Breast Cancer Classification using logistic Regression and Neural Network

Dataset used: Breast Cancer Dataset from the Sklearn library

```
# Importing the dependencies
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder # Encode the labels
import sklearn.datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Data collection - loading the data from sklearn
breast_cancer_dataset = sklearn.datasets.load_breast_cancer()

print(breast_cancer_dataset)

{'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.. dropdown:: References\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',
'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'}
```

Loading the data into a Pandas dataframe

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

# Distribution of the target variable: Diagnosis

#df['diagnosis'].value_counts() # There is an imbalance in the data,
but as much

# Adding the target column
df['label'] = breast_cancer_dataset.target
```

Exploratory Data Analysis

```
# Printing the first 5 rows of the dataset
df.head()

{"type":"dataframe","variable_name":"df"}

# Printing the last 5 rows of the dataset
df.tail()
```

```
{"type":"dataframe"}

# Removing the last column as it does not contain any data inplace

#df.drop(columns = "Unnamed: 32", axis = 1, inplace = True)
# To drop a column: Axis = 1
# To drop a row: Axis = 0

df.shape # We have 32 features and 569 samples

(569, 31)

# Removing the column 'id' as it is not necessary for the analysis

#df.drop(columns = "id", axis = 1, inplace = True)

# Encoding the target variable
#label_encoder = LabelEncoder()
#labels = label_encoder.fit_transform(df['diagnosis'])

# Creating a new column for label
#df['target'] = labels

# Dropping the diagnosis column
#df.drop(columns = 'diagnosis', axis = 1, inplace = True)

df.info() # The dataset does not contain any null values

<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  label                    569 non-null    int64
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
```

The "label" columns is CATEGORICAL has it contains objects which is converted to 0s and 1s to be processed while the other features contain continuous values.

```
df.isnull().sum() # To find if there are any null values

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
label                      0
dtype: int64
```

```
# Descriptive Statistical measures of the dataset
df.describe() # The fuction is not carried out for the categorical
feature
```

{"type":"dataframe"}

- 25% of the dataset has a values less than 11.7 in the radius_mean.
- 50% is the median value - middle value after sorting the value

In some cases the mean > median, **right-skewed or positively-skewed data**. And we would need to transform the data into a normally distributed data to make sure that the model makes right predictions.

```
df['label'].value_counts()

# One class has more sample than the other.
```

```
label
1    357
0    212
Name: count, dtype: int64
```

```
# Finding the mean values of each feature for both the targets.
df.groupby('label').mean()
```

{"type":"dataframe"}

We identify that: 0 -> Benign 1 -> Malignant

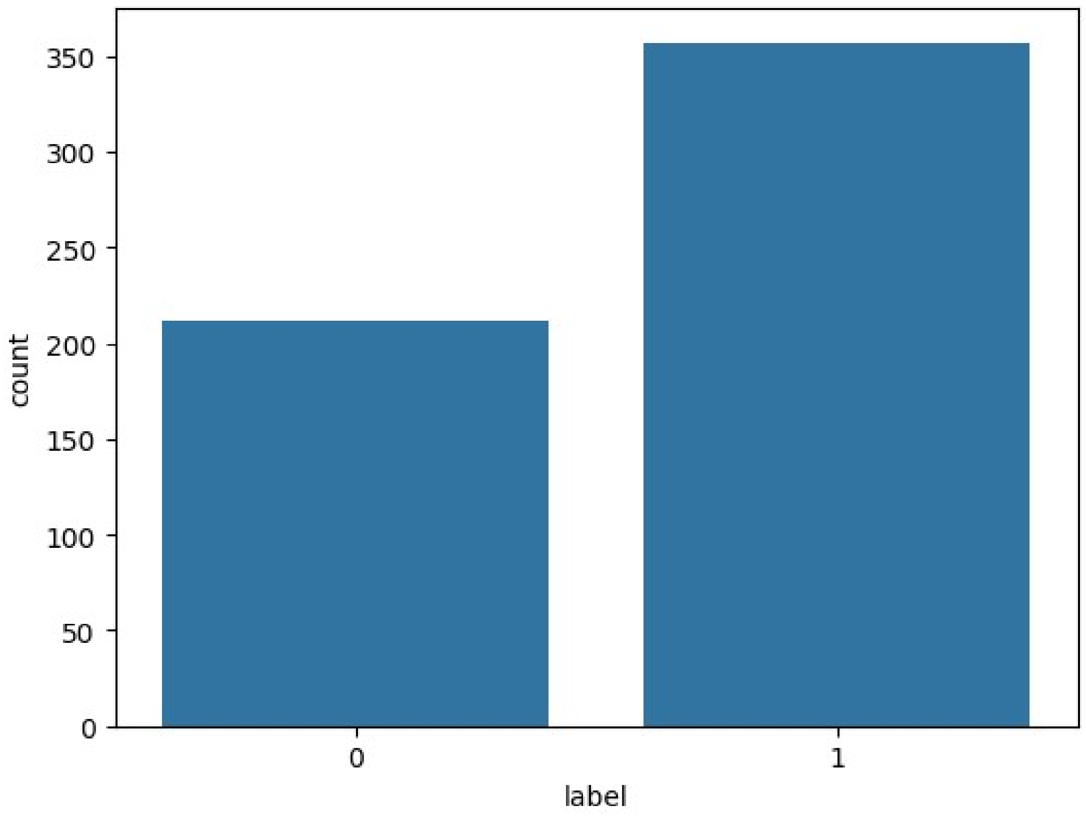
The mean values of all the features with the target as Malignant are more than that of the mean with the target as Benign.

Data Visualisation

- Finding the distribution of the features
- Checking for outliers in the dataset
- Correlation between the features

```
import seaborn as sns
import matplotlib.pyplot as plt

# Countplot for the target column
sns.countplot(x = 'label', data = df)
```

```
<Axes: xlabel='label', ylabel='count'>
```



Understanding the distribution of features in the dataset

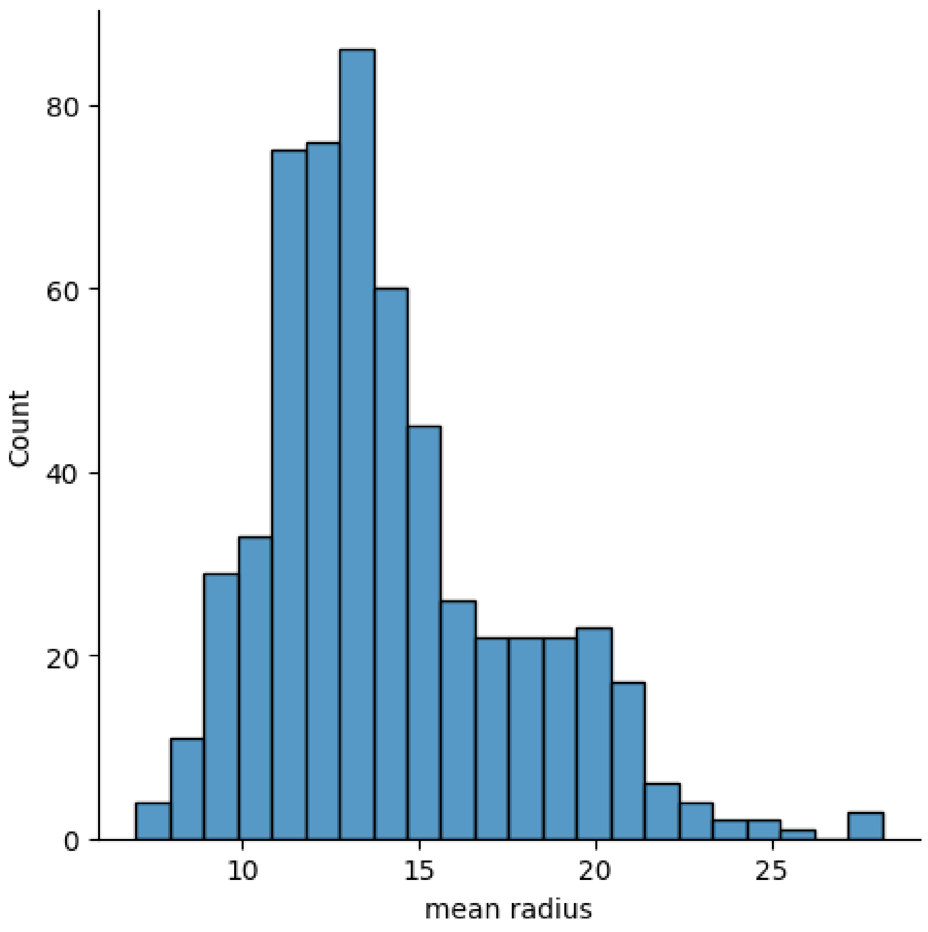
```
# Getting all the column in the dataset
for column in df:
    print(column)
```

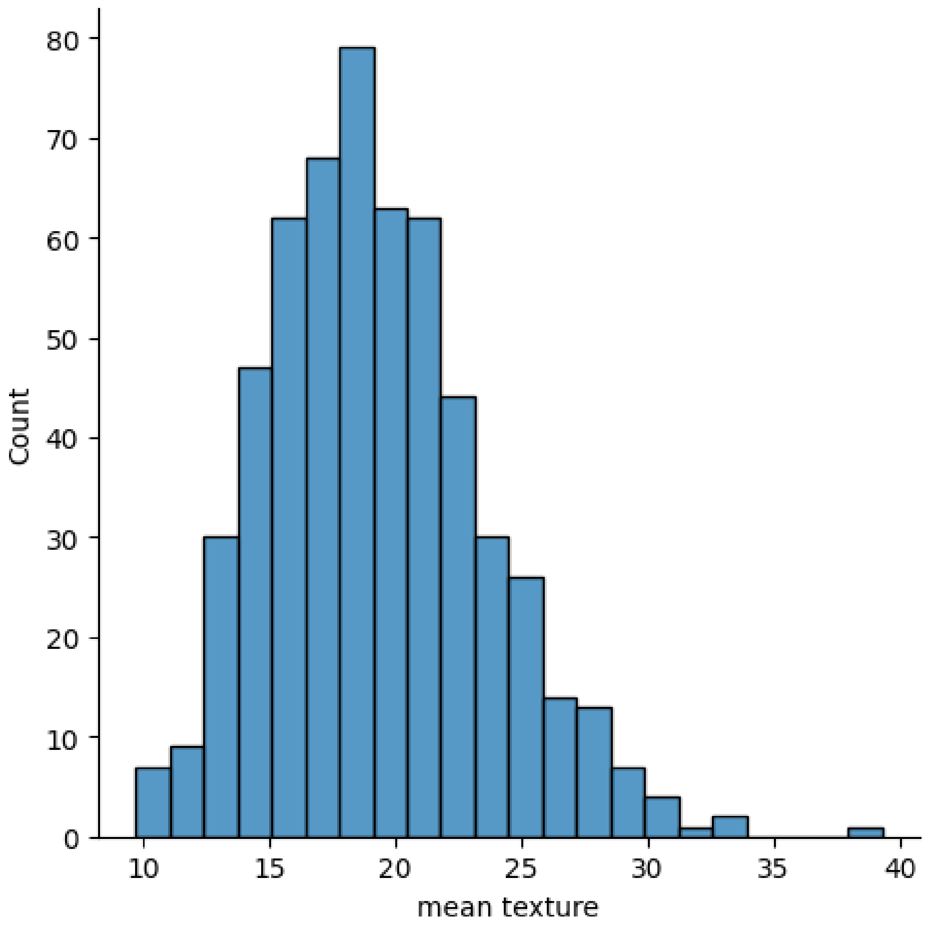
```
mean radius
mean texture
mean perimeter
mean area
mean smoothness
mean compactness
mean concavity
mean concave points
mean symmetry
mean fractal dimension
radius error
texture error
perimeter error
area error
smoothness error
compactness error
concavity error
concave points error
symmetry error
```

