            0.1444            0.42450         0.36130
3            0.1901            0.62695         0.41915
4            0.1374            0.20500         0.23640

   worst fractal dimension  target
0                  0.11890       0
1                  0.08902       0
2                  0.08758       0
3                  0.12301       0
4                  0.07678       0
```

SETTING X AND Y

```
y = df1_reduced['target']
y
```

```
0      0
1      0
2      0
3      0
4      0
      ..
564    0
565    0
566    0
567    0
568    1
Name: target, Length: 569, dtype: int32
```

```
x = df1_reduced.drop('target',axis=1)
x
```

```
     mean radius  mean texture  mean smoothness  mean compactness  \
0          17.99         10.38         0.118400           0.22862
1          20.57         17.77         0.084740           0.07864
2          19.69         21.25         0.109600           0.15990
3          11.42         20.38         0.133695           0.22862
4          20.29         14.34         0.100300           0.13280
..           ...           ...              ...               ...
564        21.56         22.39         0.111000           0.11590
565        20.13         28.25         0.097800           0.10340
566        16.60         28.08         0.084550           0.10230
567        20.60         29.33         0.117800           0.22862
568         7.76         24.54         0.057975           0.04362

     mean concavity  mean symmetry  mean fractal dimension  radius
error  \
0           0.28241         0.2419                 0.07871
0.84865
1           0.08690         0.1812                 0.05667
0.54350
2           0.19740         0.2069                 0.05999
0.74560
3           0.24140         0.2464                 0.07875
0.49560
4           0.19800         0.1809                 0.05883
0.75720
..              ...            ...                     ...            .
..
564         0.24390         0.1726                 0.05623
0.84865
565         0.14400         0.1752                 0.05533
0.76550
566         0.09251         0.1590                 0.05648
0.45640
567         0.28241         0.2397                 0.07016
0.72600
568         0.00000         0.1587                 0.05884
0.38570

     texture error  smoothness error  compactness error  concavity
error  \
0          0.90530          0.006399           0.049040
0.05373
1          0.73390          0.005225           0.013080
0.01860
2          0.78690          0.006150           0.040060
0.03832
3          1.15600          0.009110           0.061505
0.05661
4          0.78130          0.011490           0.024610
0.05688
```

```
..              ...                 ...                  ...                .
..
564        1.25600           0.010300            0.028910
0.05198
565        2.43415           0.005769            0.024230
0.03950
566        1.07500           0.005903            0.037310
0.04730
567        1.59500           0.006522            0.061505
0.07117
568        1.42800           0.007189            0.004660
0.00000

     concave points error  symmetry error  fractal dimension error  \
0                 0.01587         0.03003                 0.006193
1                 0.01340         0.01389                 0.003532
2                 0.02058         0.02250                 0.004571
3                 0.01867         0.03596                 0.008023
4                 0.01885         0.01756                 0.005115
..                    ...             ...                      ...
564               0.02454         0.01114                 0.004239
565               0.01678         0.01898                 0.002498
566               0.01557         0.01318                 0.003892
567               0.01664         0.02324                 0.006185
568               0.00000         0.02676                 0.002783

     worst smoothness  worst compactness  worst symmetry  \
0             0.16220            0.62695         0.41915
1             0.12380            0.18660         0.27500
2             0.14440            0.42450         0.36130
3             0.19010            0.62695         0.41915
4             0.13740            0.20500         0.23640
..                ...                ...             ...
564           0.14100            0.21130         0.20600
565           0.11660            0.19220         0.25720
566           0.11390            0.30940         0.22180
567           0.16500            0.62695         0.40870
568           0.08996            0.06444         0.28710

     worst fractal dimension
0                    0.11890
1                    0.08902
2                    0.08758
3                    0.12301
4                    0.07678
..                       ...
564                  0.07115
565                  0.06637
566                  0.07820
567                  0.12301
```