```
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
label
```

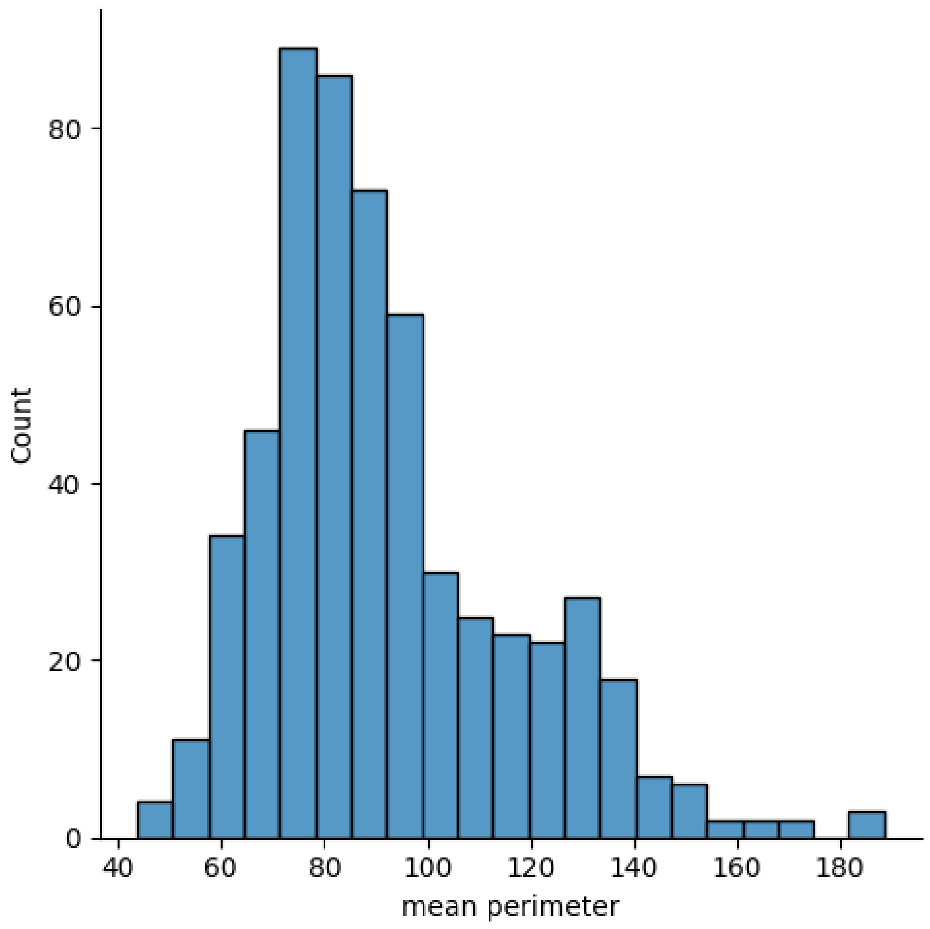
```python
# Univariant analysis - Taking one feature and analysing the
distribution at a time
for column in df:
    sns.displot(x = column, data = df)
```

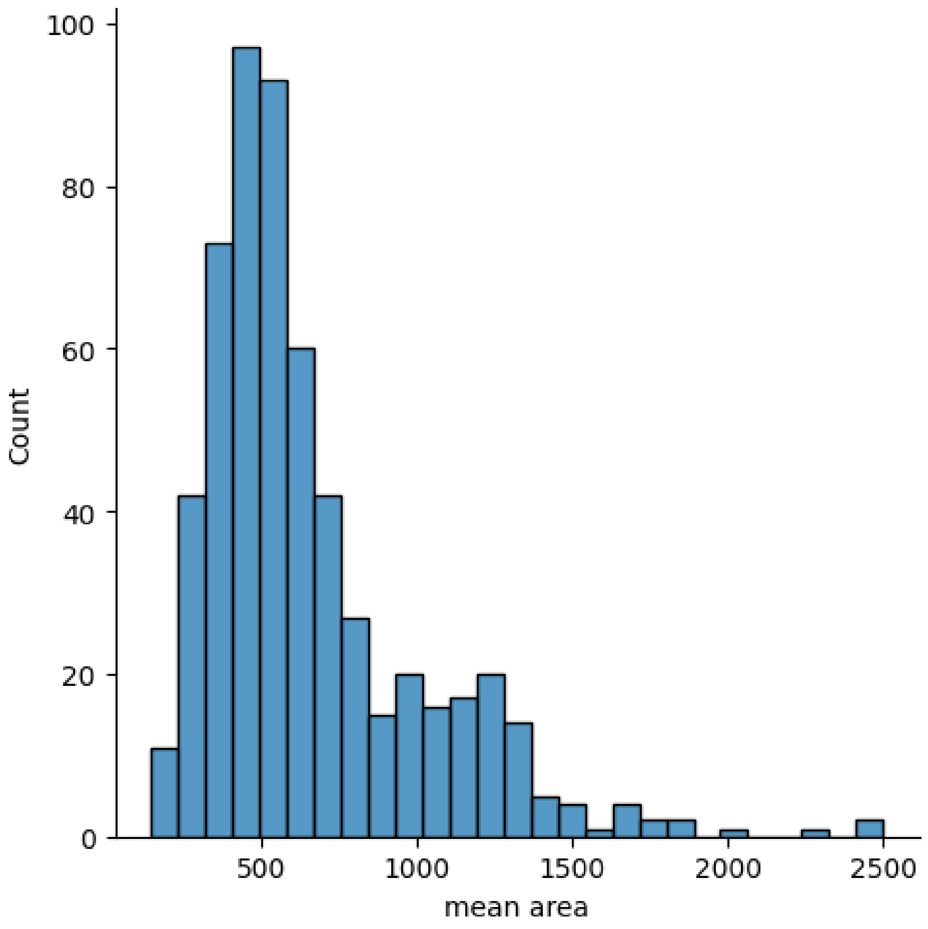
```
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:453:
RuntimeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retained
until explicitly closed and may consume too much memory. (To control
this warning, see the rcParam `figure.max_open_warning`). Consider
using `matplotlib.pyplot.close()`.
  fig = plt.figure(figsize=figsize)
```


Count
0
20
40
60
80
10
15
20
25
mean radius

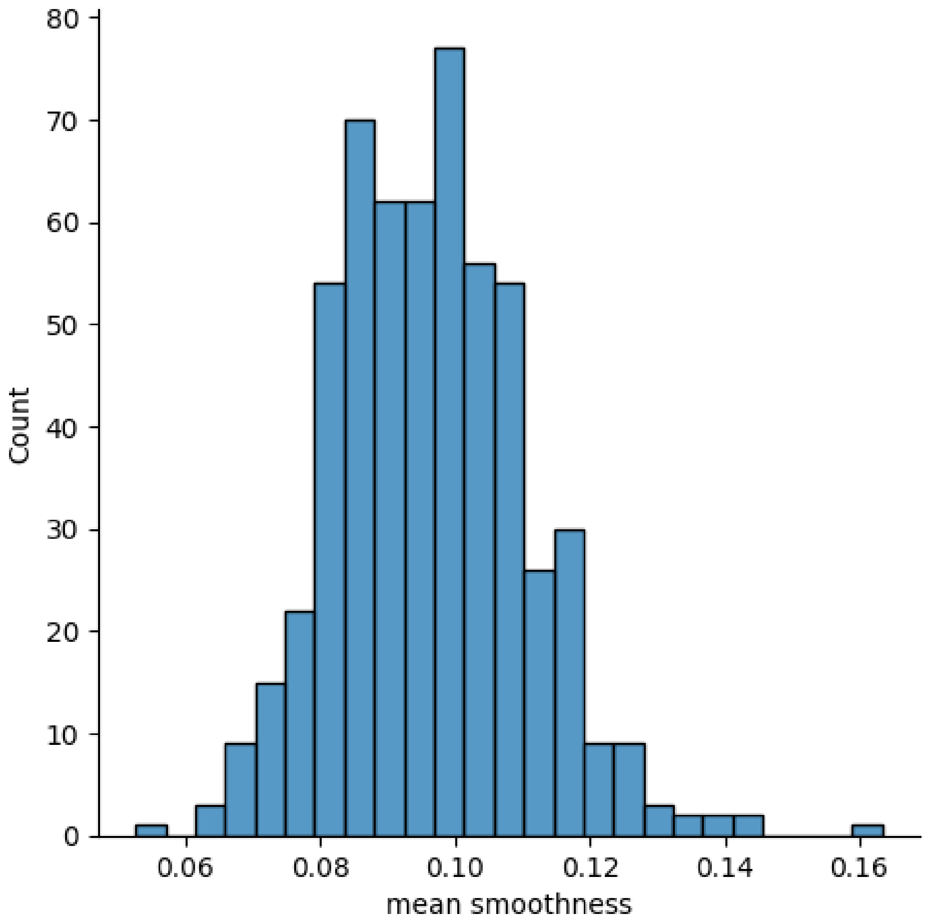


Count
mean texture
0
10
20
30
40
50
60
70
80
10
15
20
25
30
35
40

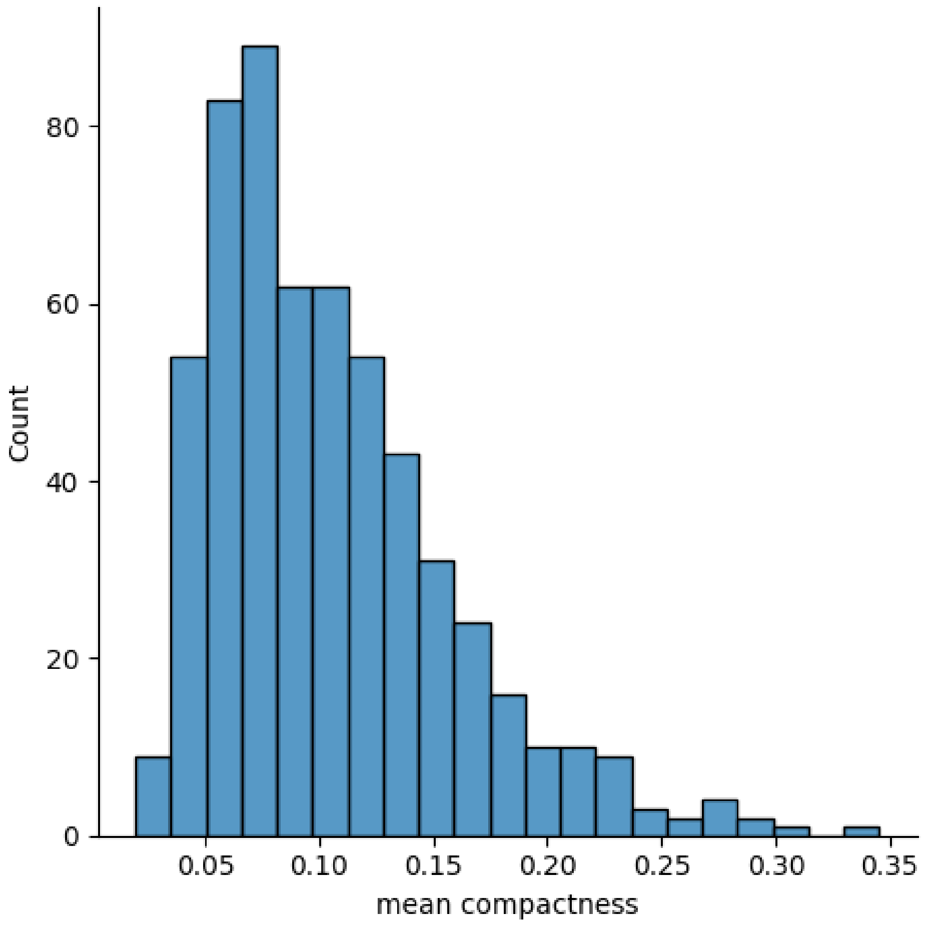

Count
0
20
40
60
80
40
60
80
100
120
140
160
180
mean perimeter

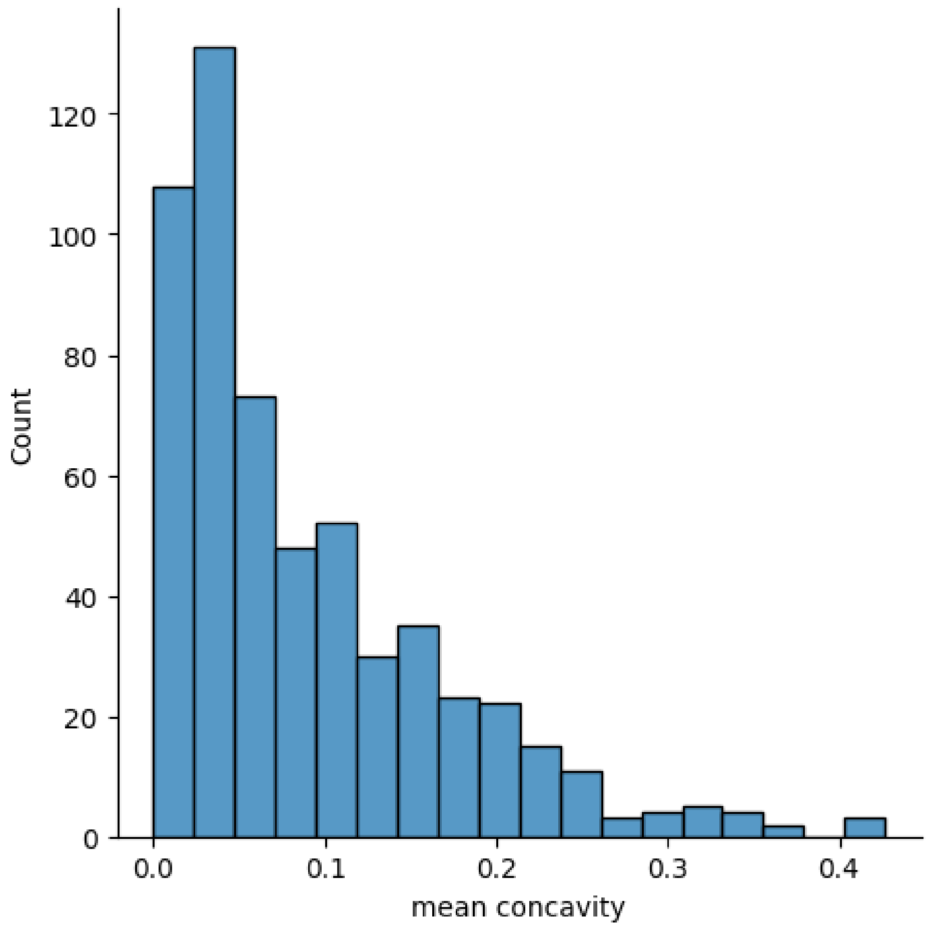


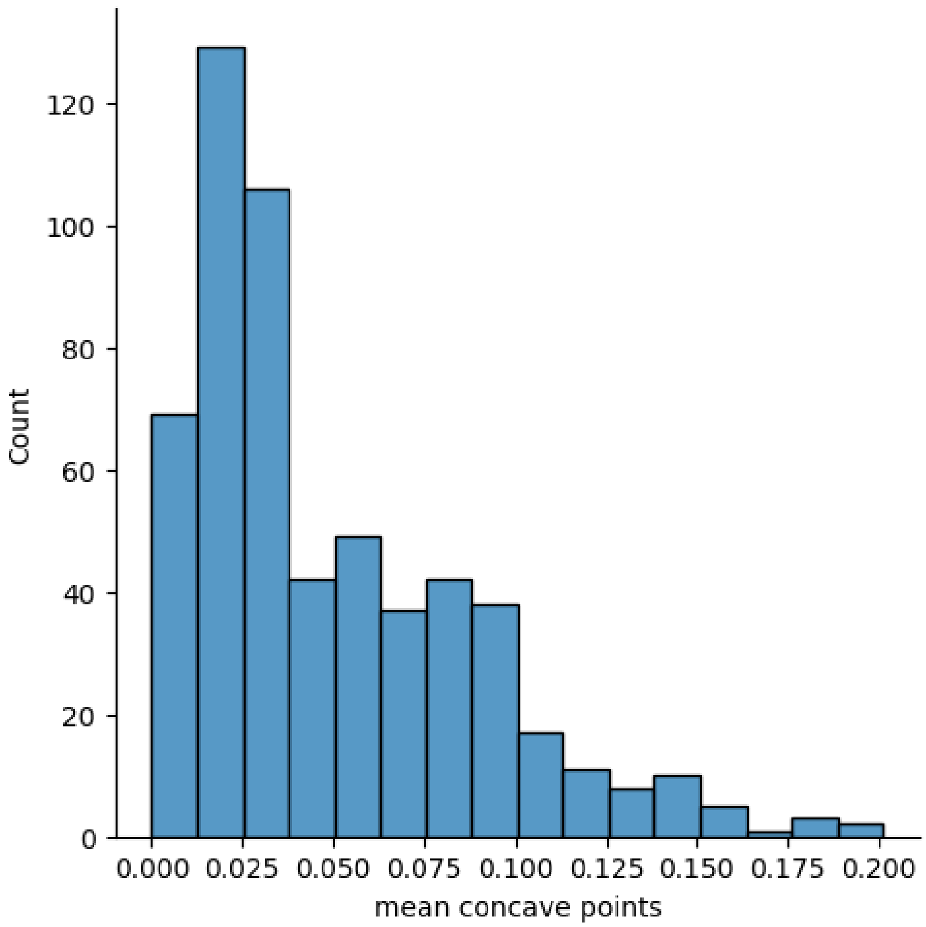
Count
100
80
60
40
20
0
500
1000
1500
2000
2500
mean area

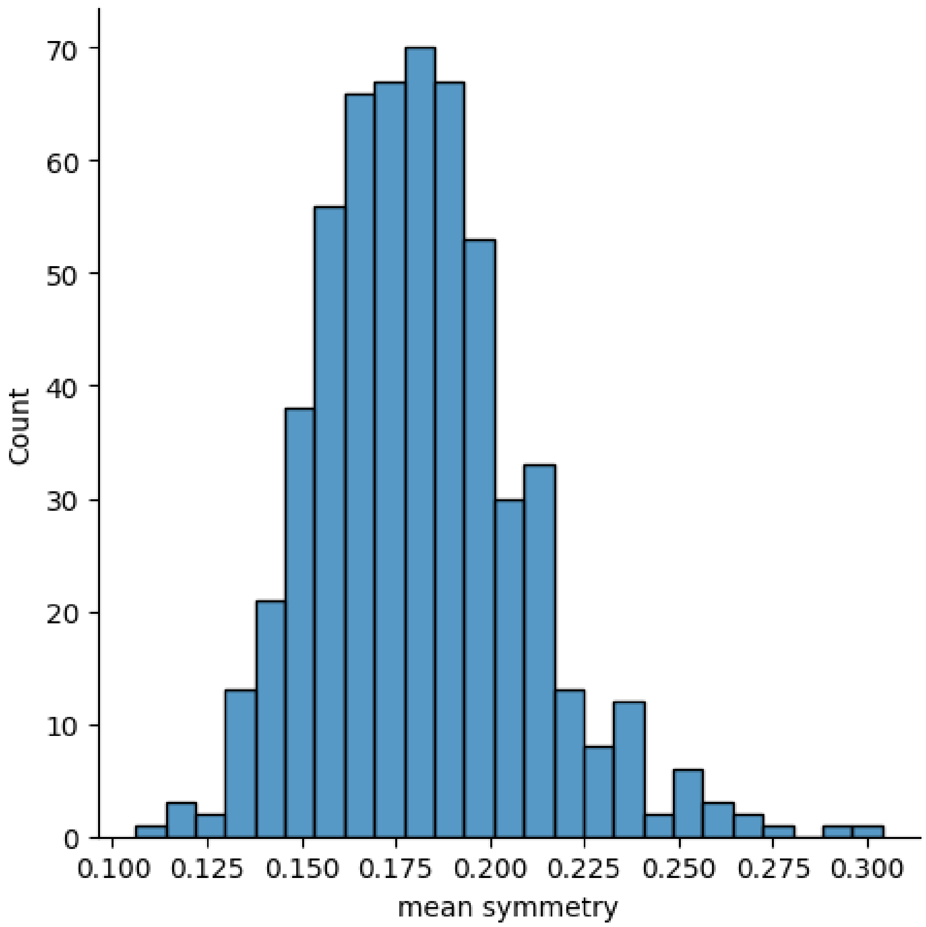


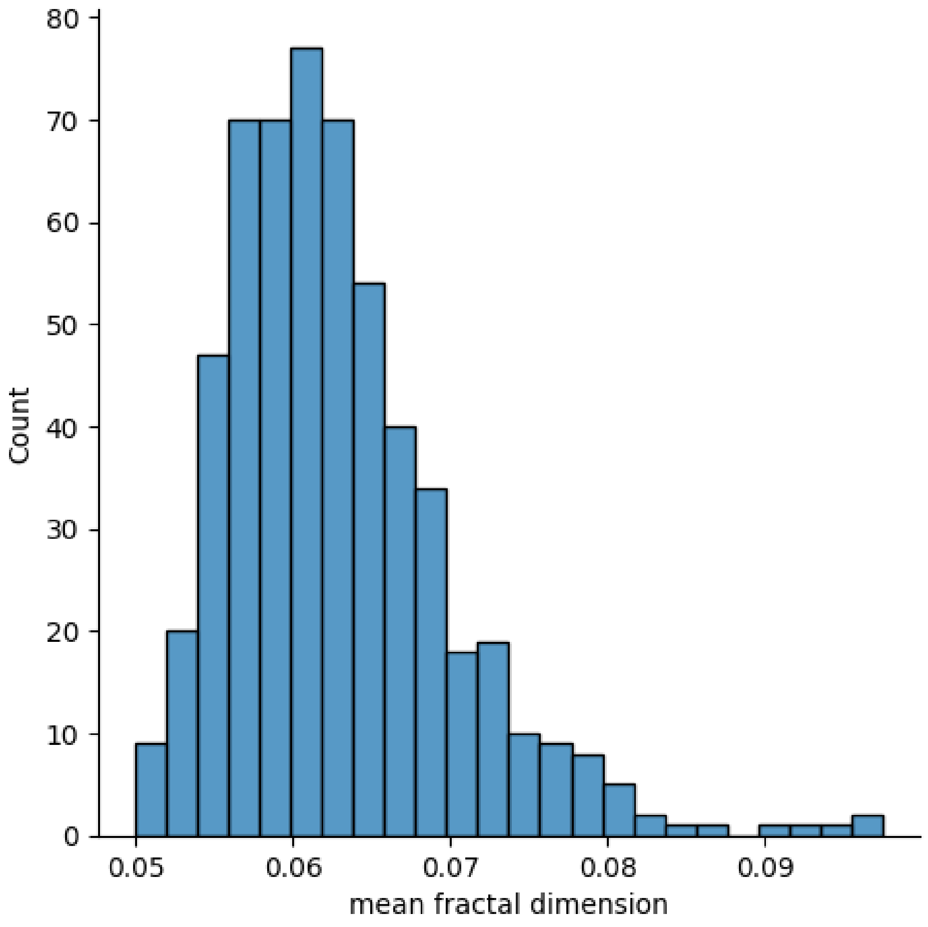
Count
mean smoothness
0
10
20
30
40
50
60
70
80
0.06
0.08
0.10
0.12
0.14
0.16

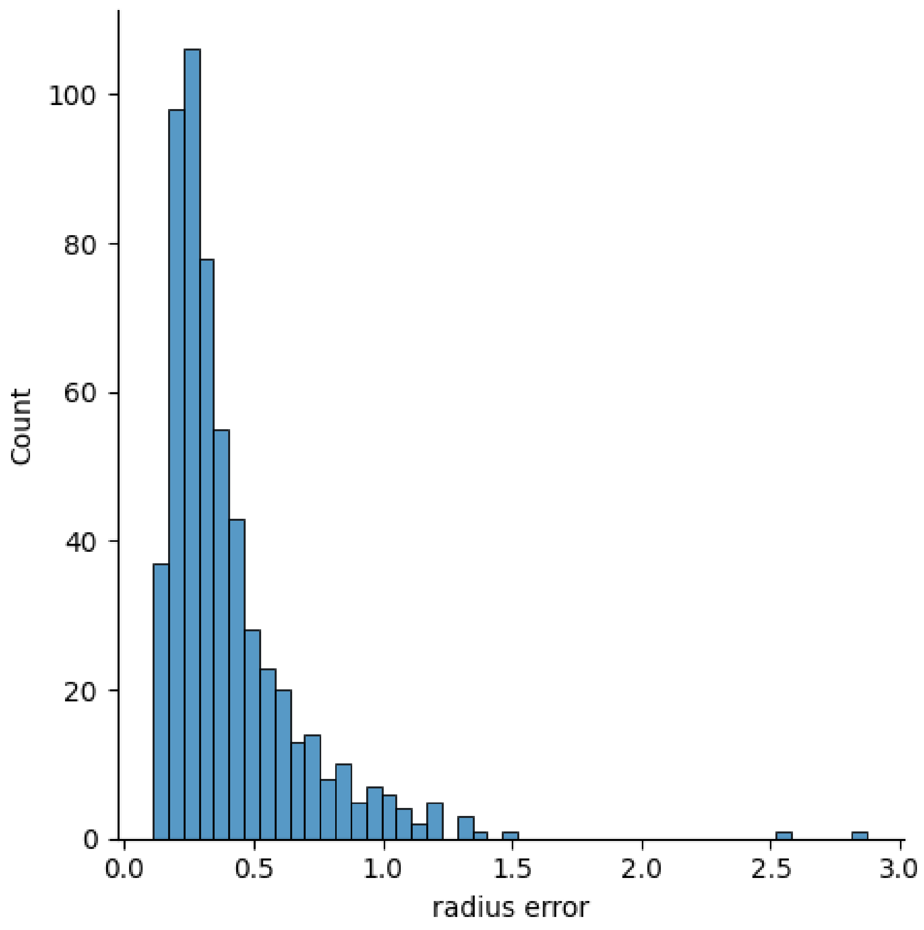

Count
0
20
40
60
80
0.05
0.10
0.15
0.20
0.25
0.30
0.35
mean compactness

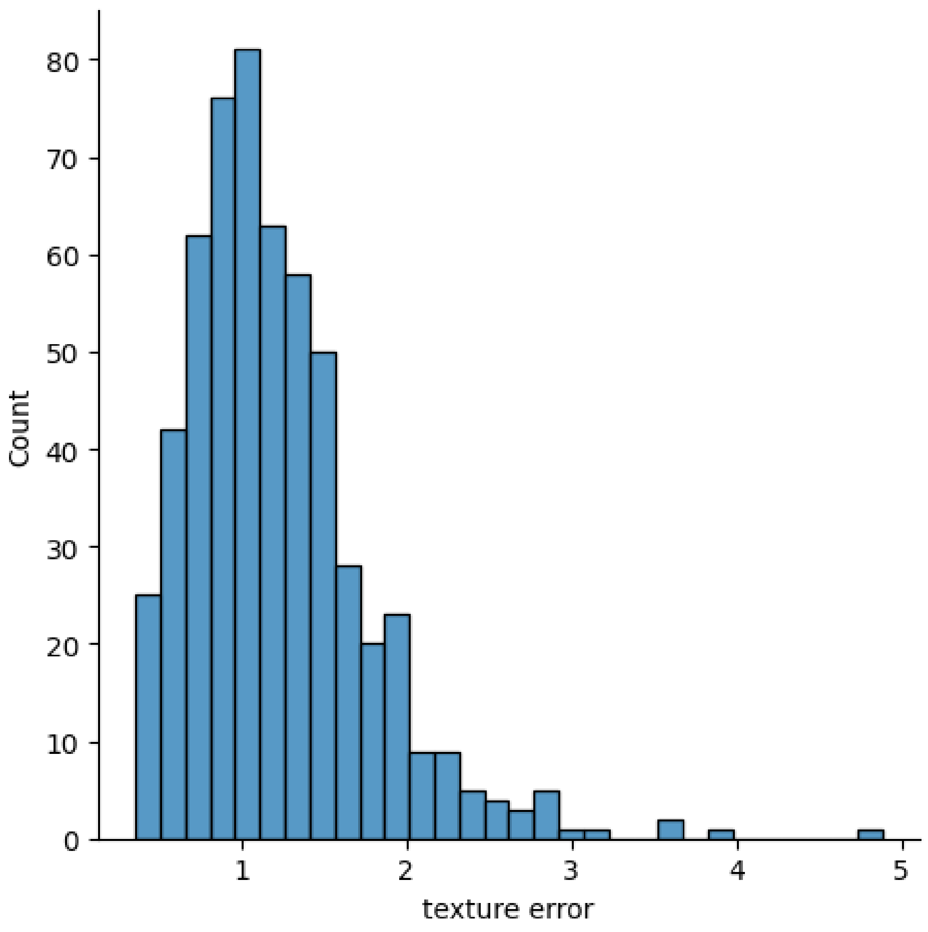

Count
0
20
40
60
80
100
120
0.0
0.1
0.2
0.3
0.4
mean concavity


Count
0
20