```
568                    0.07039

[569 rows x 19 columns]
```

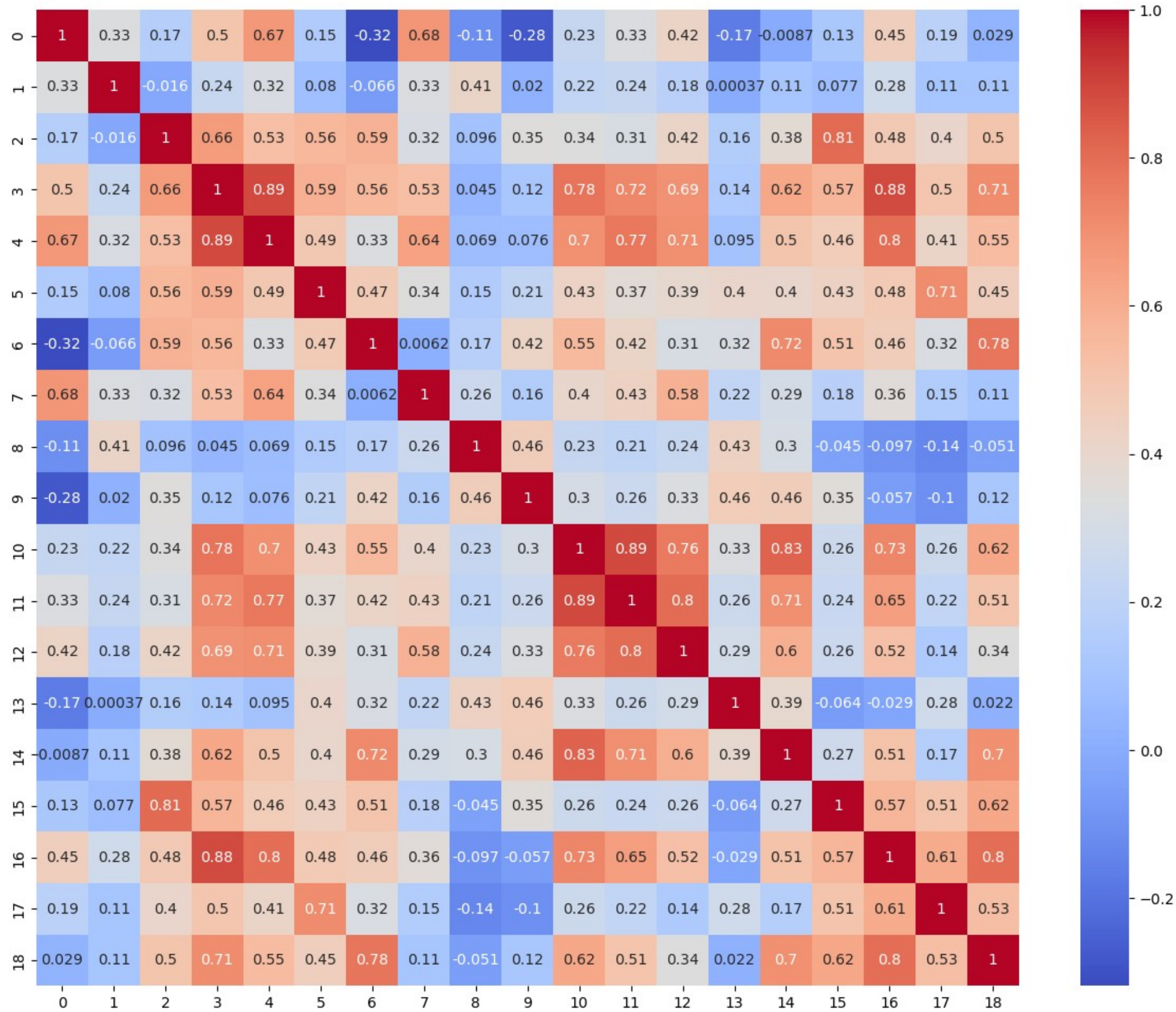
FEATURE SCALING

```
minmax_scaler = MinMaxScaler()

#Applying scaling
x_normalized = minmax_scaler.fit_transform(x)

# converting into dataframe
x_normalized = pd.DataFrame(x_normalized)
x_normalized.head()
```

```
          0         1         2         3         4         5         6
\
0  0.737918  0.032627  0.798006  1.000000  1.000000  0.966716
0.998611
1  0.910852  0.392501  0.353473  0.283215  0.307709  0.517751
0.233067
2  0.851867  0.561967  0.681788  0.671573  0.698984  0.707840
0.348385
3  0.297540  0.519601  1.000000  1.000000  0.854786  1.000000
1.000000
4  0.892084  0.225469  0.558967  0.542057  0.701108  0.515533
0.308093

          7         8         9        10        11        12
13  \
0  1.000000  0.262832  0.429967  0.789631  0.651352  0.626827
0.788803
1  0.586041  0.180188  0.322246  0.182742  0.225482  0.529268
0.213975
2  0.860205  0.205743  0.407120  0.638077  0.464541  0.812860
0.520621
3  0.521061  0.383712  0.678717  1.000000  0.686265  0.737420
1.000000
4  0.875941  0.203043  0.897096  0.377331  0.689538  0.744530
0.344683

         14        15        16        17        18
0  0.743273  0.762755  1.000000  1.000000  0.939532
1  0.369967  0.436224  0.265667  0.451171  0.499926
2  0.515726  0.611395  0.662392  0.779745  0.478741
3  1.000000  1.000000  1.000000  1.000000  1.000000
4  0.592043  0.551871  0.296351  0.304207  0.319847
```

```
correlation = x_normalized.corr()
plt.figure(figsize=(15, 12))
```

```python
sns.heatmap(correlation, annot=True, cmap='coolwarm')
plt.show()
```



SPLITTING TRAIN AND TEST

```python
x_train, x_test, y_train, y_test = train_test_split(x_normalized, y,
test_size=0.2, random_state=42)
```

BUILDING MODELS

```python
models = {
    "Logistic Regression": LogisticRegression(max_iter=500),
    "Decision Tree": DecisionTreeClassifier(max_depth=5),
    "Random Forest": RandomForestClassifier(n_estimators=100),
    "SVM": SVC(kernel='linear', C=1),
    "k-NN": KNeighborsClassifier(n_neighbors=5)
}
```

LOGISTIC REGRESSION

A logistic (sigmoid) function is used in the linear model known as logistic regression to forecast probability. Based on a threshold (usually 0.5), it generates probabilities for binary classification and designates a label (e.g., malignant or non-cancerous). In the feature space, the decision boundary is linear.

Why its suitable: It works well with linearly separable datasets and is easy to understand. Datasets on breast cancer frequently exhibit distinct patterns that logistic regression is well suited to model. In medical analysis, the coefficients offer useful insights regarding feature relevance.

DECISION TREE CLASSIFIER

Based on feature values, decision trees divide the data into subsets and produce branches until a leaf node makes a categorization. Criteria such as Gini Impurity or Information Gain are used to determine the divides.