40
60
80
100
120
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
mean concave points

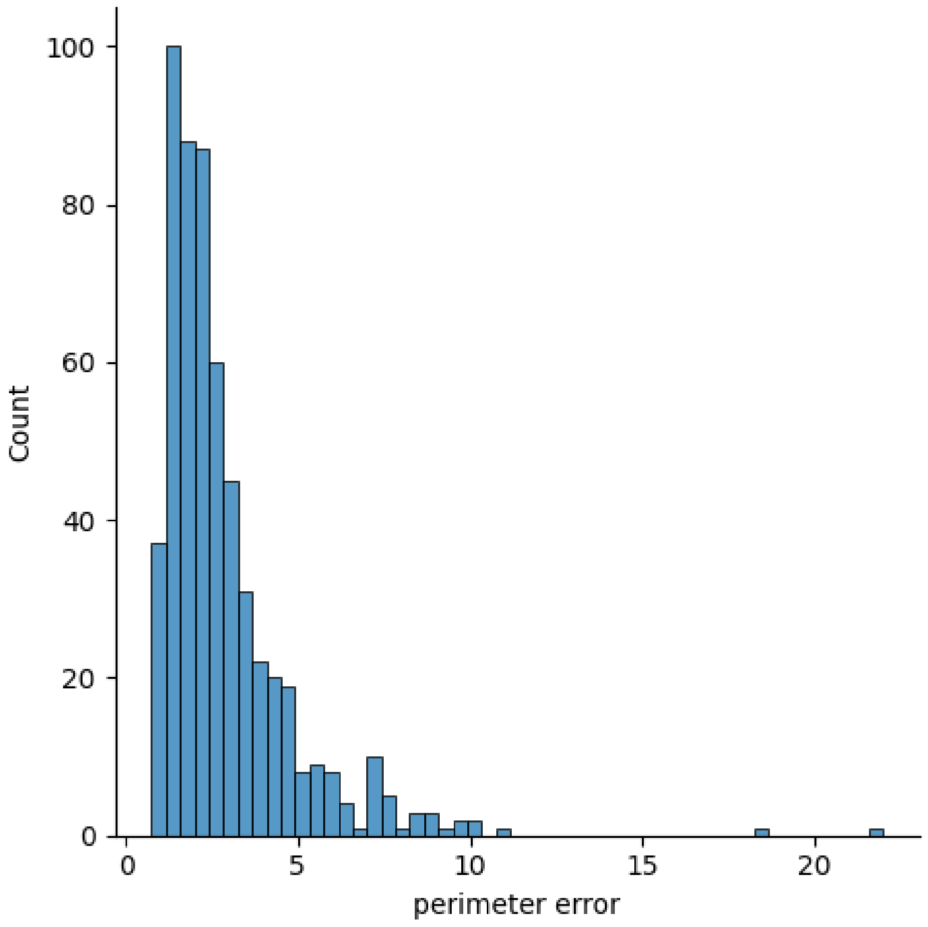

Count
0
10
20
30
40
50
60
70
0.100
0.125
0.150
0.175
0.200
0.225
0.250
0.275
0.300
mean symmetry

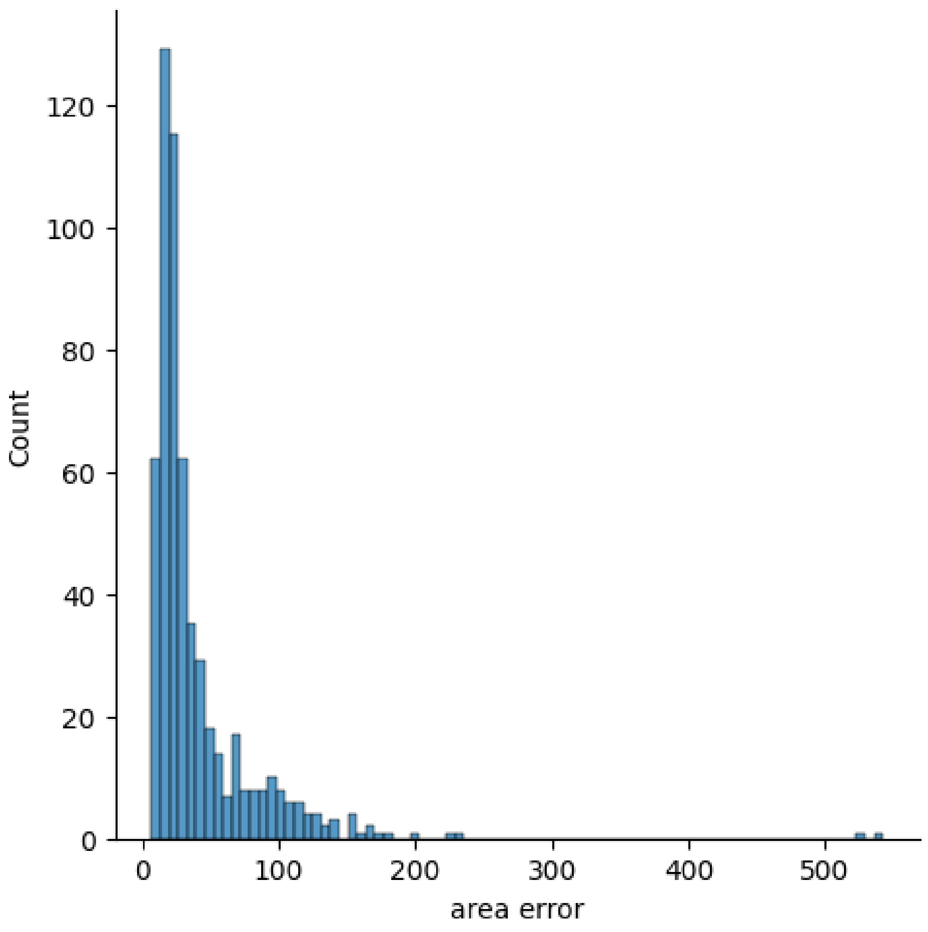

Count
0
10
20
30
40
50
60
70
80
0.05
0.06
0.07
0.08
0.09
mean fractal dimension


Count
0
20
40
60
80
100
0.0
0.5
1.0
1.5
2.0
2.5
3.0
radius error

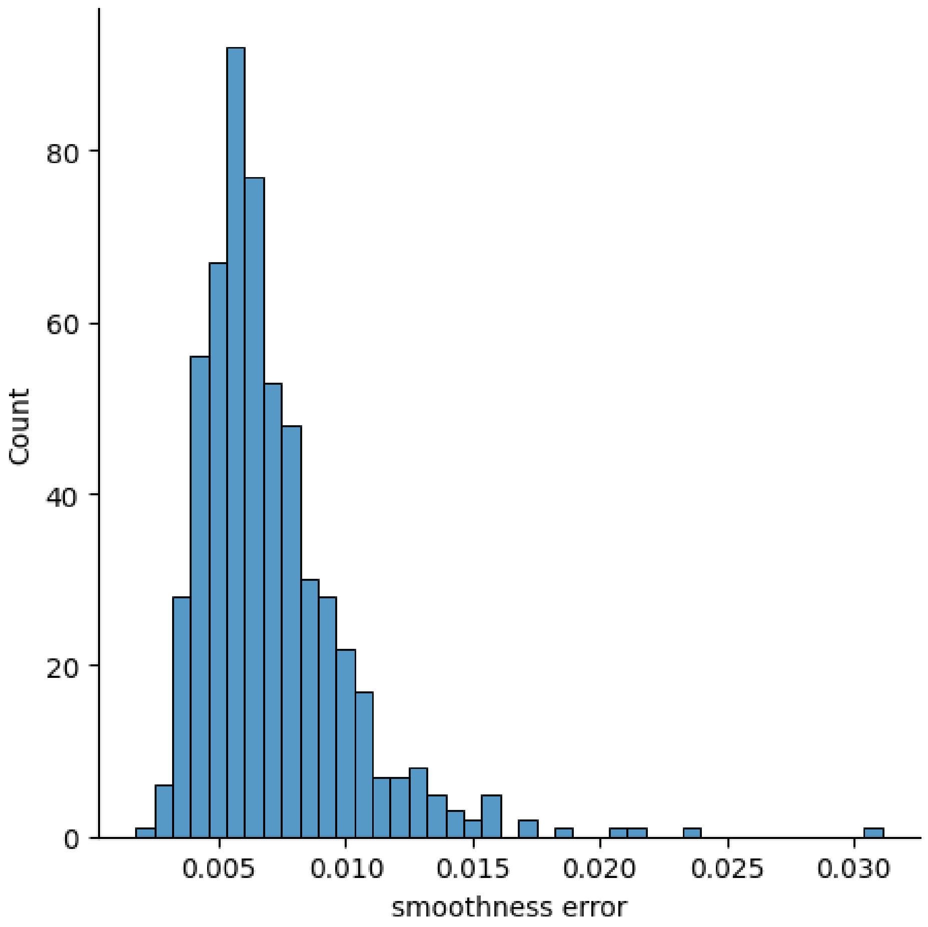


Count
0
10
20
30
40
50
60
70
80
1
2
3
4
5
texture error

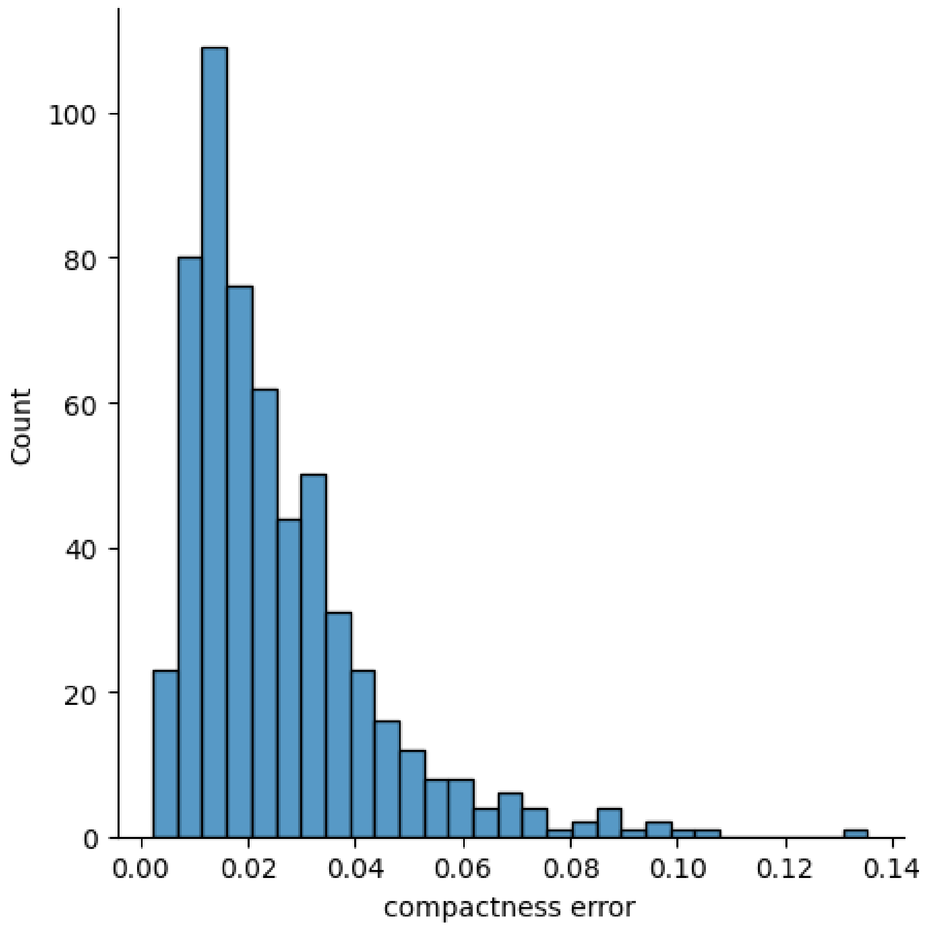

Count
100
80
60
40
20
0
0
5
10
15
20
perimeter error

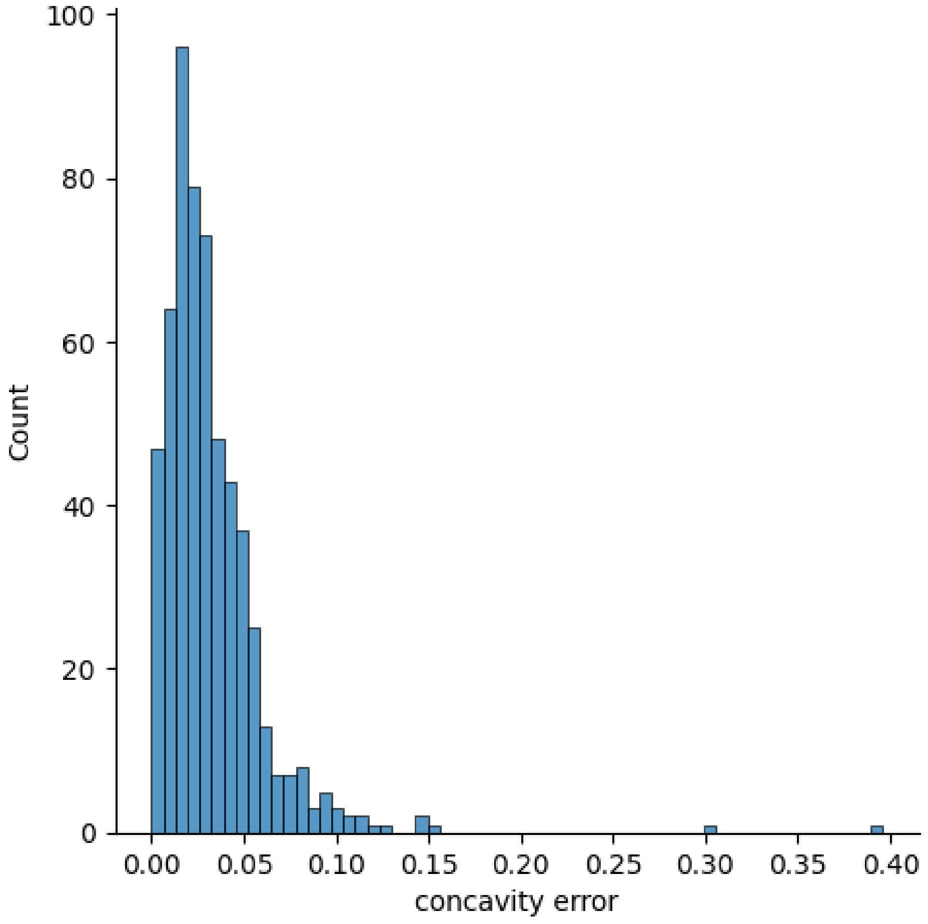


Count
120
100
80
60
40
20
0
0
100
200
300
400
500
area error

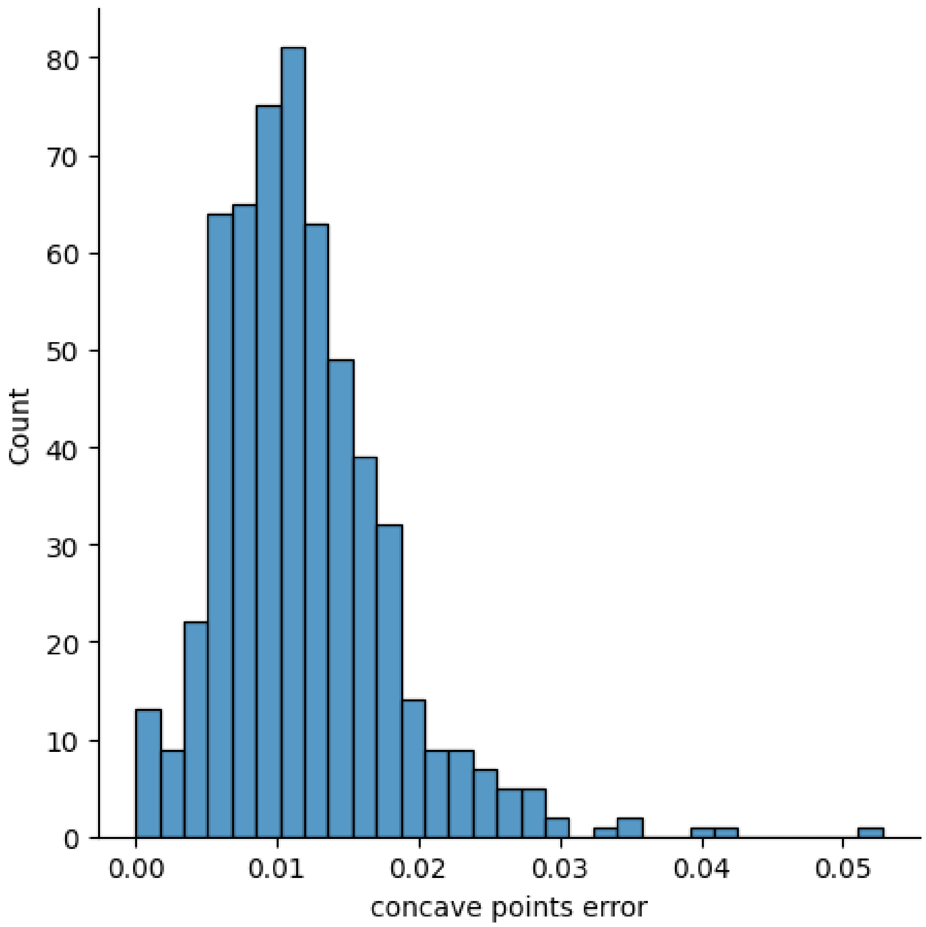


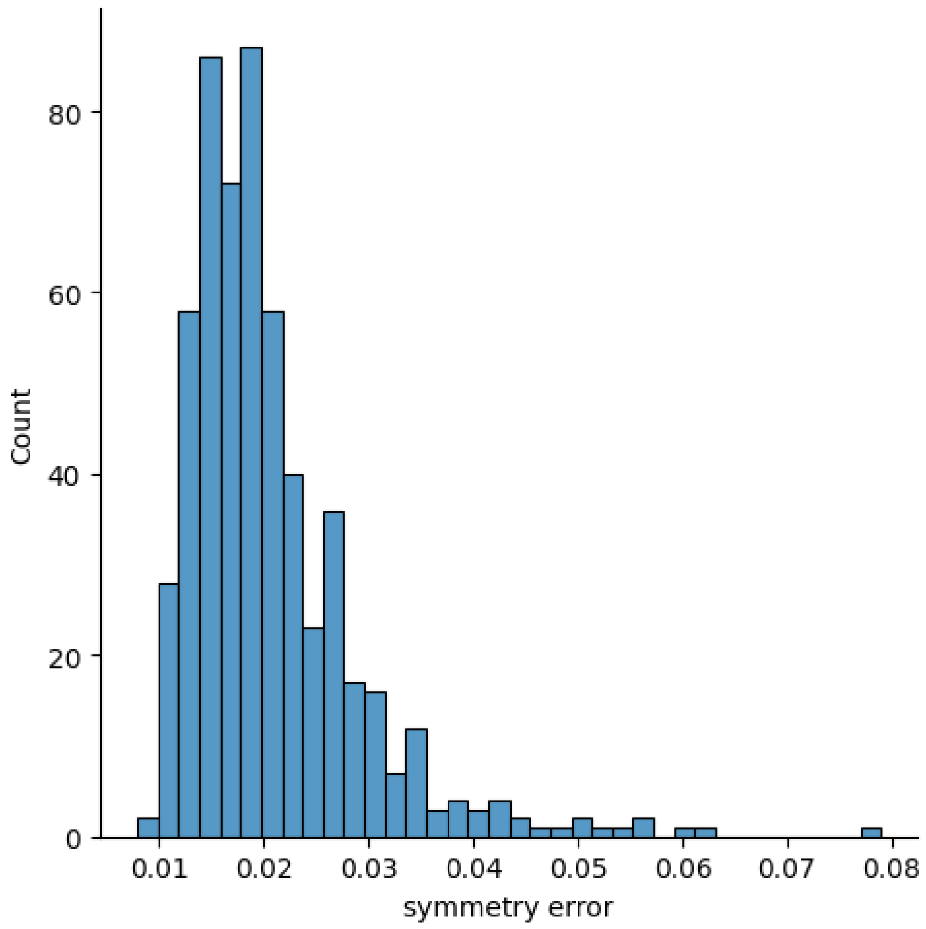
Count
0
20
40
60
80
0.005
0.010
0.015
0.020
0.025
0.030
smoothness error

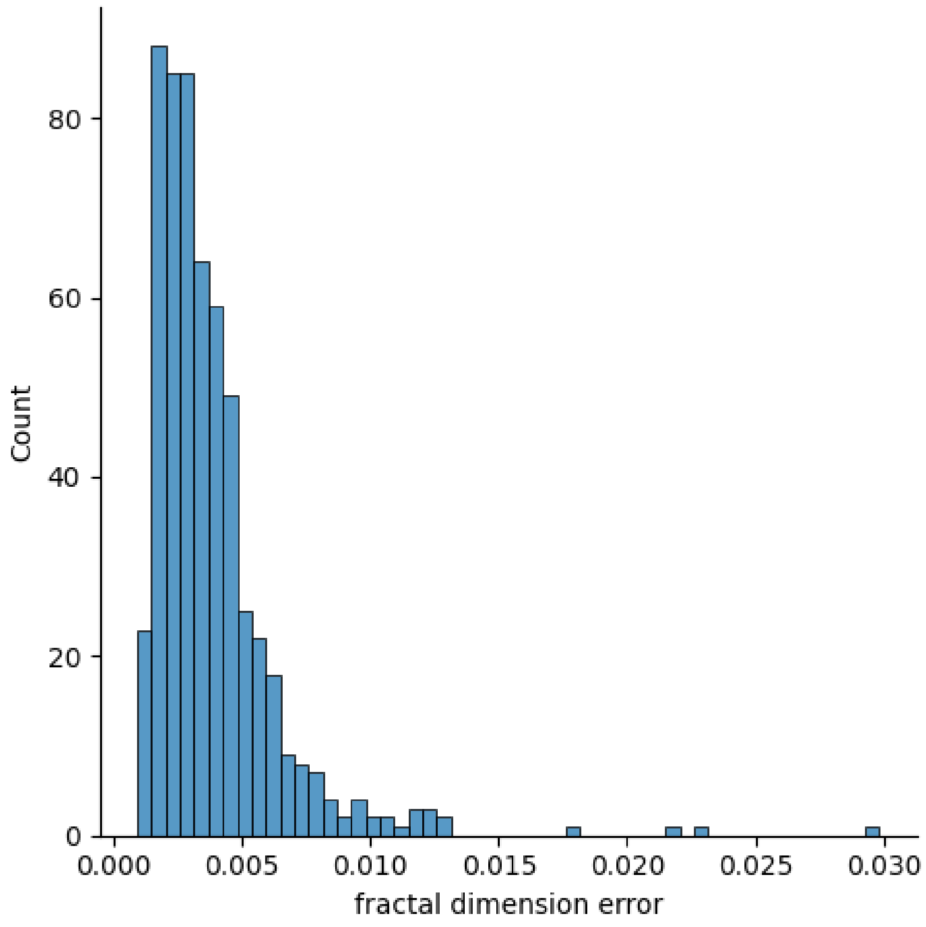

Count
100
80
60
40
20
0
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
compactness error

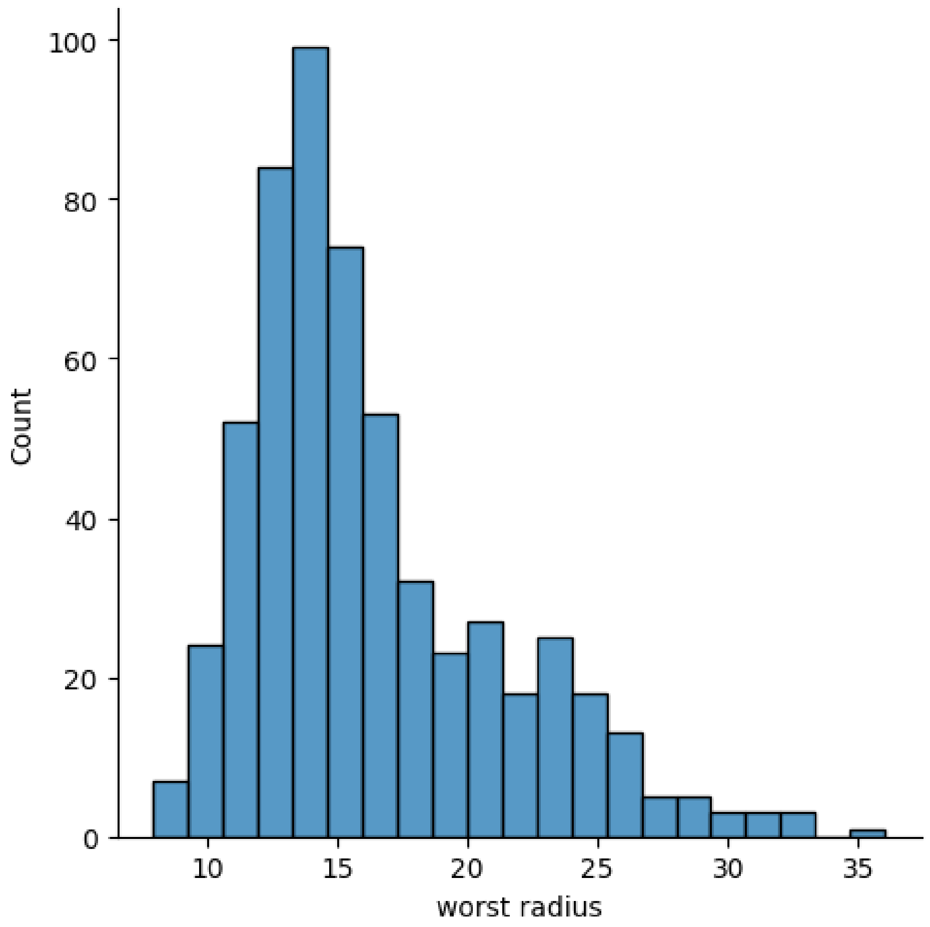


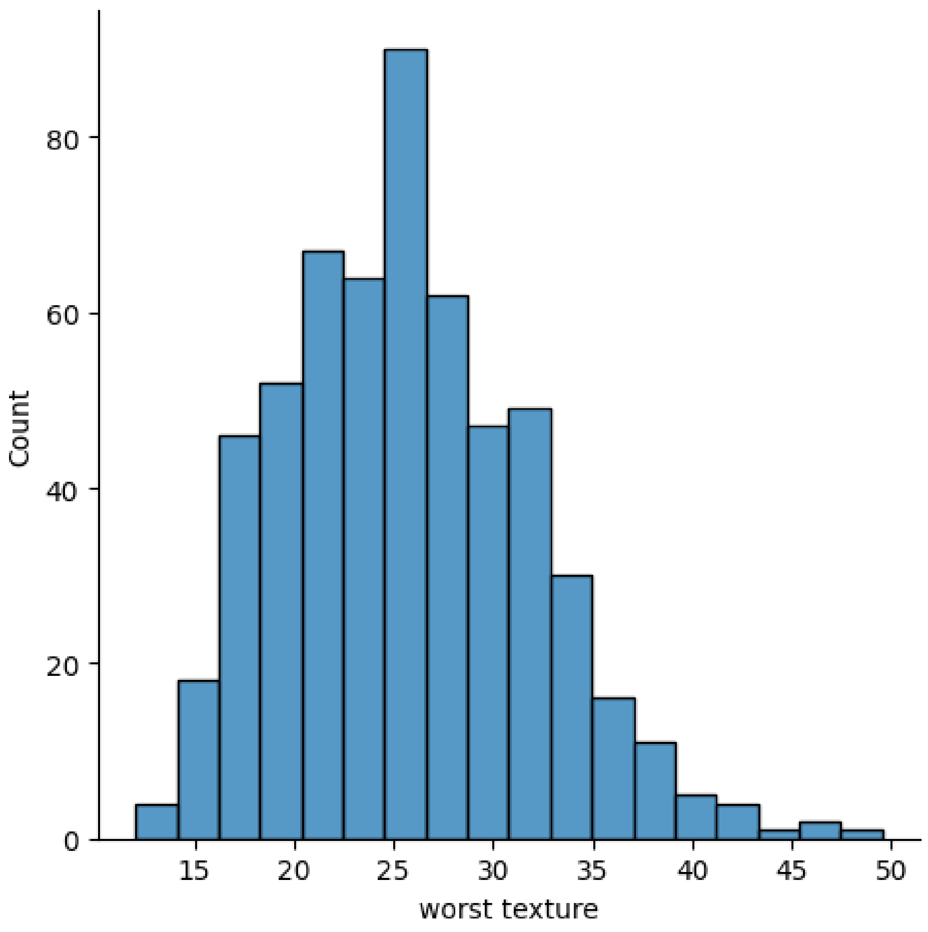
Count
100
80
60
40
20
0
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
concavity error

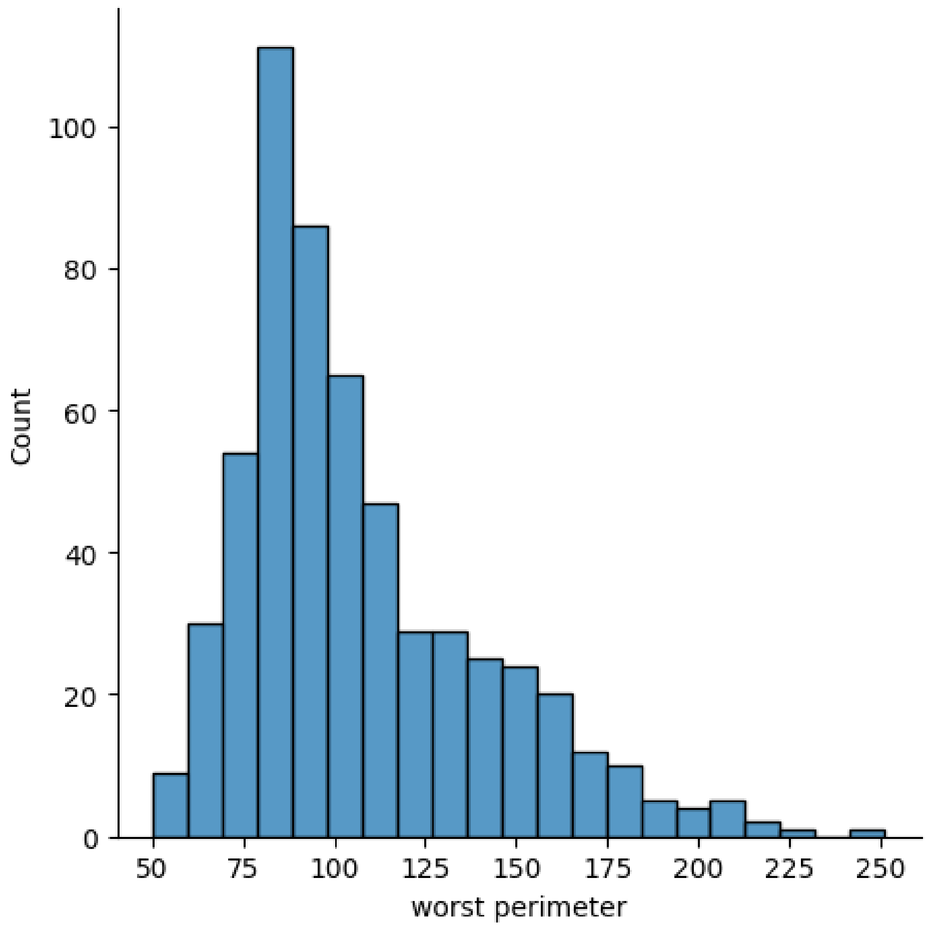

Count
0
10
20
30
40
50
60
70
80
0.00
0.01
0.02
0.03
0.04
0.05
concave points error

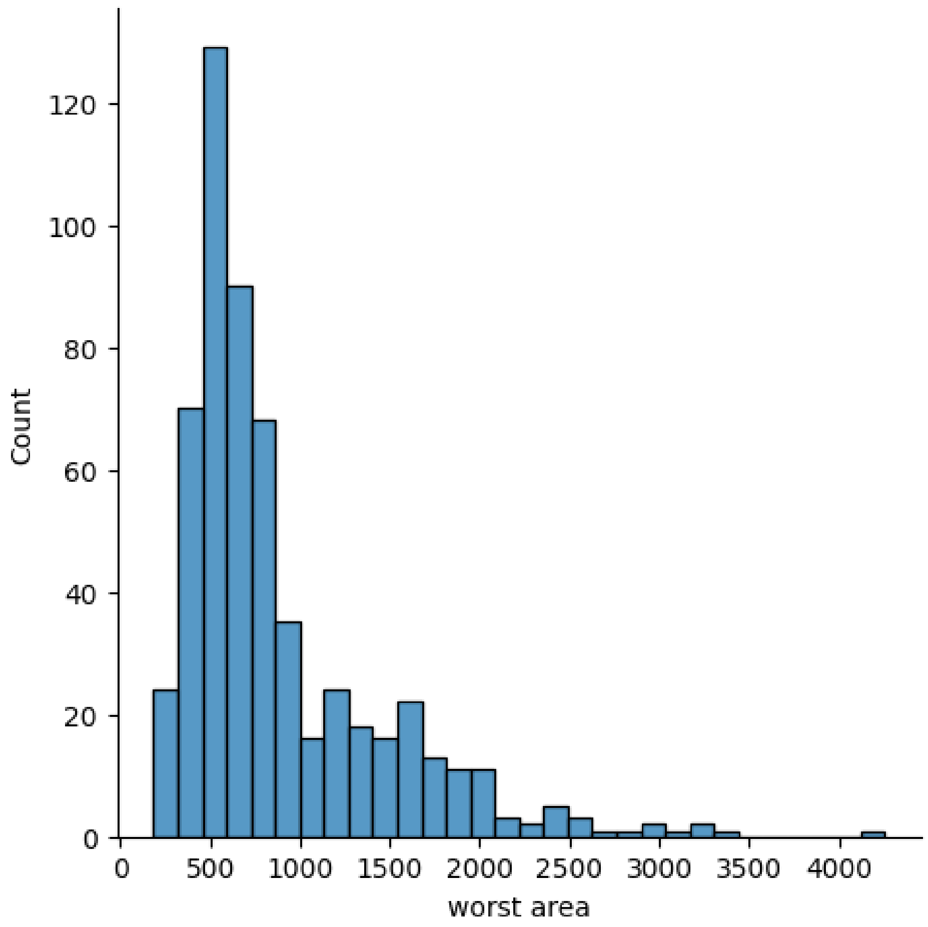

Count
80
60
40
20
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
symmetry error

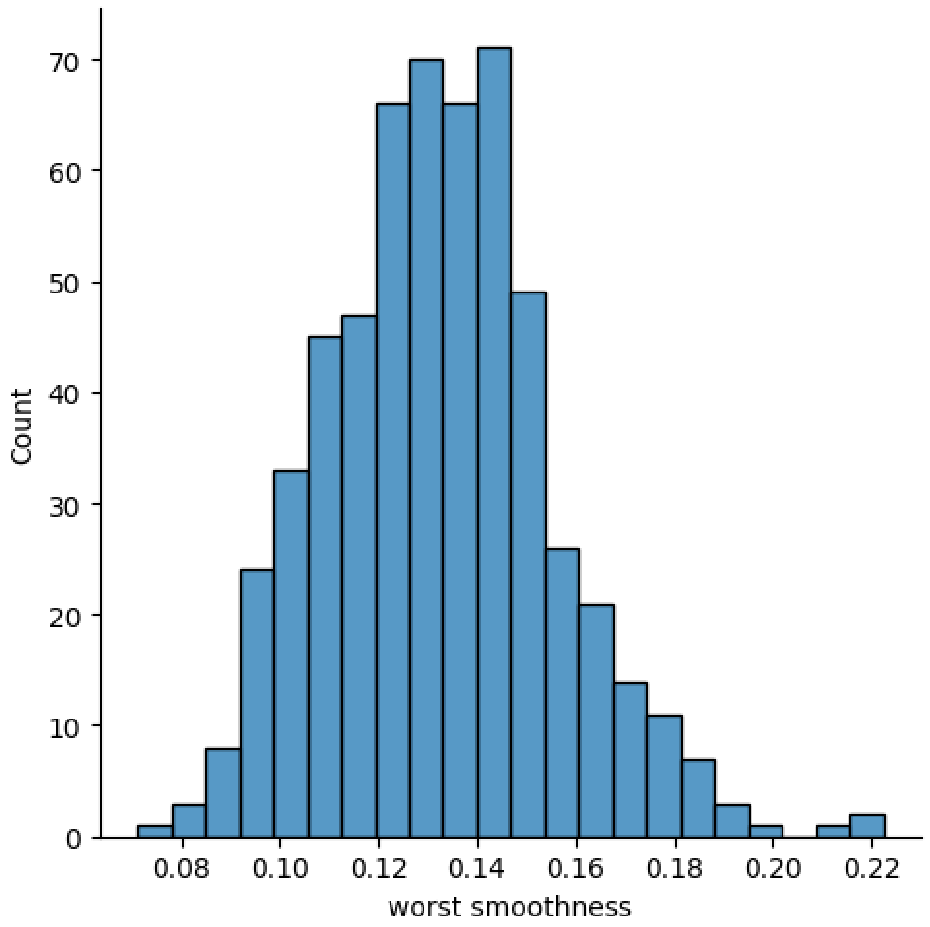

Count
0
20
40
60
80
0.000
0.005
0.010
0.015
0.020
0.025
0.030
fractal dimension error


Count
0
20
40
60
80
100
10
15
20
25
30
35
worst radius

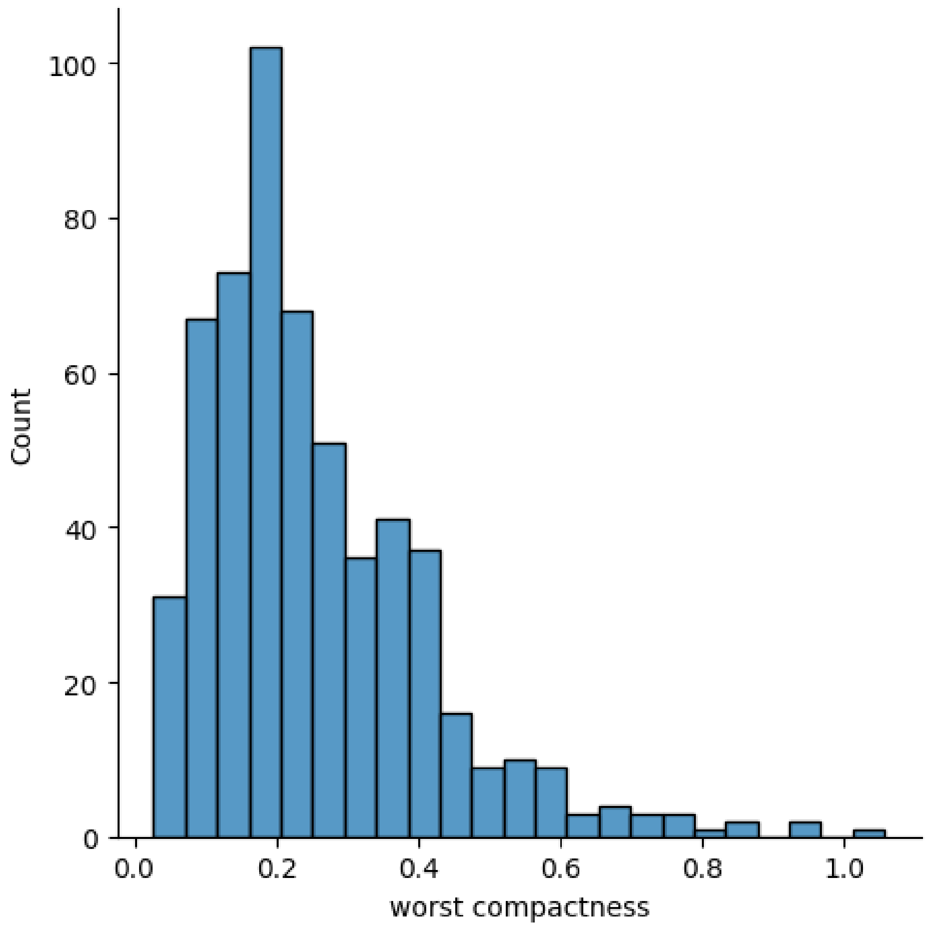

Count
0
20
40
60
80
15
20
25
30
35
40
45
50
worst texture

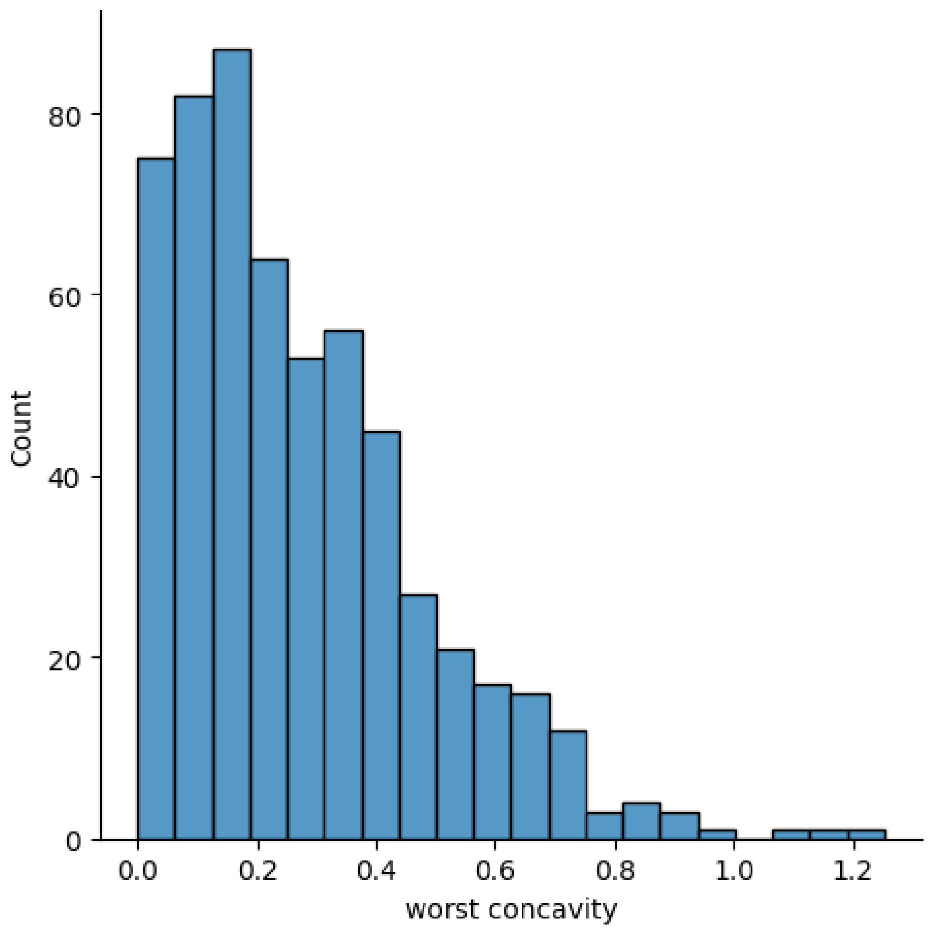

Count
0
20
40
60
80
100
50
75
100
125
150
175
200
225
250
worst perimeter

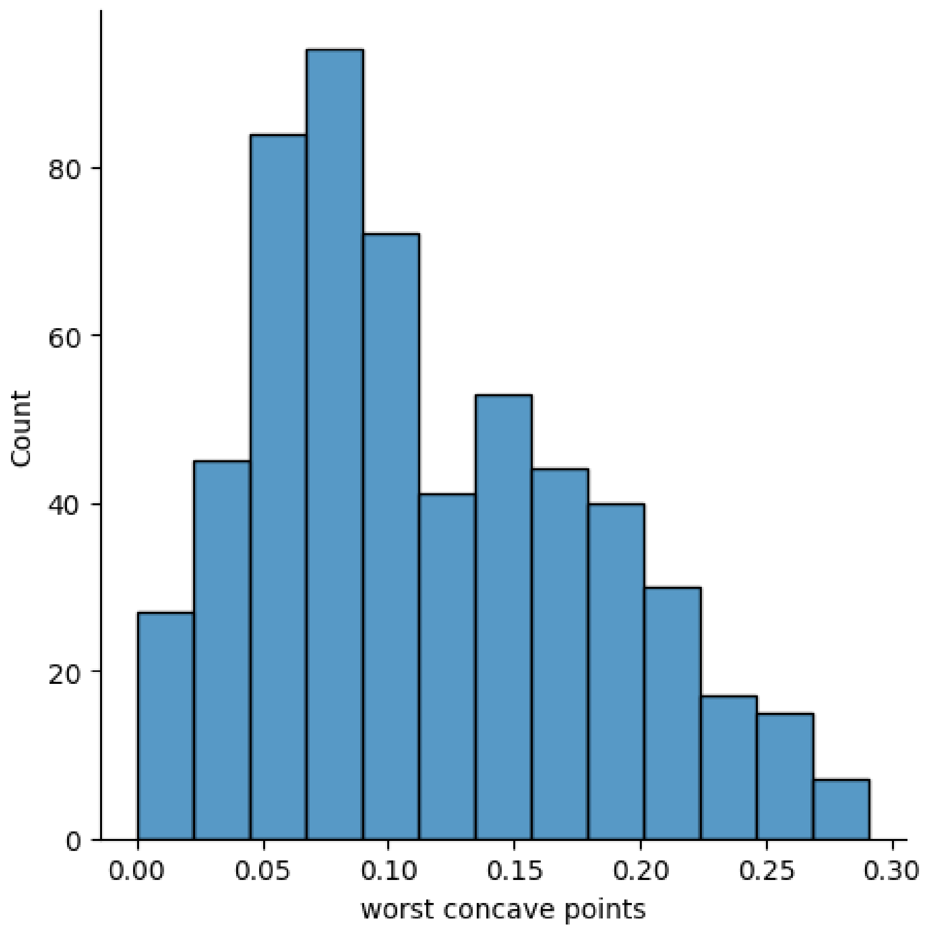

Count
worst area
0
20
40
60
80
100
120
0
500
1000
1500
2000
2500
3000
3500
4000

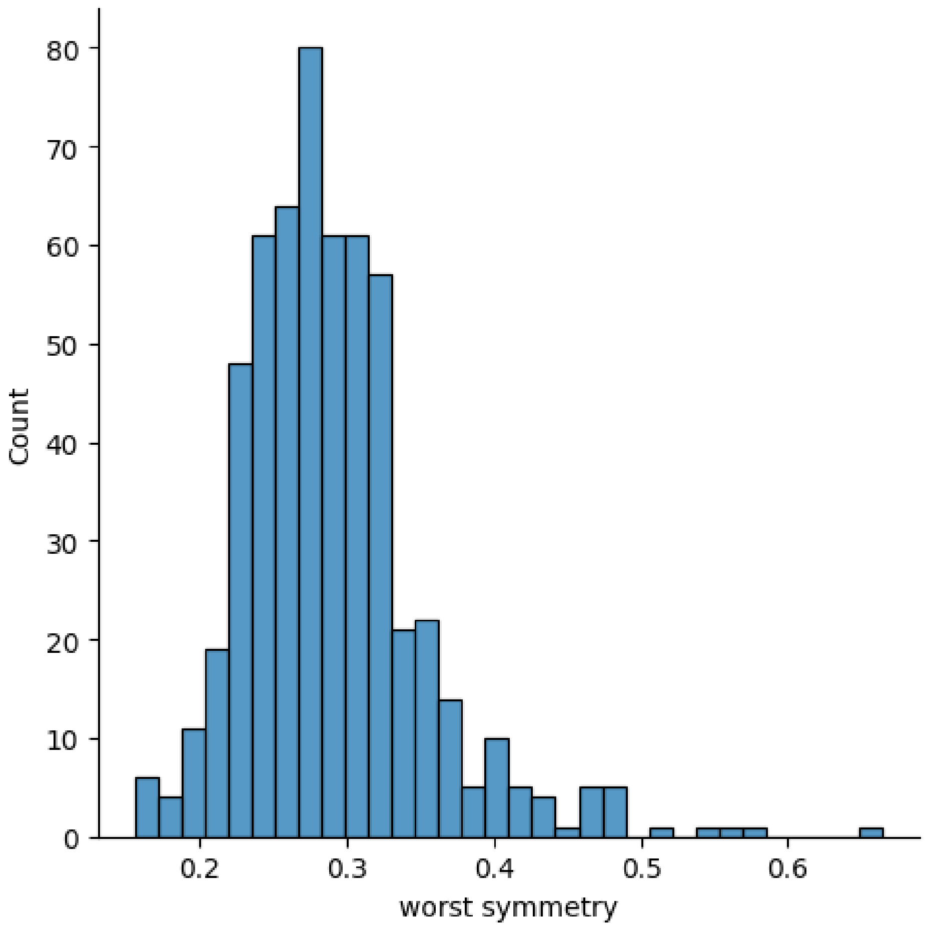


Count
0
10
20
30
40
50
60
70
0.08
0.10
0.12
0.14
0.16
0.18
0.20
0.22
worst smoothness

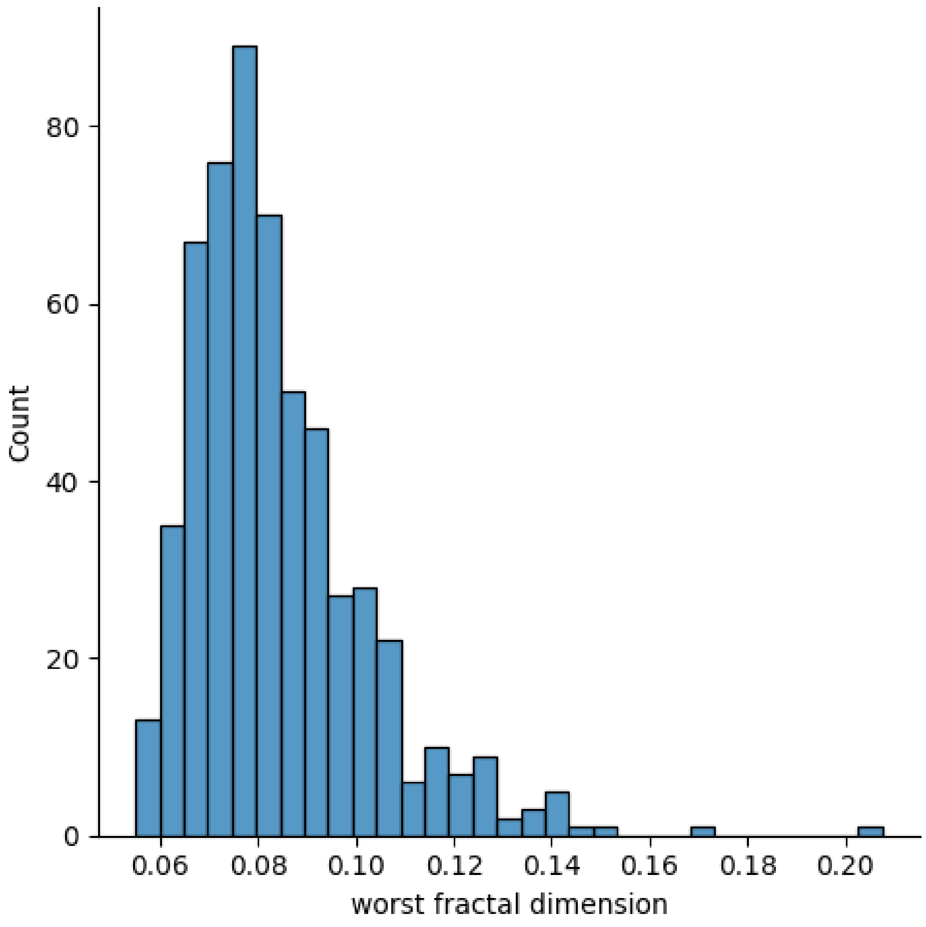

Count
0
20
40
60
80
100
0.0
0.2
0.4
0.6
0.8
1.0
worst compactness

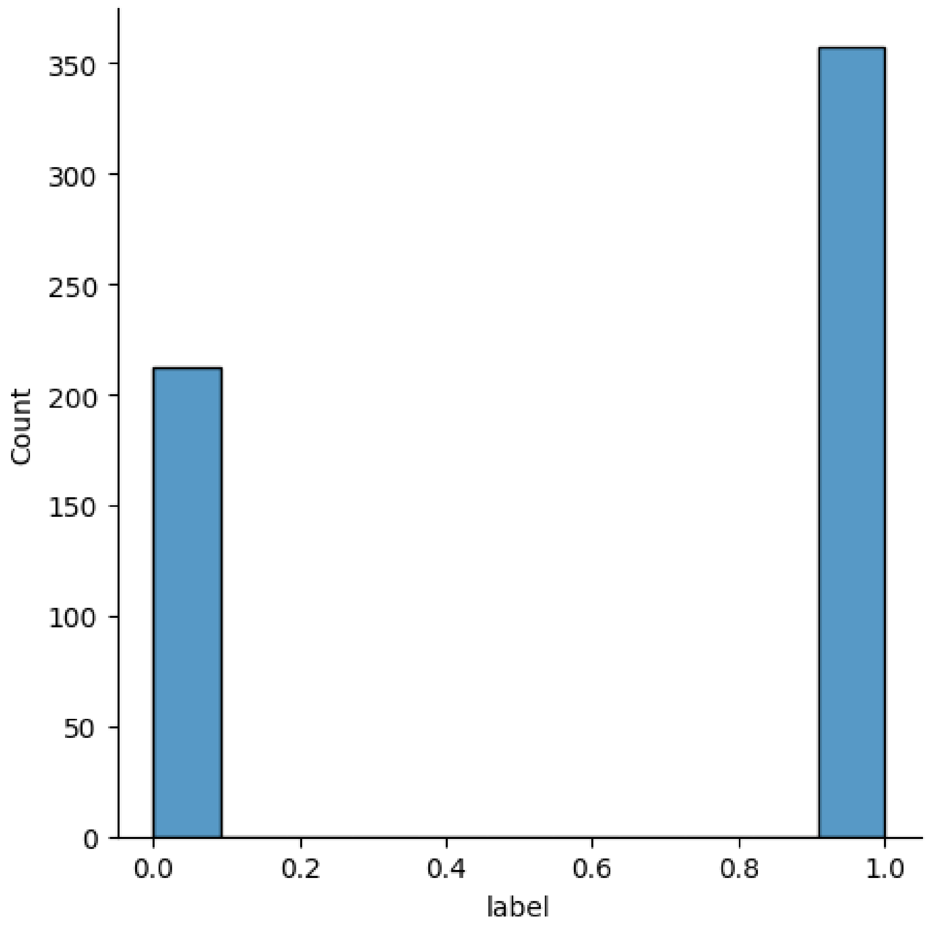

Count
0
20
40
60
80
0.0
0.2
0.4
0.6
0.8
1.0
1.2
worst concavity


Count
0
20
40
60
80
0.00
0.05
0.10
0.15
0.20
0.25
0.30
worst concave points



Count
80
70
60
50
40
30
20
10
0
0.2
0.3
0.4
0.5
0.6
worst symmetry


Count
0
20
40
60
80
0.06
0.08
0.10
0.12
0.14
0.16
0.18
0.20
worst fractal dimension

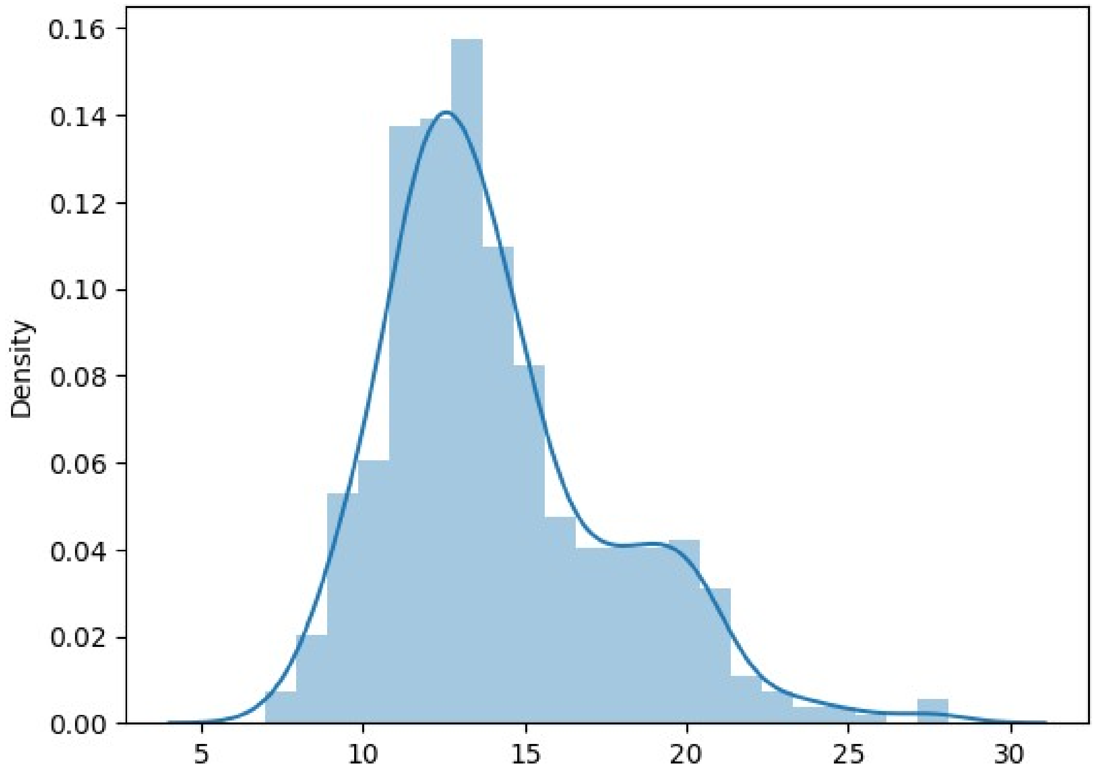


We can understand the **most of the features are positively skewed**. IE, the mean is greater the median, meaning we have more samples for one condition and less data in the other leading to a case where the model finds it difficult to make predictions for the regions where it has less data in.