Why its suitable: It efficiently manages both numerical and category features. The tree structure can be used to describe the classification process in medical circumstances because it is simple to see and understand. Even if they are non-linear, it captures intricate correlations in the data.

RANDOM FOREST CLASSIFIER

Multiple decision trees are constructed during training using the Random Forest ensemble approach, which then aggregates the results (using majority vote for classification). By bootstrapping the data and using random feature subsets for splitting, it adds unpredictability.

By lessening overfitting, which can happen in individual decision trees, it increases accuracy. robust to outliers and noisy data, which makes it appropriate for datasets with different feature distributions, such as breast cancer. It can help identify the elements impacting forecasts by ranking the relevance of features.

SVC

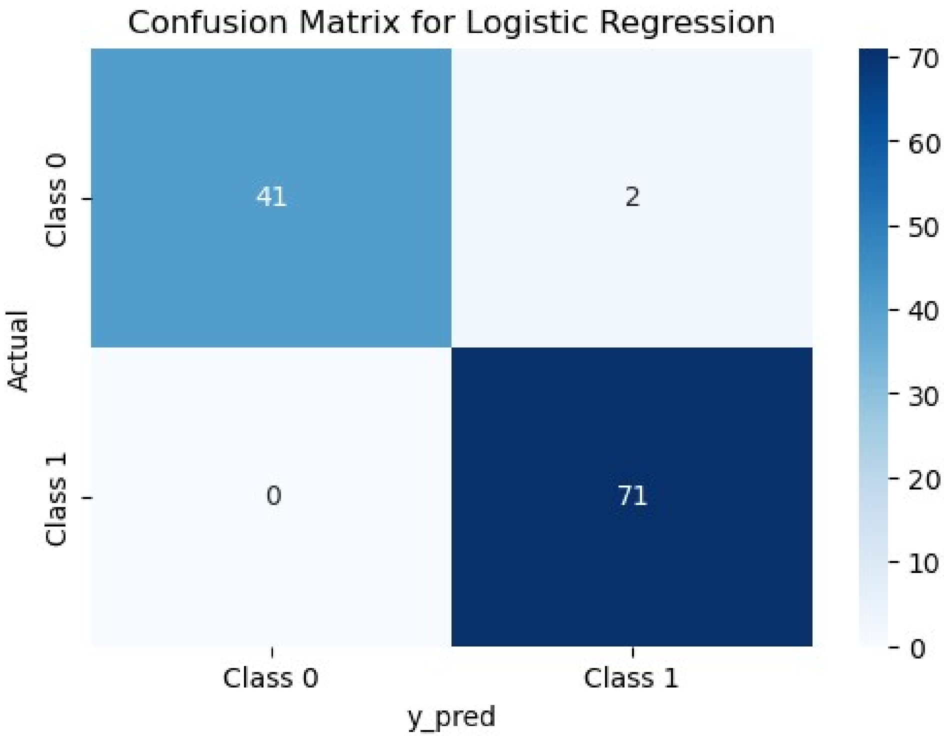
By maximizing the margin between data points of distinct classes, SVC seeks to identify the hyperplane that best divides them. For improved separability, it can convert data into higher dimensions using kernels (such as linear and RBF). It performs well in high-dimensional areas, such as the feature-rich breast cancer dataset. useful for datasets in which the initial feature space does not readily allow for the separation of classes. When there are more features than samples, it is resistant to overfitting.

K- NEAREST NEIGHBOURS (K-NN)

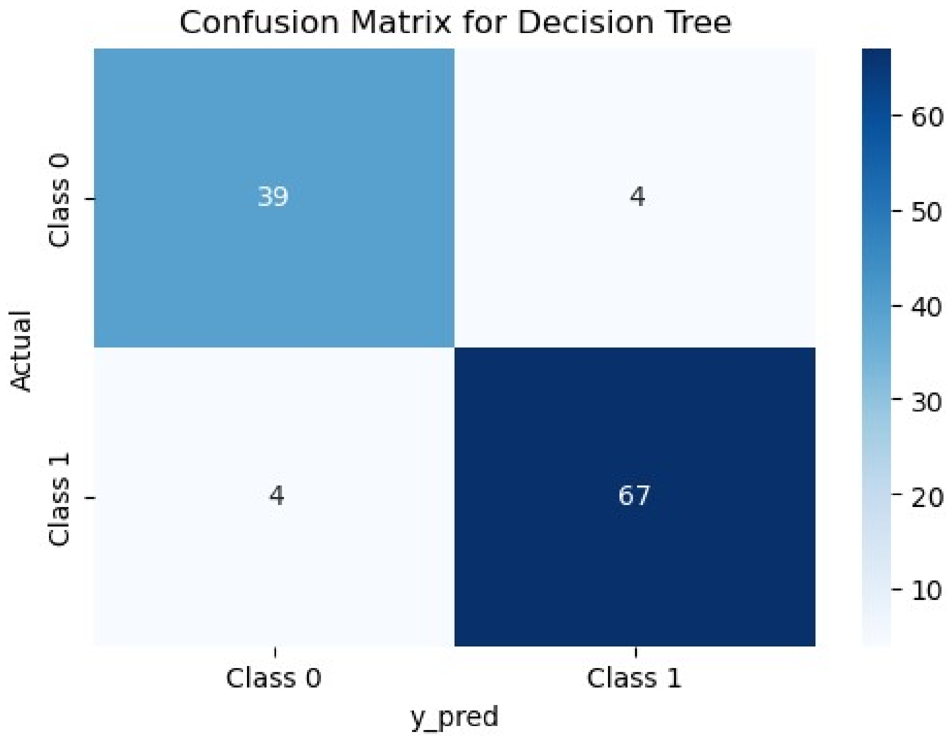
A data point is categorized by k-NN using the majority class of its k nearest neighbors in the feature space. Euclidean and other distance metrics are used to find neighbors. It is a straightforward non-parametric approach that doesn't assume anything about the distribution of the underlying data. Ideal for datasets with distinct and well-clustered class distributions. Because it is sensitive to feature magnitudes, it works best when feature scaling (such as standardization) is used.