```
sns.distplot(x = df['mean radius'])
```

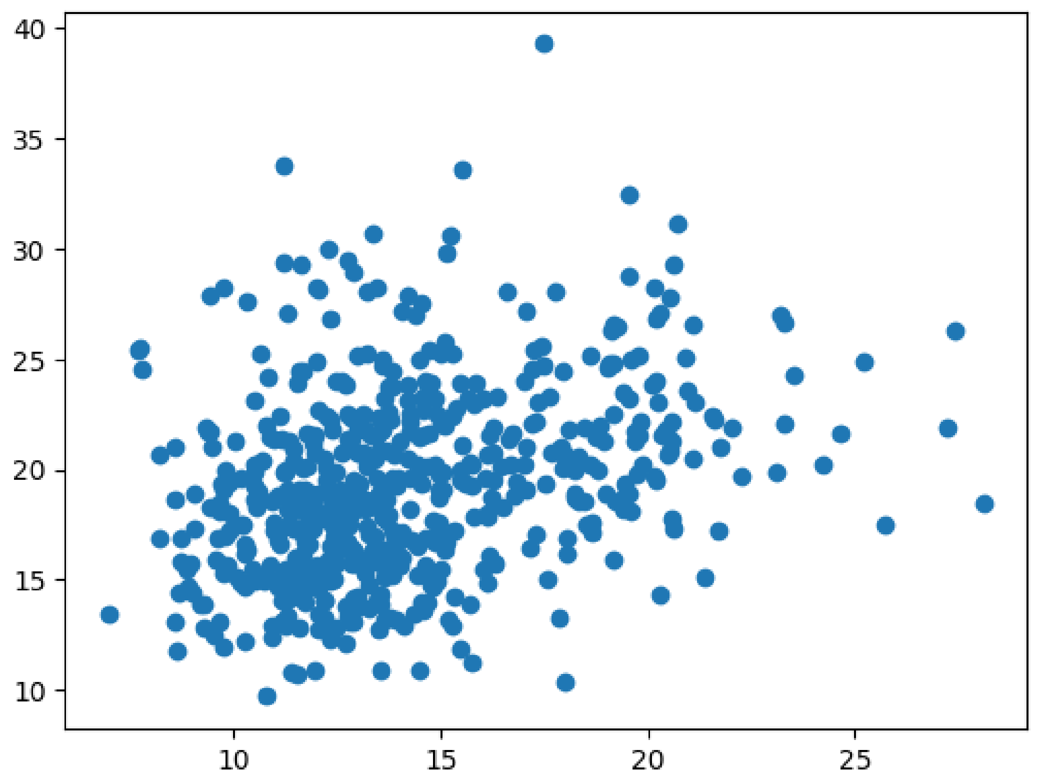
```
<ipython-input-21-df84fe73072b>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(x = df['mean radius'])
```

```
<Axes: ylabel='Density'>
```



```
# Finding correlation between the features - Bi-variate analysis
# sns.pairplot(df)

# Generate 30*30 graphs - Time consuming

# Making a pairplot for 2 featuers

first_feature = df.iloc[:, 0]
second_feature = df.iloc[:,1]

plt.scatter(x = first_feature, y = second_feature)
```

```
<matplotlib.collections.PathCollection at 0x78dac18127d0>
```



Outlier Detection

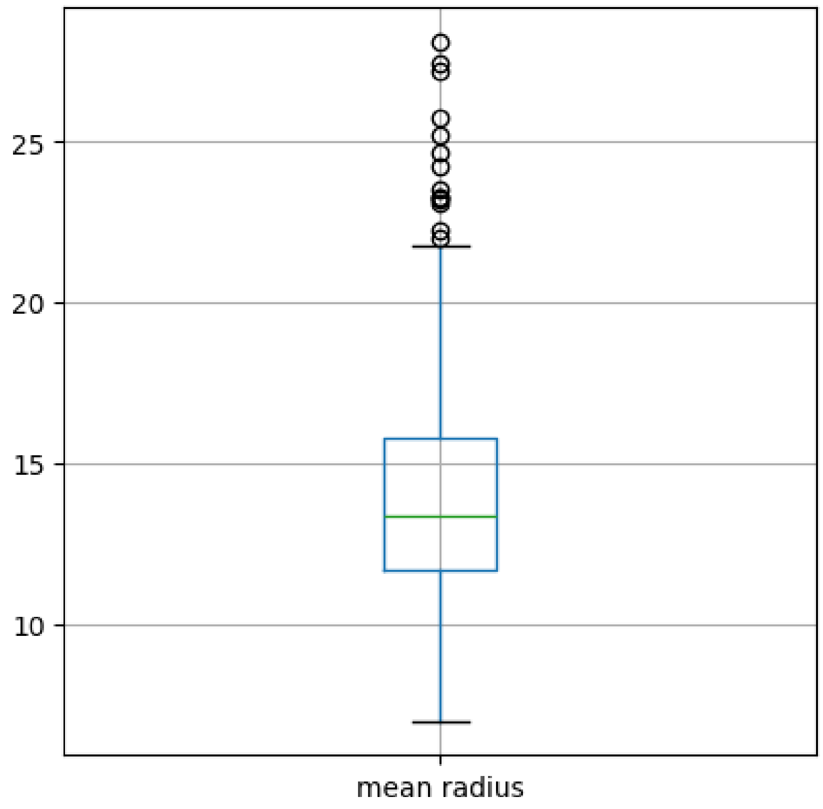
An observation that lies an abnormal distance from other values in a random sample from a population.

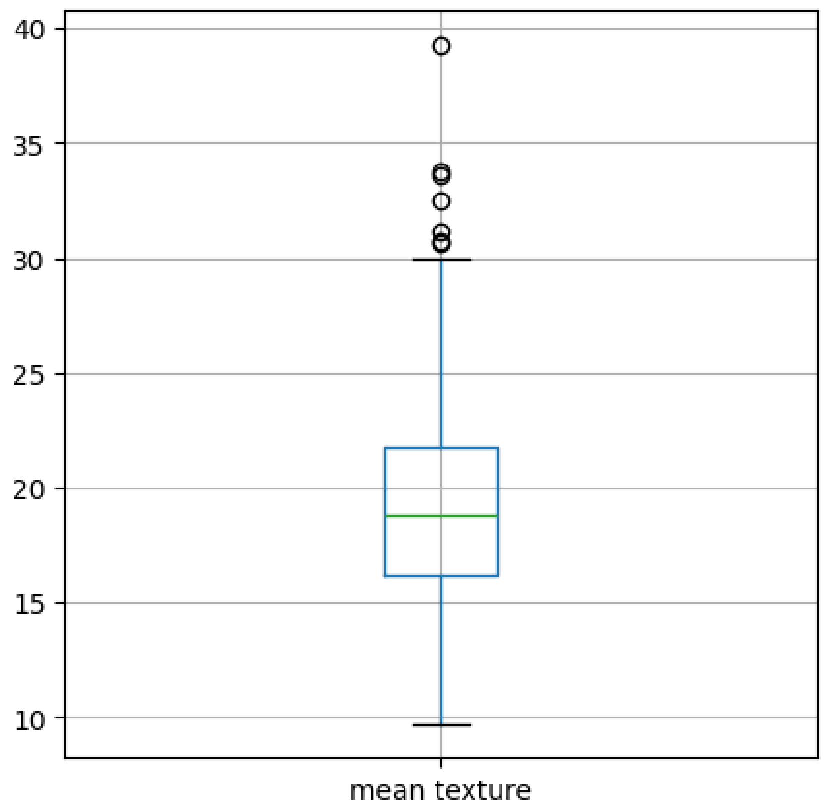
Regression models are sensitive to outliers and are needed to be processed.

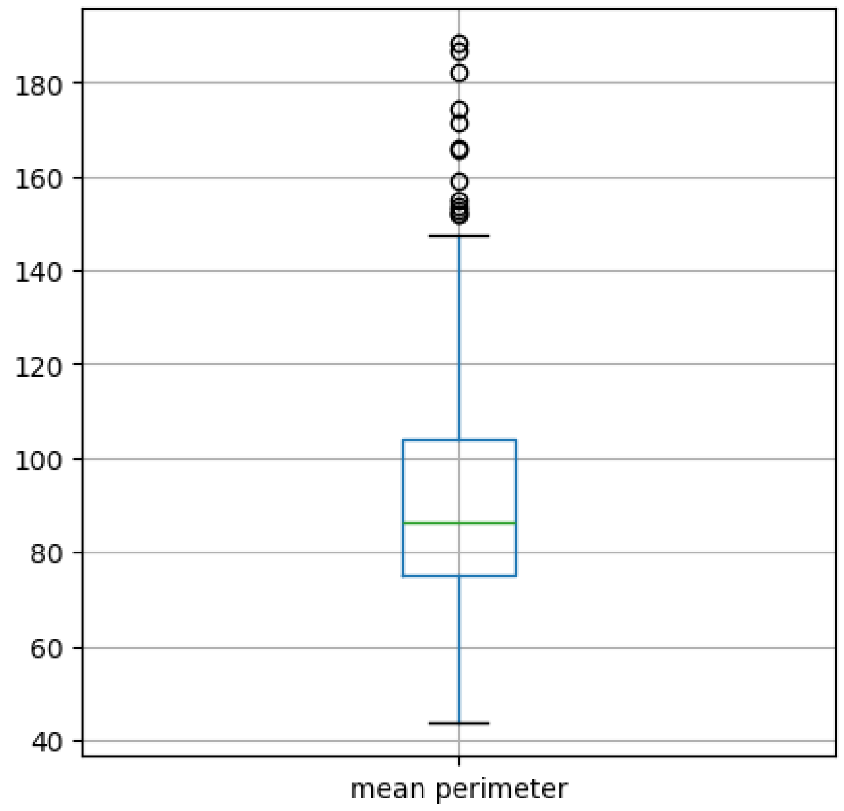
```
# Box and Whisker plot
for column in df:
    plt.figure(figsize = (5,5))
    df.boxplot([column])