```python
# Train and evaluate each model
results = {}
for model_name, model in models.items():
    print(f"Training {model_name}...")
    model.fit(x_train, y_train)  # Train the model
    y_pred = model.predict(x_test)  # Make predictions

    # Evaluate the model
    accuracy = accuracy_score(y_test, y_pred)
    results[model_name] = accuracy

    print(f"{model_name} Accuracy: {accuracy:.4f}")
    print(f"{model_name} Classification Report:\
n{classification_report(y_test, y_pred)}\n")

     # Generate Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    print(f"{model_name} Confusion Matrix:\n{cm}\n")

    # Visualize Confusion Matrix
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
xticklabels=["Class 0", "Class 1"], yticklabels=["Class 0", "Class
1"])
    plt.title(f"Confusion Matrix for {model_name}")
    plt.ylabel("Actual")
    plt.xlabel("y_pred")
    plt.show()
```

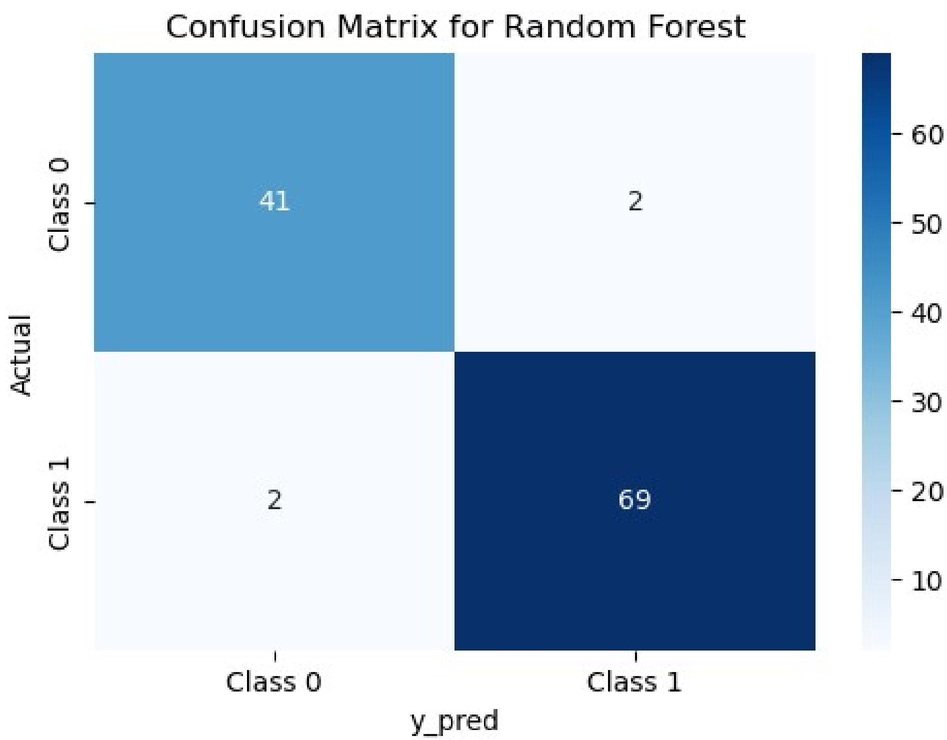
```
Training Logistic Regression...
Logistic Regression Accuracy: 0.9825
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.95      0.98        43
           1       0.97      1.00      0.99        71

    accuracy                           0.98       114
   macro avg       0.99      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114


Logistic Regression Confusion Matrix:
[[41  2]
 [ 0 71]]
```



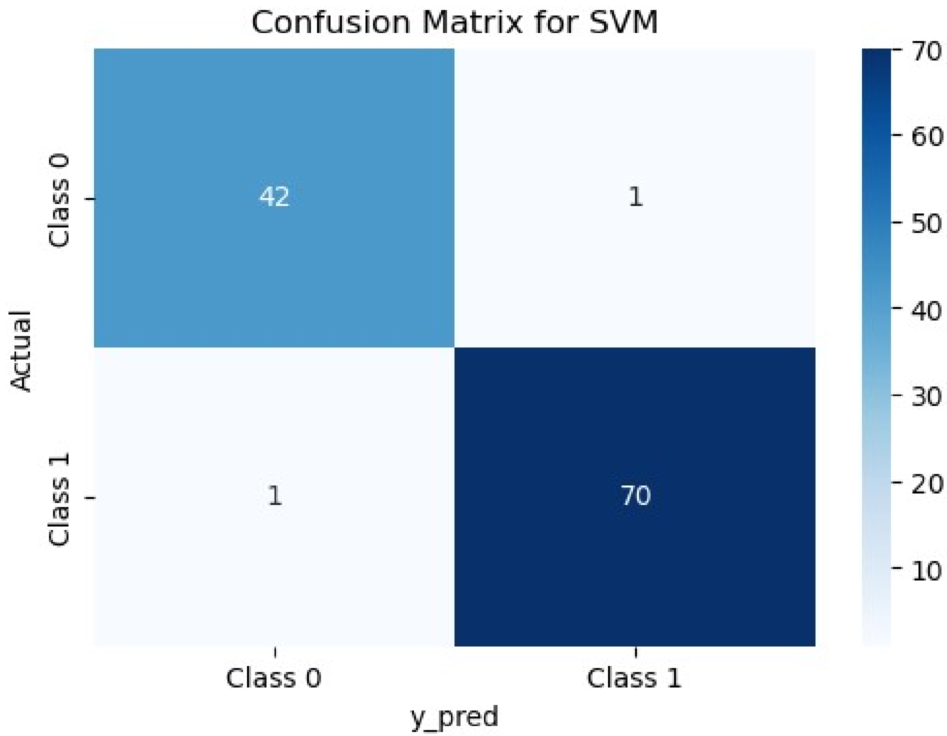