# The black circle represents the outliers, with the blue horizontal
lines representing the 25th(Q1) and 75th(Q3) percentile. The green
line(Q2) represents the median.
# The black horizontal line are the minimum and maximum values
# Q3 - Q1 is the interquantile range
# Values that lie above 1.5 times the inter-quantile range are the
outliers.
```

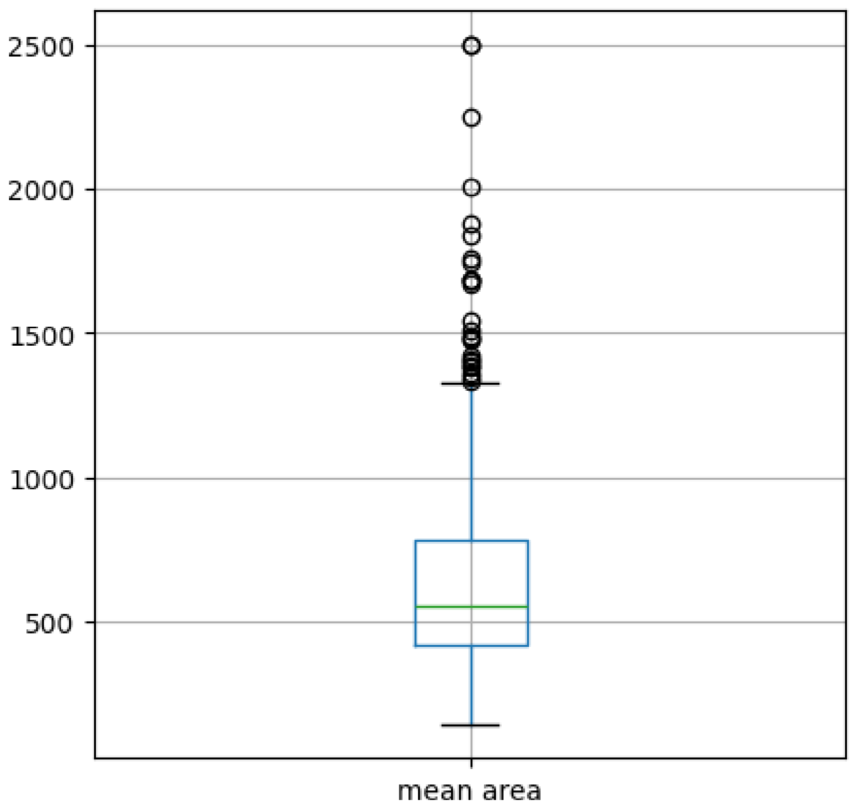
```
<ipython-input-24-be0ffa52dc02>:3: RuntimeWarning: More than 20
figures have been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and
may consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`). Consider using
```

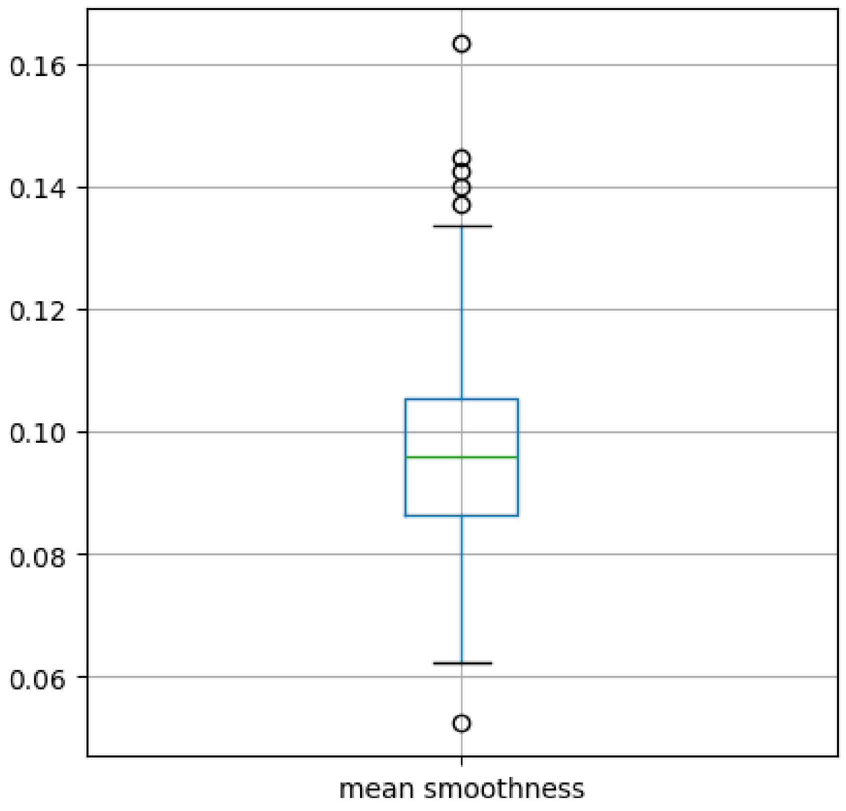
```
`matplotlib.pyplot.close()`.
  plt.figure(figsize = (5,5))
```

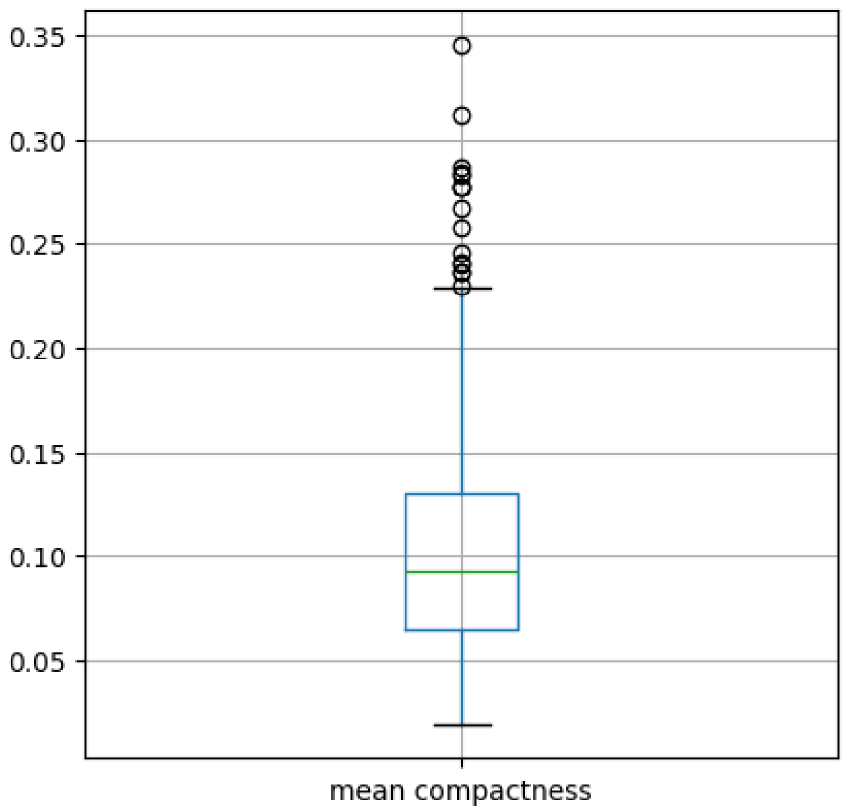


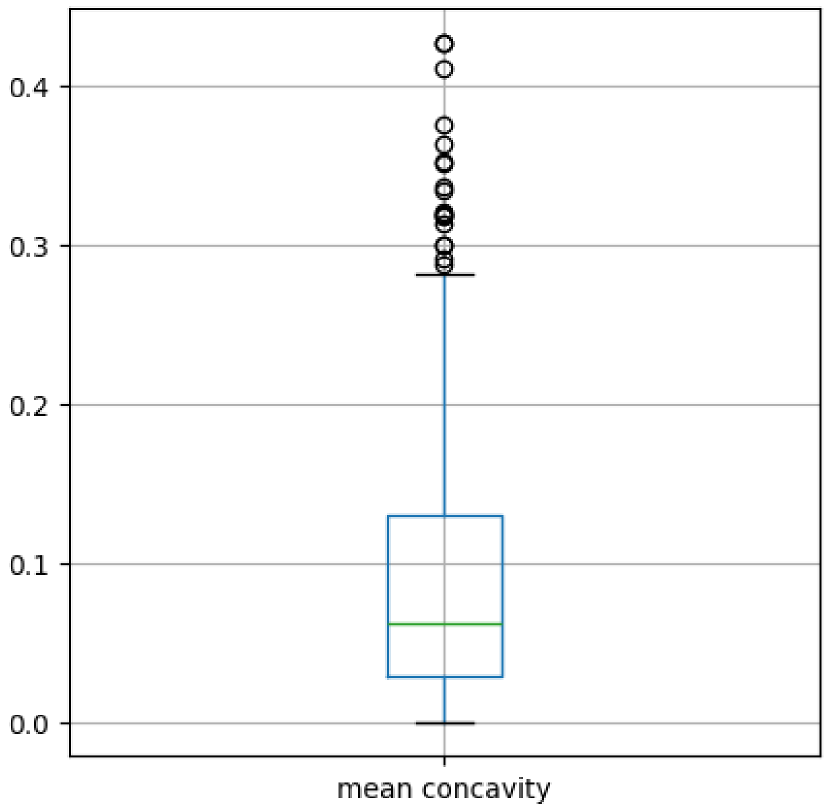

40
35
30
25
20
15
10
mean texture

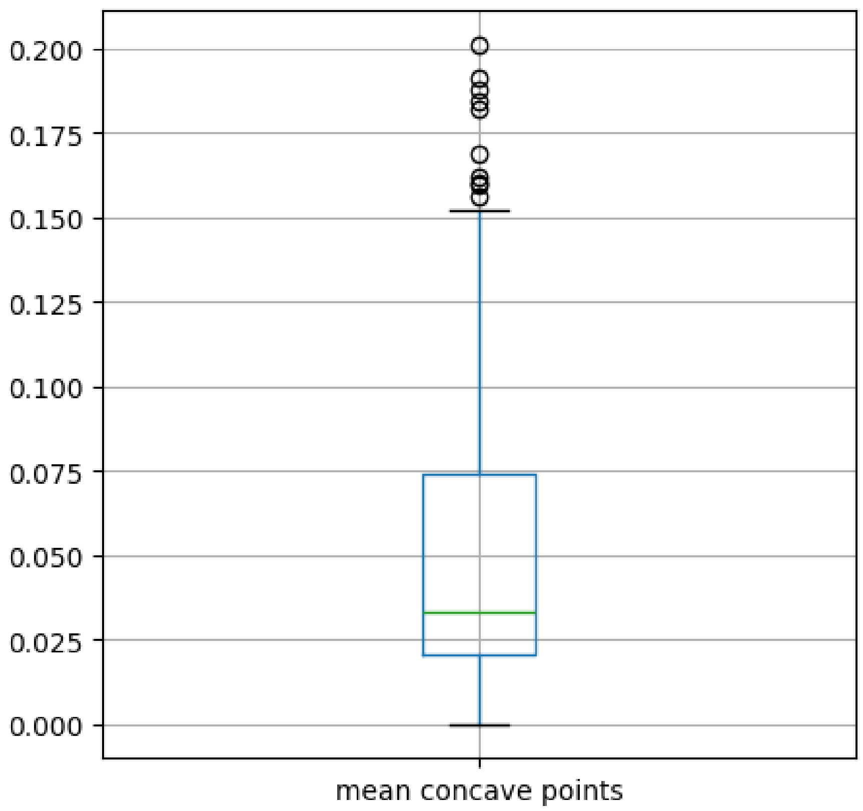

180
160
140
120
100
80
60
40
mean perimeter

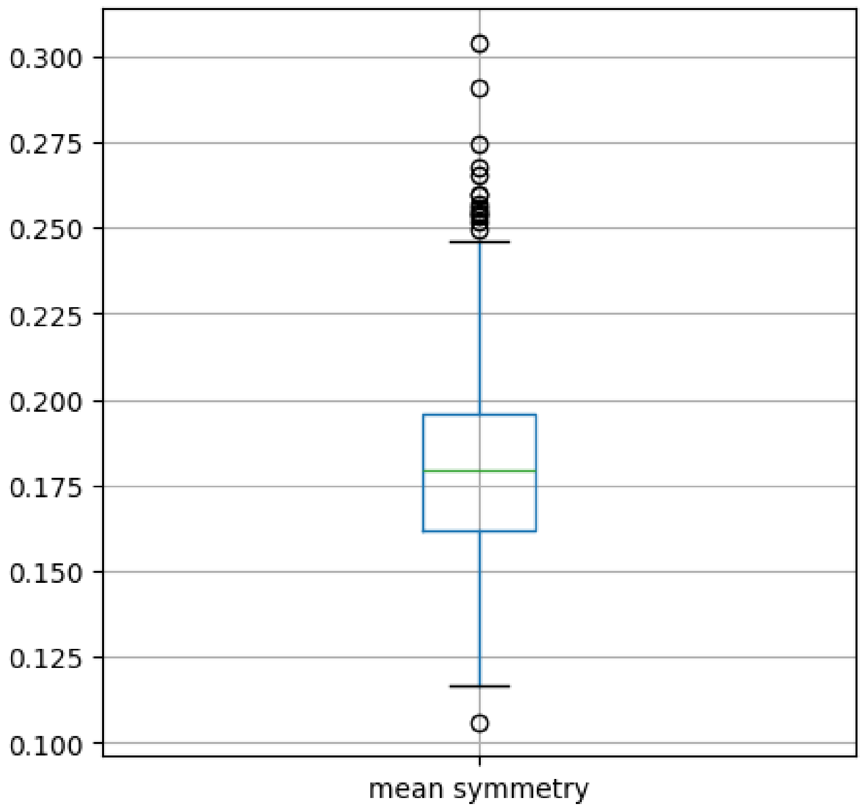

2500
2000
1500
1000
500
mean area

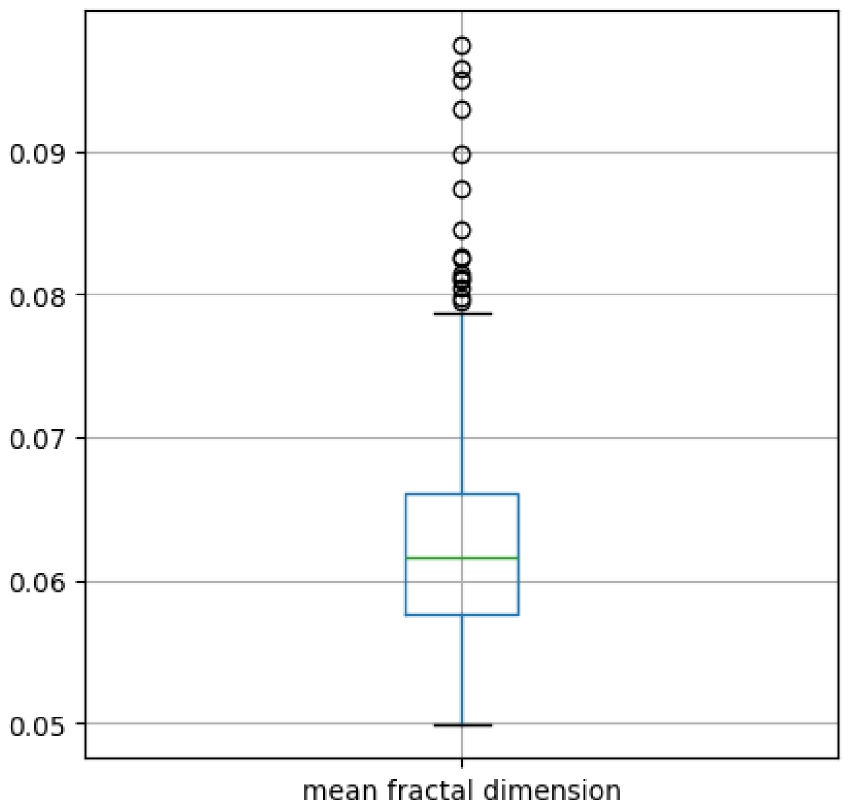

0.16
0.14
0.12
0.10
0.08
0.06
mean smoothness

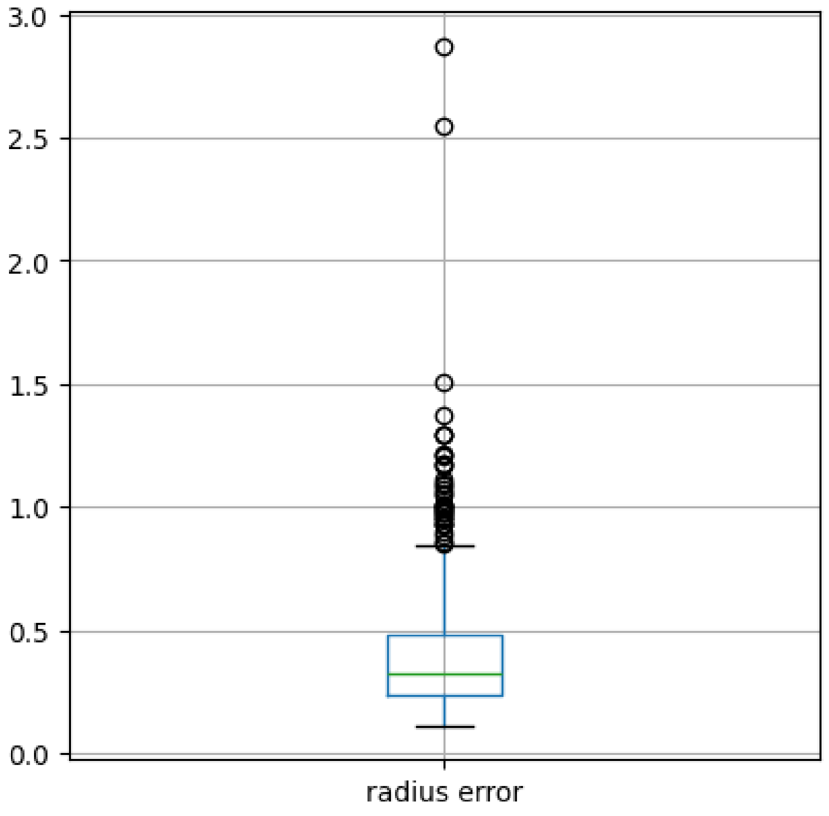

0.35
0.30
0.25
0.20
0.15
0.10
0.05
mean compactness

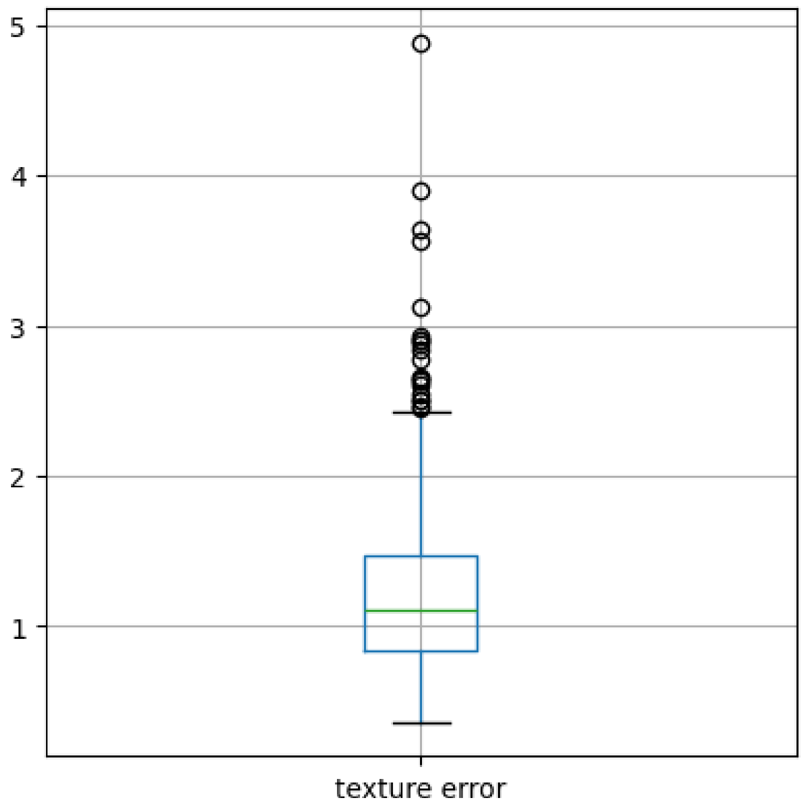

0.4
0.3
0.2
0.1
0.0
mean concavity

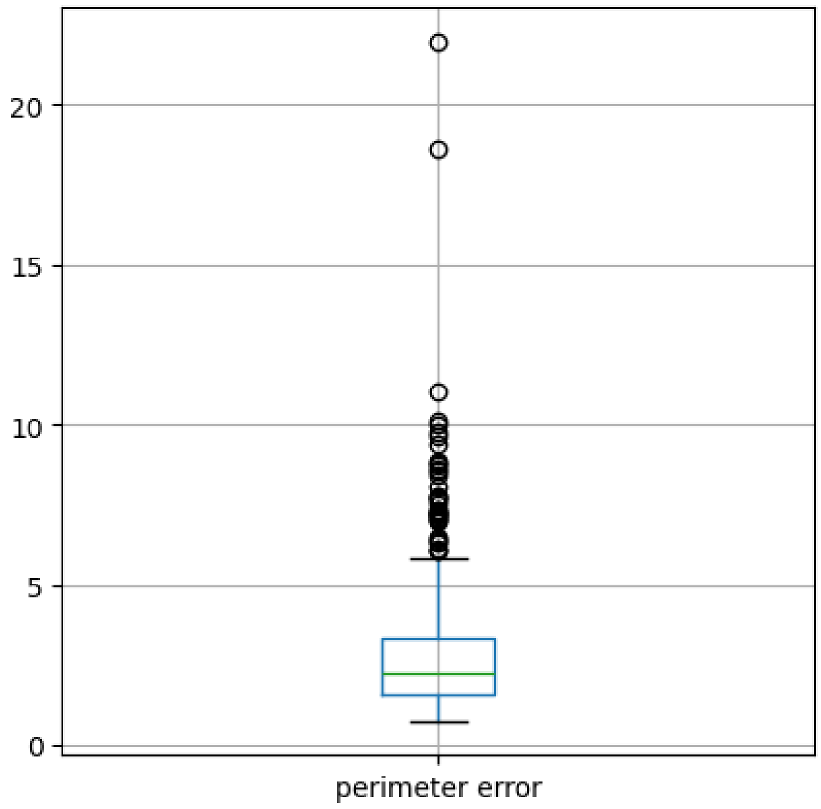

0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
mean concave points

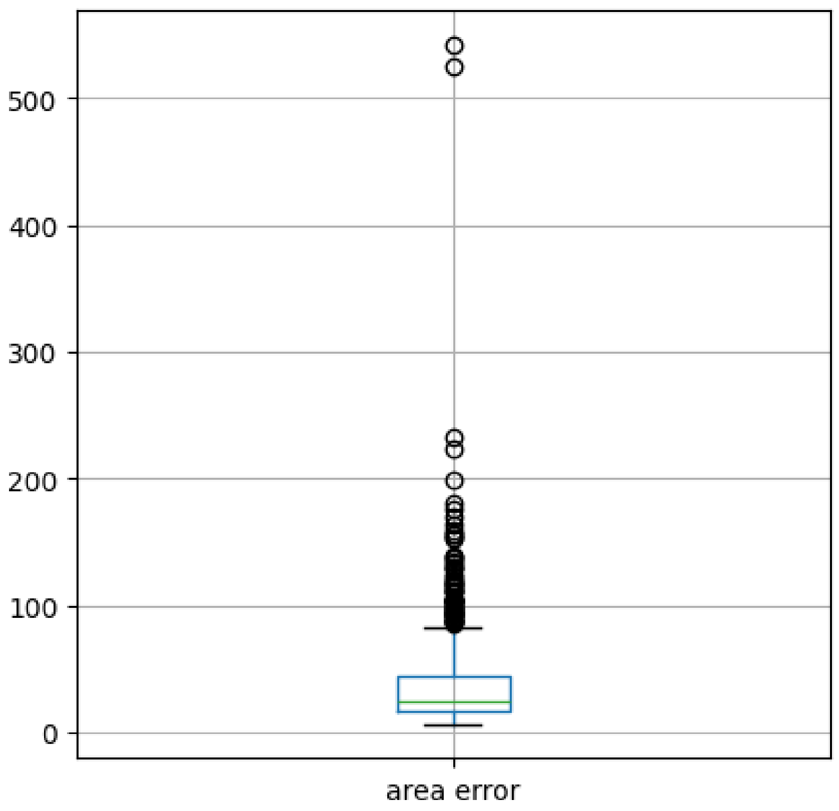

0.300
0.275
0.250
0.225
0.200
0.175
0.150
0.125
0.100
mean symmetry

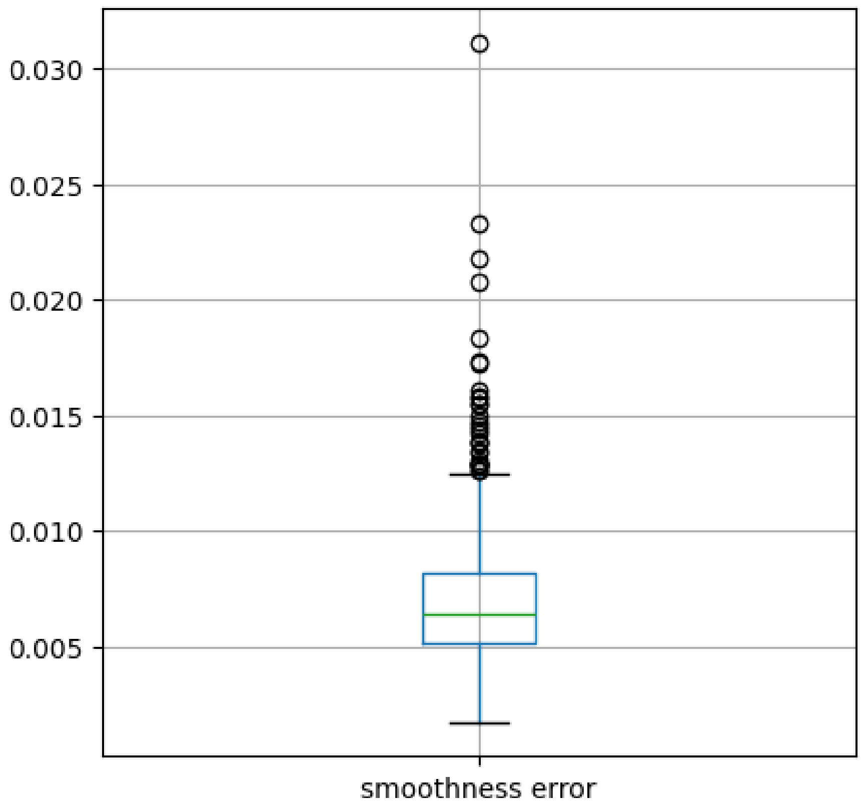

0.09
0.08
0.07
0.06
0.05
mean fractal dimension

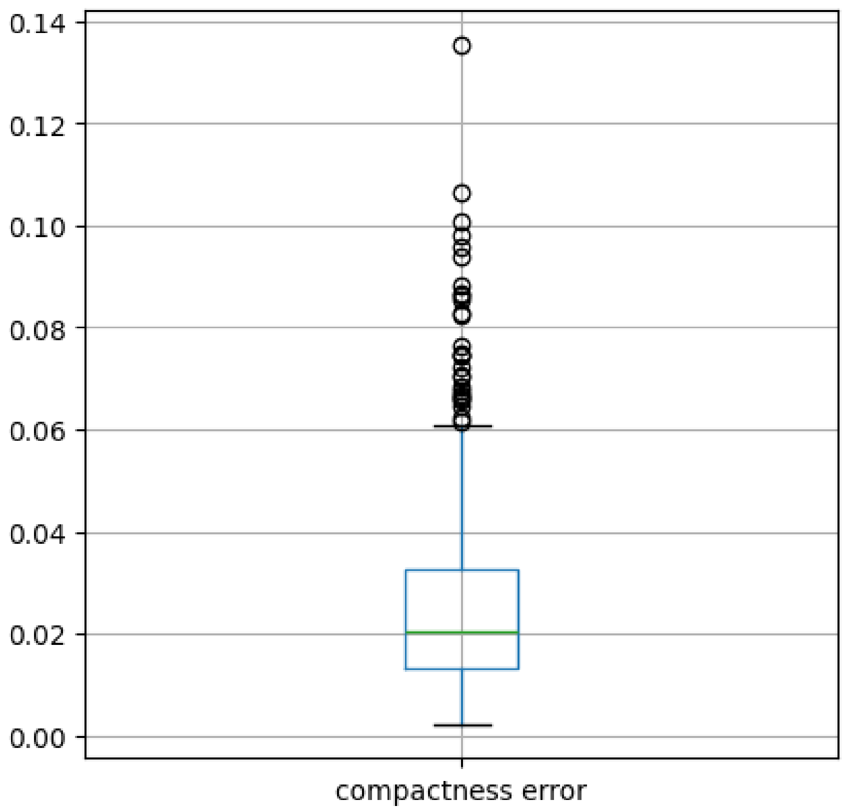

3.0
2.5
2.0
1.5
1.0
0.5
0.0
radius error

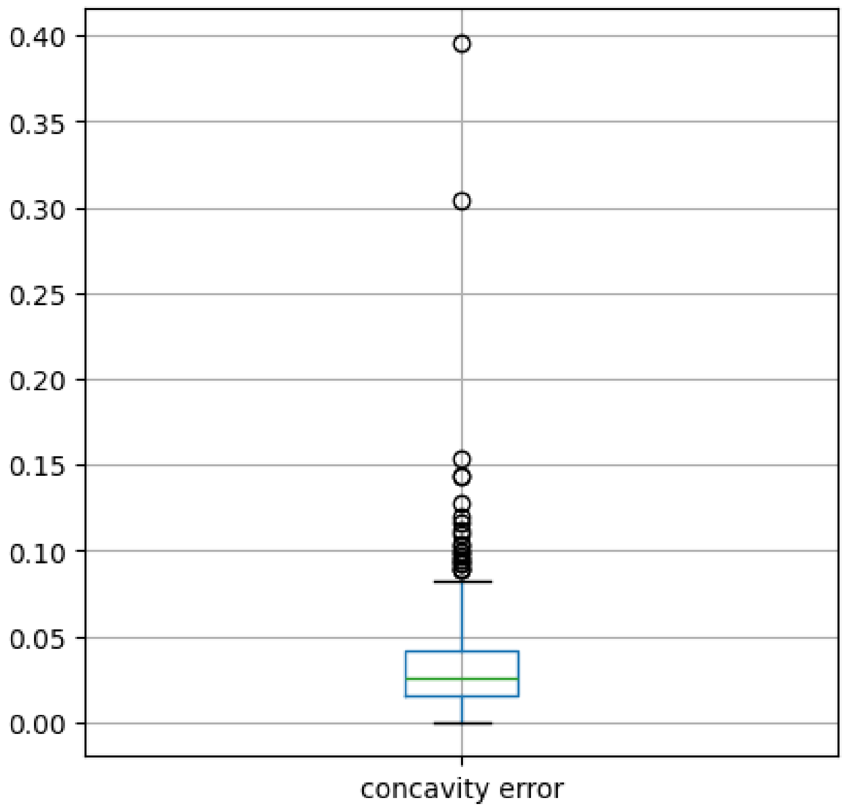

5
4
3
2
1
texture error

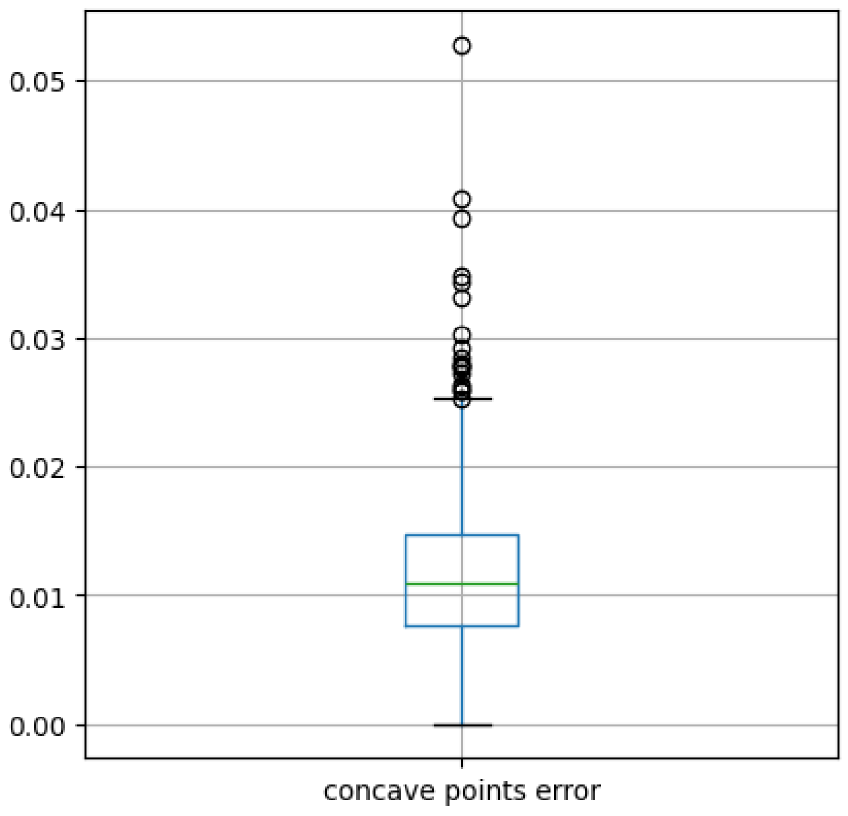

20
15
10
5
0
perimeter error

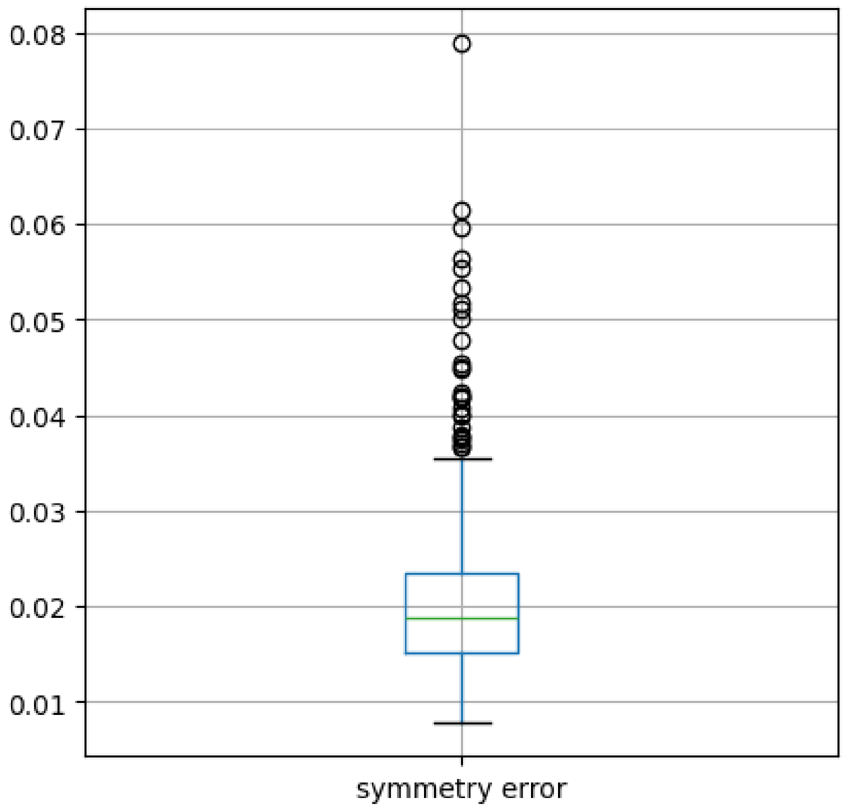

500
400
300
200
100
0
area error

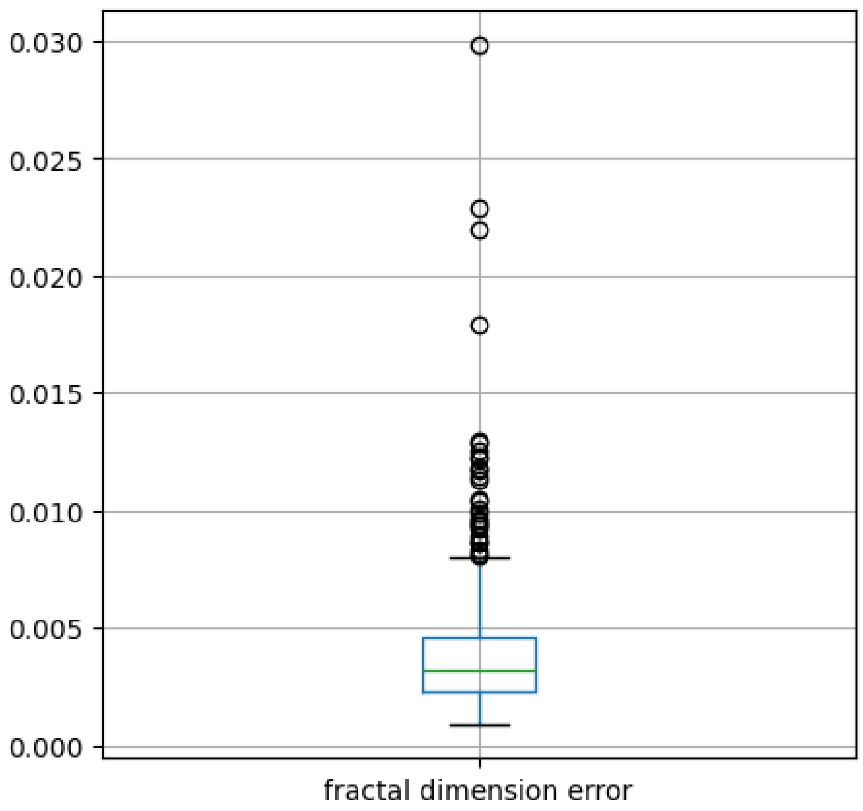

0.030
0.025
0.020
0.015
0.010
0.005
smoothness error

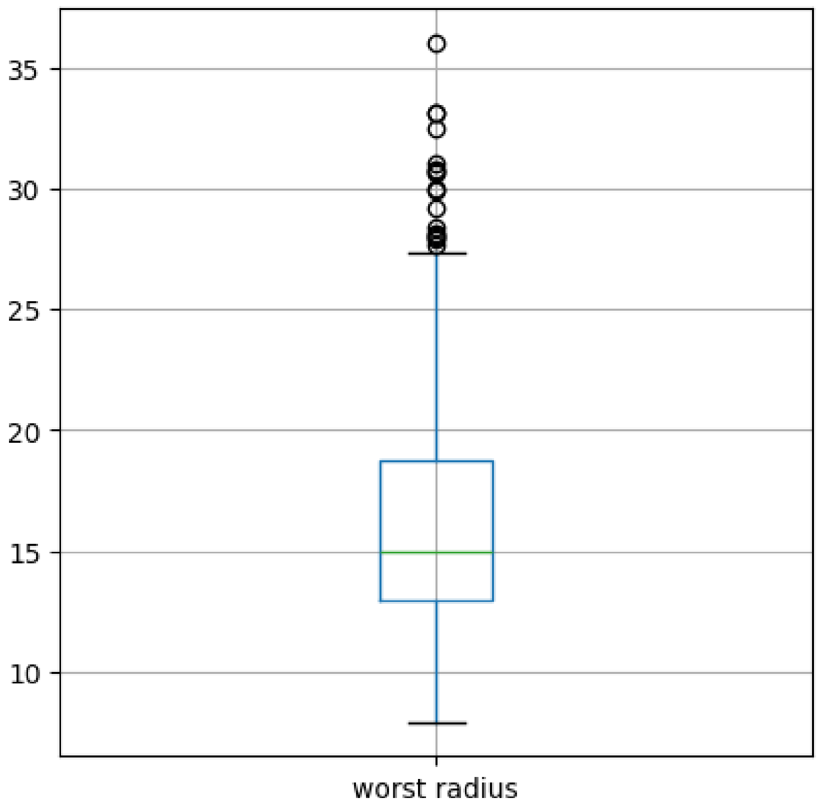

0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
compactness error

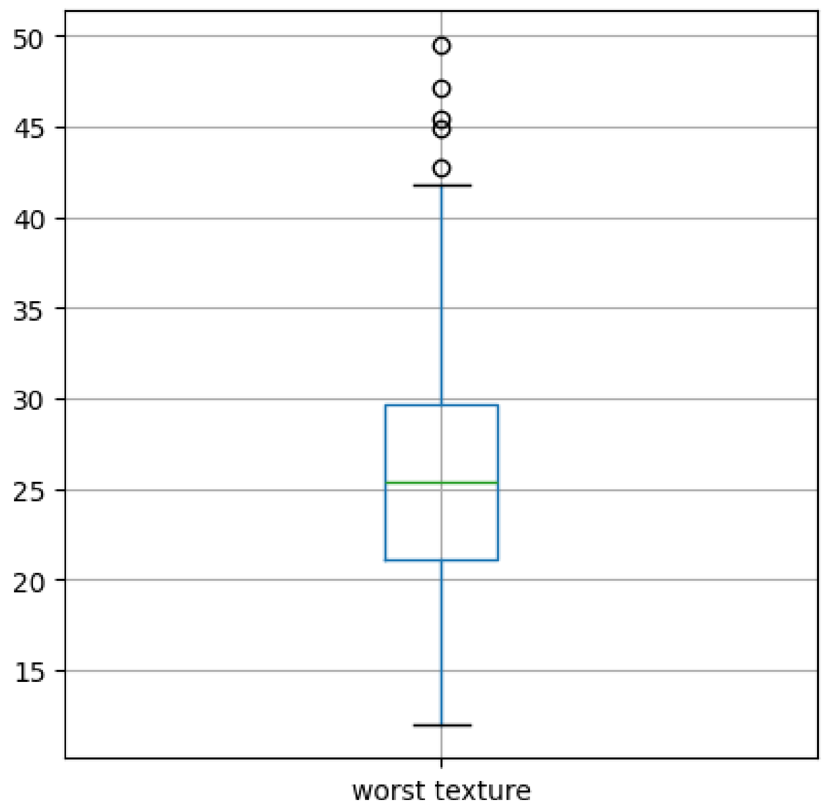

0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
concavity error

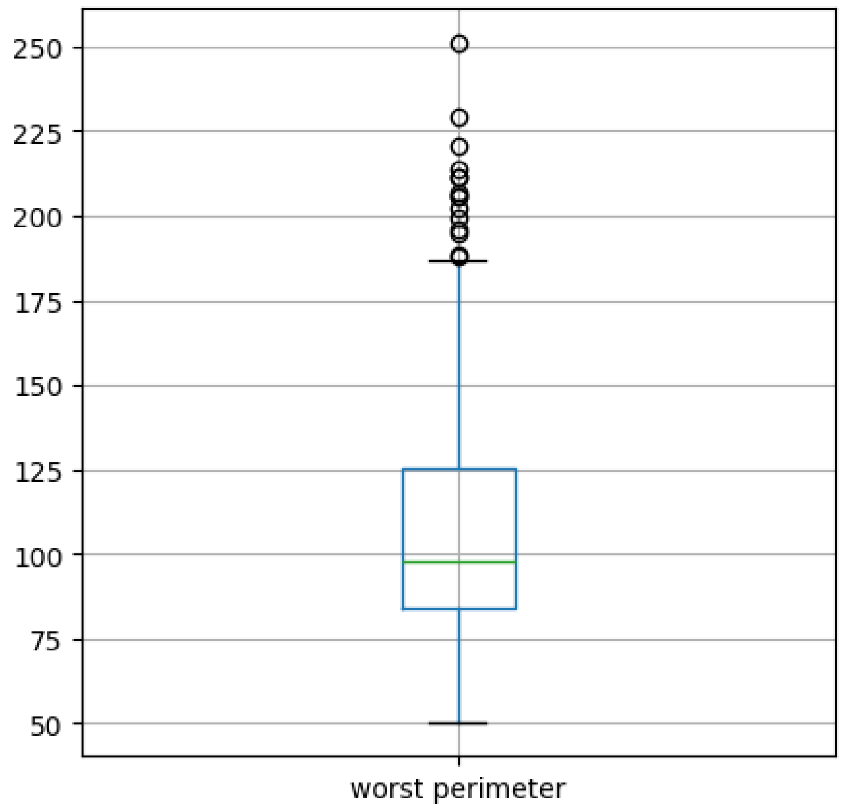

0.05
0.04
0.03
0.02