```
Training Decision Tree...
Decision Tree Accuracy: 0.9298
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.91      0.91        43
           1       0.94      0.94      0.94        71

    accuracy                           0.93       114
   macro avg       0.93      0.93      0.93       114
weighted avg       0.93      0.93      0.93       114


Decision Tree Confusion Matrix:
[[39  4]
 [ 4 67]]
```



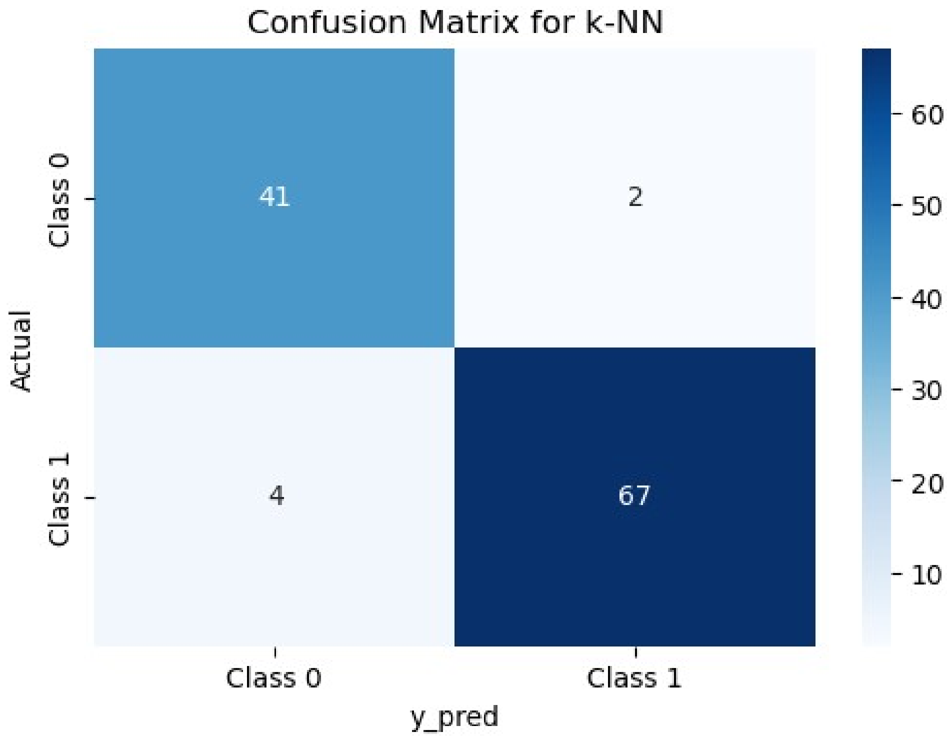

```
Training Random Forest...
Random Forest Accuracy: 0.9649
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.95      0.95        43
           1       0.97      0.97      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.96      0.96       114
weighted avg       0.96      0.96      0.96       114


Random Forest Confusion Matrix:
[[41  2]
 [ 2 69]]
```




```
Training SVM...
SVM Accuracy: 0.9825
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        43
           1       0.99      0.99      0.99        71

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114


SVM Confusion Matrix:
[[42  1]
 [ 1 70]]
```




```
Training k-NN...
k-NN Accuracy: 0.9474
k-NN Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.95      0.93        43
           1       0.97      0.94      0.96        71

    accuracy                           0.95       114
   macro avg       0.94      0.95      0.94       114
weighted avg       0.95      0.95      0.95       114


k-NN Confusion Matrix:
[[41  2]
 [ 4 67]]
```



MODEL EVALUATION

```
# Comparison of results
print("Model Accuracy Comparison:")
for model, acc in results.items():
    print(f"{model}: {acc:.4f}")
```

```
Model Accuracy Comparison:
Logistic Regression: 0.9825
Decision Tree: 0.9298
Random Forest: 0.9649
SVM: 0.9825
k-NN: 0.9474
```

```
## finding best model
# Get the best model from sorted_results
best_model_name = max(results, key=results.get)
best_model_accuracy = results[best_model_name]

print(f"\nThe Best Model is: {best_model_name}")
print(f"Accuracy: {best_model_accuracy:.4f}")
```

```

The Best Model is: Logistic Regression
Accuracy: 0.9825
```

HYPERPARAMETER TUNING WITH GRIDSEARCH CV

```
# Define the Logistic Regression model
logreg = LogisticRegression(max_iter=1000)

# Define the parameter grid
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],         # Regularization strength
    'penalty': ['l1', 'l2', 'elasticnet'], # Regularization type
    'solver': ['liblinear', 'saga']        # Solver choice
}

# Perform GridSearchCV
grid_search = GridSearchCV(estimator=logreg, param_grid=param_grid,
scoring='accuracy', cv=5, verbose=2)
grid_search.fit(x_train, y_train)

# Best parameters and best score
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validated Accuracy:", grid_search.best_score_)
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
[CV] END ...............C=0.01, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END ...............C=0.01, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END ...............C=0.01, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END ...............C=0.01, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END ...............C=0.01, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END ....................C=0.01, penalty=l1, solver=saga; total
time=   0.0s
[CV] END ....................C=0.01, penalty=l1, solver=saga; total
time=   0.0s
[CV] END ....................C=0.01, penalty=l1, solver=saga; total
time=   0.0s
[CV] END ....................C=0.01, penalty=l1, solver=saga; total
time=   0.0s
[CV] END ....................C=0.01, penalty=l1, solver=saga; total
time=   0.0s
[CV] END ...............C=0.01, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END ...............C=0.01, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END ...............C=0.01, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END ...............C=0.01, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END ...............C=0.01, penalty=l2, solver=liblinear; total
```

```
time=   0.0s
[CV] END ....................C=0.01, penalty=l2, solver=saga; total
time=   0.0s
[CV] END ....................C=0.01, penalty=l2, solver=saga; total
time=   0.0s
[CV] END ....................C=0.01, penalty=l2, solver=saga; total
time=   0.0s
[CV] END ....................C=0.01, penalty=l2, solver=saga; total
time=   0.0s
[CV] END ....................C=0.01, penalty=l2, solver=saga; total
time=   0.0s
[CV] END .......C=0.01, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END .......C=0.01, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END .......C=0.01, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END .......C=0.01, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END .......C=0.01, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END ............C=0.01, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END ............C=0.01, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END ............C=0.01, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END ............C=0.01, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END ............C=0.01, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END ................C=0.1, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END ................C=0.1, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END ................C=0.1, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END ................C=0.1, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END ................C=0.1, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END .....................C=0.1, penalty=l1, solver=saga; total
time=   0.0s
[CV] END .....................C=0.1, penalty=l1, solver=saga; total
time=   0.0s
[CV] END .....................C=0.1, penalty=l1, solver=saga; total
time=   0.0s
[CV] END .....................C=0.1, penalty=l1, solver=saga; total
time=   0.0s
```

```
[CV] END .....................C=0.1, penalty=l1, solver=saga; total
time=   0.0s
[CV] END ................C=0.1, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END ................C=0.1, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END ................C=0.1, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END ................C=0.1, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END ................C=0.1, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END .....................C=0.1, penalty=l2, solver=saga; total
time=   0.0s
[CV] END .....................C=0.1, penalty=l2, solver=saga; total
time=   0.0s
[CV] END .....................C=0.1, penalty=l2, solver=saga; total
time=   0.0s
[CV] END .....................C=0.1, penalty=l2, solver=saga; total
time=   0.0s
[CV] END .....................C=0.1, penalty=l2, solver=saga; total
time=   0.0s
[CV] END ........C=0.1, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END ........C=0.1, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END ........C=0.1, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END ........C=0.1, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END ........C=0.1, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END .............C=0.1, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END .............C=0.1, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END .............C=0.1, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END .............C=0.1, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END .............C=0.1, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END ..................C=1, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END ..................C=1, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END ..................C=1, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END ..................C=1, penalty=l1, solver=liblinear; total
```

```
time=   0.0s
[CV] END ..................C=1, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END .......................C=1, penalty=l1, solver=saga; total
time=   0.0s
[CV] END .......................C=1, penalty=l1, solver=saga; total
time=   0.0s
[CV] END .......................C=1, penalty=l1, solver=saga; total
time=   0.0s
[CV] END .......................C=1, penalty=l1, solver=saga; total
time=   0.0s
[CV] END .......................C=1, penalty=l1, solver=saga; total
time=   0.0s
[CV] END ..................C=1, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END ..................C=1, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END ..................C=1, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END ..................C=1, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END ..................C=1, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END .......................C=1, penalty=l2, solver=saga; total
time=   0.0s
[CV] END .......................C=1, penalty=l2, solver=saga; total
time=   0.0s
[CV] END .......................C=1, penalty=l2, solver=saga; total
time=   0.0s
[CV] END .......................C=1, penalty=l2, solver=saga; total
time=   0.0s
[CV] END .......................C=1, penalty=l2, solver=saga; total
time=   0.0s
[CV] END ..........C=1, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END ..........C=1, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END ..........C=1, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END ..........C=1, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END ..........C=1, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END ...............C=1, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END ...............C=1, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END ...............C=1, penalty=elasticnet, solver=saga; total
time=   0.0s
```

```
[CV] END ...............C=1, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END ...............C=1, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END .................C=10, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END .................C=10, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END .................C=10, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END .................C=10, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END .................C=10, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END ......................C=10, penalty=l1, solver=saga; total
time=   0.0s
[CV] END ......................C=10, penalty=l1, solver=saga; total
time=   0.0s
[CV] END ......................C=10, penalty=l1, solver=saga; total
time=   0.0s
[CV] END ......................C=10, penalty=l1, solver=saga; total
time=   0.0s
[CV] END ......................C=10, penalty=l1, solver=saga; total
time=   0.0s
[CV] END .................C=10, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END .................C=10, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END .................C=10, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END .................C=10, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END .................C=10, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END ......................C=10, penalty=l2, solver=saga; total
time=   0.0s
[CV] END ......................C=10, penalty=l2, solver=saga; total
time=   0.0s
[CV] END ......................C=10, penalty=l2, solver=saga; total
time=   0.0s
[CV] END ......................C=10, penalty=l2, solver=saga; total
time=   0.0s
[CV] END ......................C=10, penalty=l2, solver=saga; total
time=   0.0s
[CV] END .........C=10, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END .........C=10, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END .........C=10, penalty=elasticnet, solver=liblinear; total
```

```
time=   0.0s
[CV] END .........C=10, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END .........C=10, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END ..............C=10, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END ..............C=10, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END ..............C=10, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END ..............C=10, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END ..............C=10, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END ................C=100, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END ................C=100, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END ................C=100, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END ................C=100, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END ................C=100, penalty=l1, solver=liblinear; total
time=   0.0s
[CV] END .....................C=100, penalty=l1, solver=saga; total
time=   0.0s
[CV] END .....................C=100, penalty=l1, solver=saga; total
time=   0.0s
[CV] END .....................C=100, penalty=l1, solver=saga; total
time=   0.0s
[CV] END .....................C=100, penalty=l1, solver=saga; total
time=   0.0s
[CV] END .....................C=100, penalty=l1, solver=saga; total
time=   0.0s
[CV] END ................C=100, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END ................C=100, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END ................C=100, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END ................C=100, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END ................C=100, penalty=l2, solver=liblinear; total
time=   0.0s
[CV] END .....................C=100, penalty=l2, solver=saga; total
time=   0.0s
[CV] END .....................C=100, penalty=l2, solver=saga; total
time=   0.0s
```

```
[CV] END .....................C=100, penalty=l2, solver=saga; total
time=   0.0s
[CV] END .....................C=100, penalty=l2, solver=saga; total
time=   0.0s
[CV] END .....................C=100, penalty=l2, solver=saga; total
time=   0.0s
[CV] END ........C=100, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END ........C=100, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END ........C=100, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END ........C=100, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END ........C=100, penalty=elasticnet, solver=liblinear; total
time=   0.0s
[CV] END .............C=100, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END .............C=100, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END .............C=100, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END .............C=100, penalty=elasticnet, solver=saga; total
time=   0.0s
[CV] END .............C=100, penalty=elasticnet, solver=saga; total
time=   0.0s
Best Parameters: {'C': 10, 'penalty': 'l2', 'solver': 'saga'}
Best Cross-Validated Accuracy: 0.9758241758241759
```

```
# Evaluate the best model on the test set
best_logreg_model = grid_search.best_estimator_
test_accuracy = best_logreg_model.score(x_test, y_test)
print("Test Accuracy of Best Logistic Regression Model:",
test_accuracy)
```

```
Test Accuracy of Best Logistic Regression Model: 0.9824561403508771
```

SAVING THE MODEL

```
# Save the tuned logistic regression model
joblib.dump(best_logreg_model,
'classification_lr_model(breast_cancer).joblib')
print("Model saved as classification_lr_model(breast_cancer).joblib.")
```

```
Model saved as classification_lr_model(breast_cancer).joblib.
```