0.01
0.00
concave points error

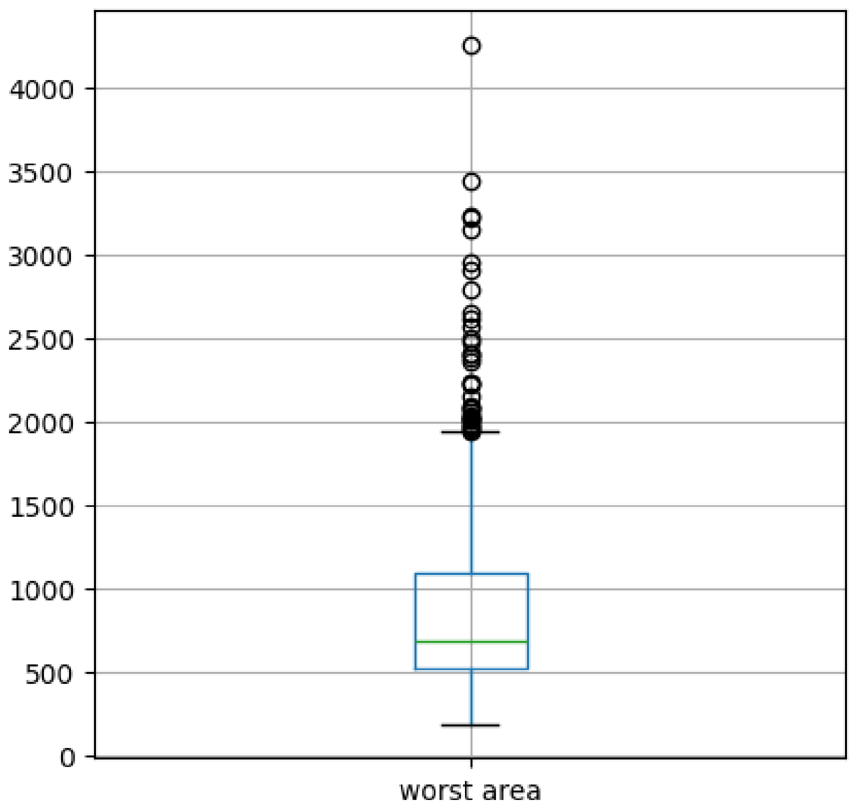

0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
symmetry error

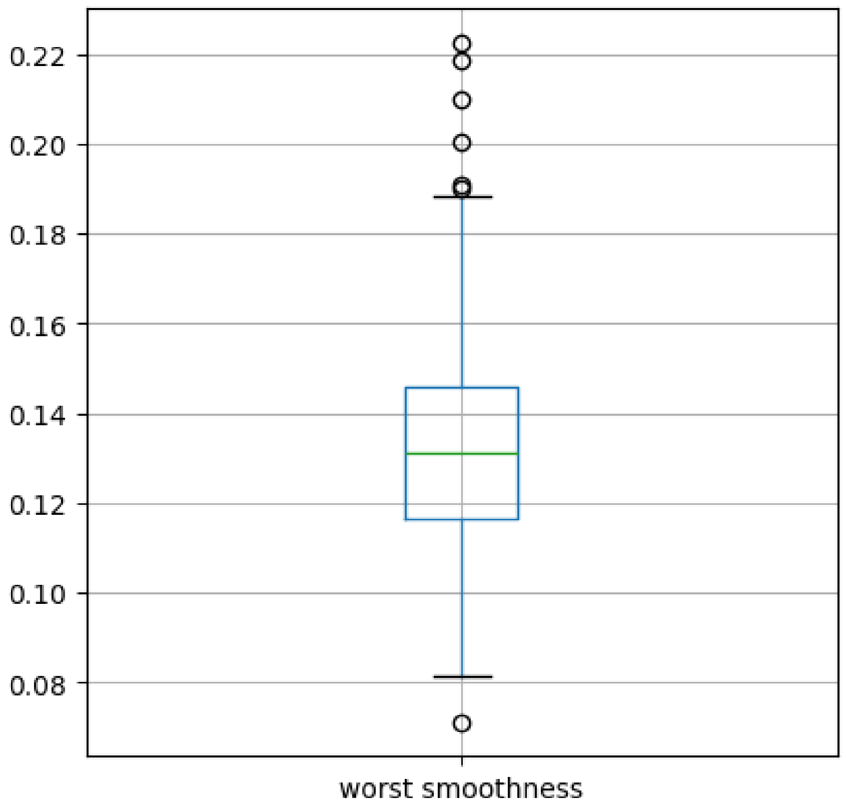

0.030
0.025
0.020
0.015
0.010
0.005
0.000
fractal dimension error

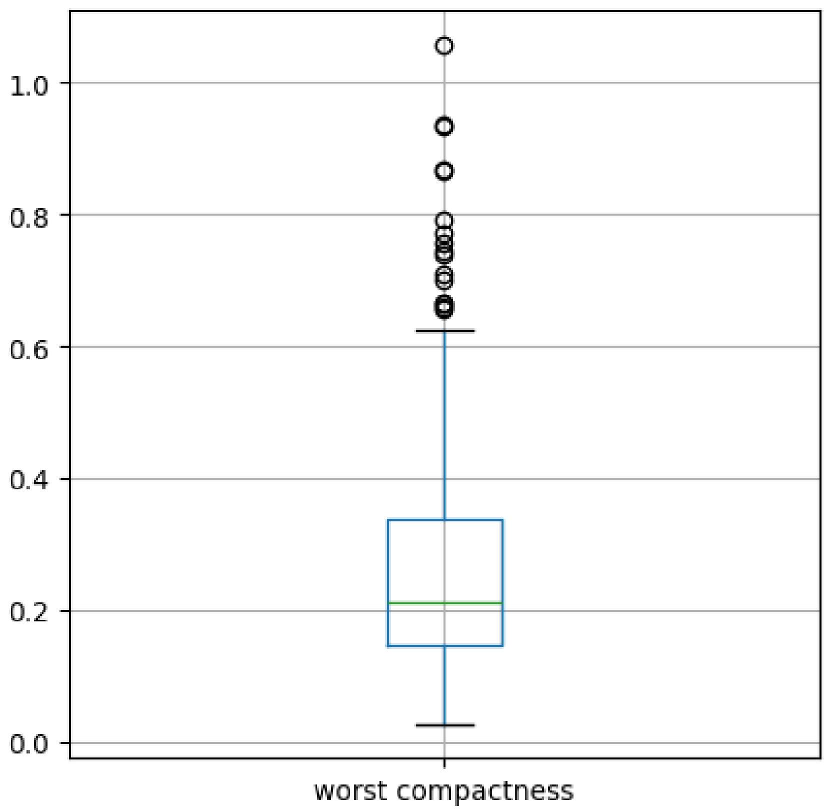

35
30
25
20
15
10
worst radius

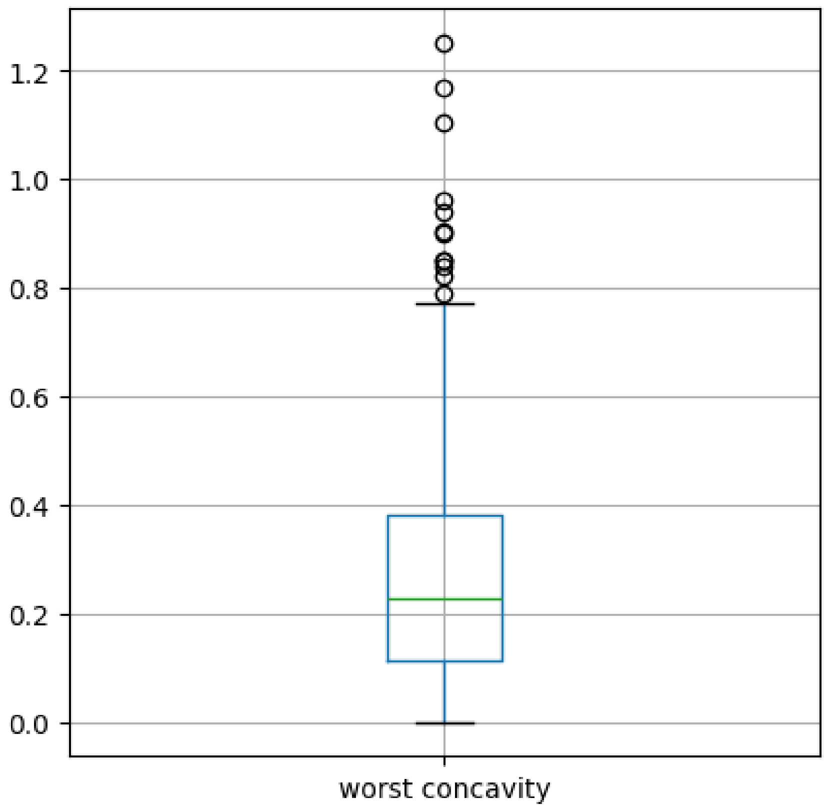

50
45
40
35
30
25
20
15
worst texture

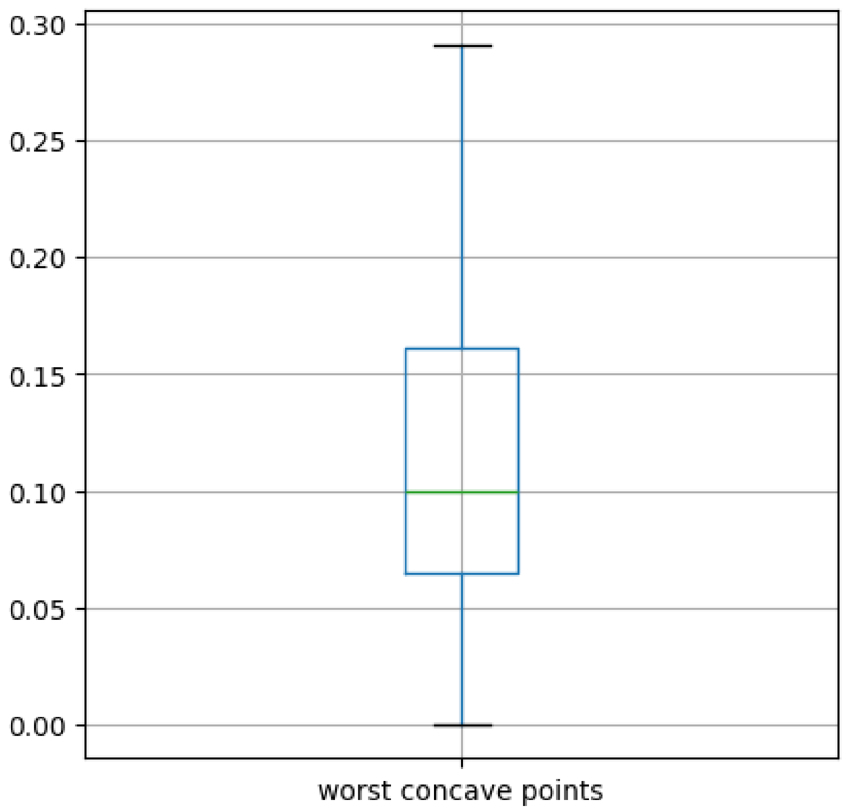

250
225
200
175
150
125
100
75
50
worst perimeter


4000
3500
3000
2500
2000
1500
1000
500
0
worst area

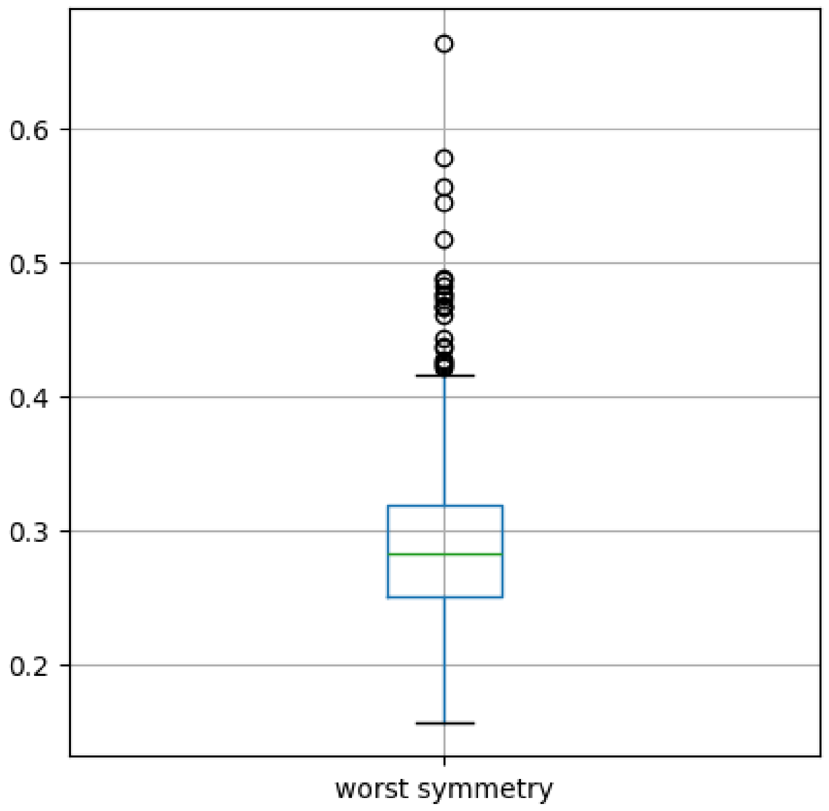

0.22
0.20
0.18
0.16
0.14
0.12
0.10
0.08
worst smoothness


1.0
0.8
0.6
0.4
0.2
0.0
worst compactness

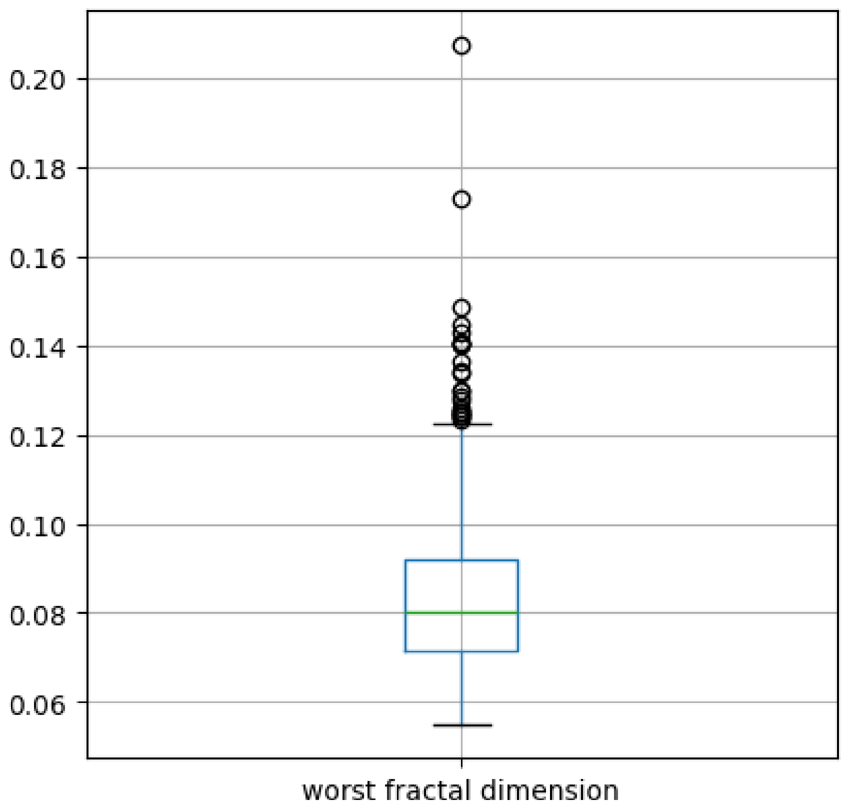

1.2
1.0
0.8
0.6
0.4
0.2
0.0
worst concavity

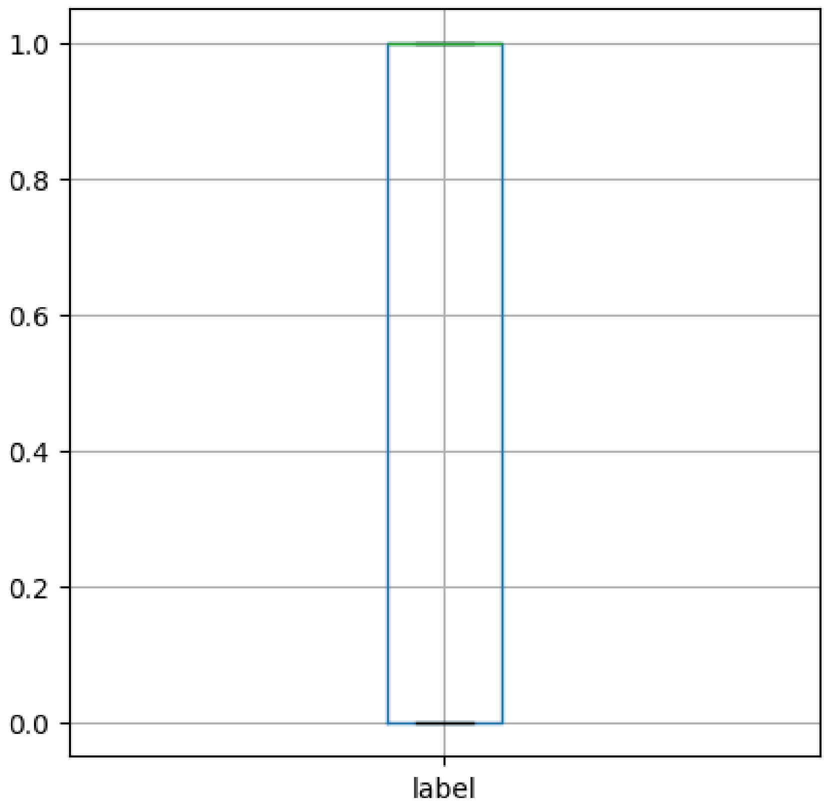


0.30
0.25
0.20
0.15
0.10
0.05
0.00
worst concave points

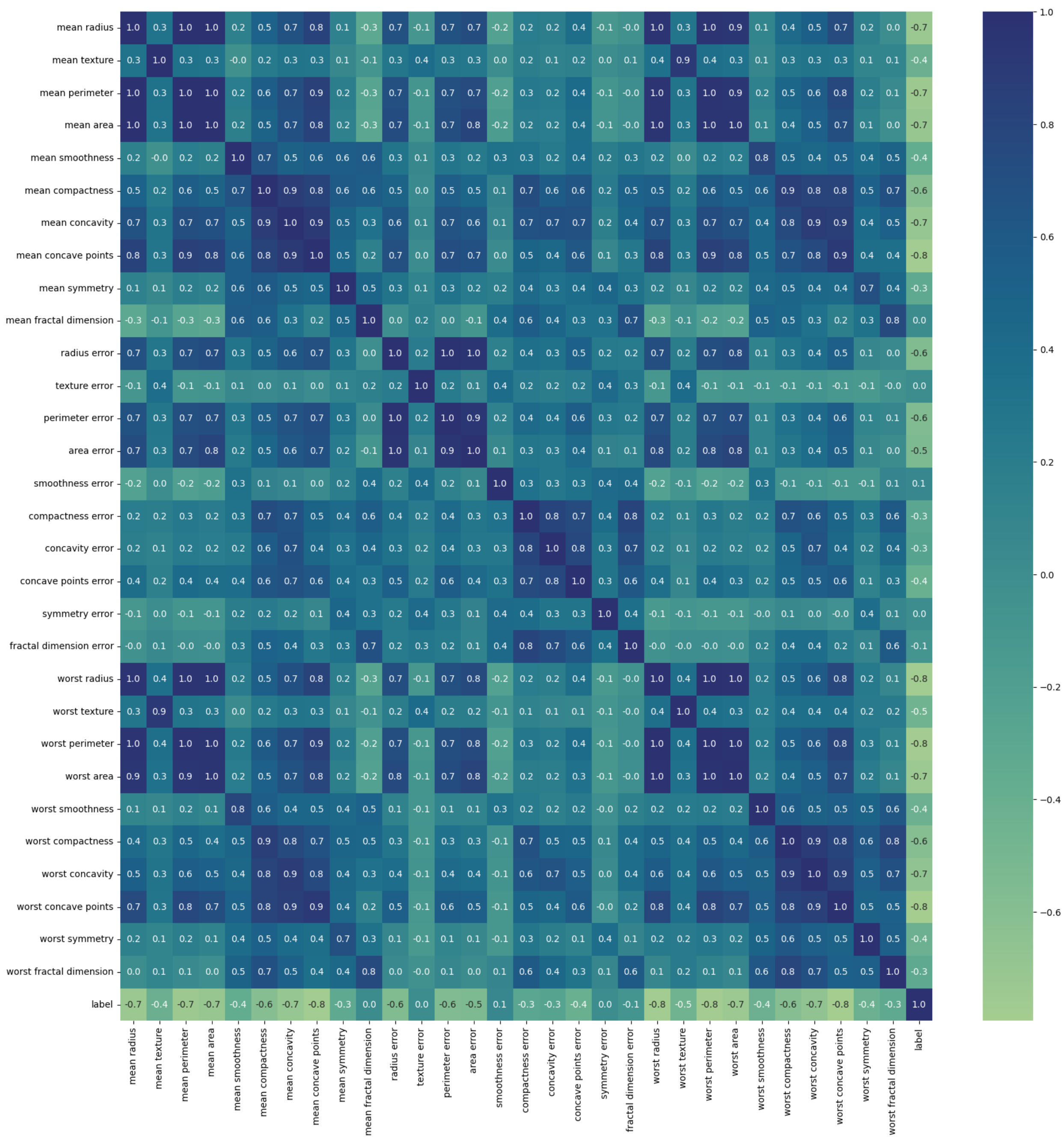


0.6
0.5
0.4
0.3
0.2
worst symmetry


0.20
0.18
0.16
0.14
0.12
0.10
0.08
0.06
worst fractal dimension


1.0
0.8
0.6
0.4
0.2
0.0
label

```python
# When using dataset such as this to model a regression model, the
outliers need to be replaced.
# We replace them using the median(more)/mean(less) value.

correlation_matrix = df.corr()
plt.figure(figsize = (20,20))
sns.heatmap(correlation_matrix, cbar = True, fmt = '.1f', annot =
True, cmap = 'crest')

# Saving the image
plt.savefig('correlation_matrix.png')
```



Observation: The correlation matrix shows the relationships between different features (variables) in a dataset. The diagonal values of the matrics are always 1, as each feature is perfectly correlated with itself.

Correlation Coefficients: Values range between -1 and +1:

+1: Perfect positive correlation (as one feature increases, the other increases proportionally). -1: Perfect negative correlation (as one feature increases, the other decreases proportionally). 0: No correlation (features are independent).

Feature Relationships: Identifing the highly correlated features (positive or negative). Features with high correlation (close to +1 or -1) may be redundant. Detect multicollinearity, where independent variables are highly correlated with one or more variables, which can affect the performance of models like regression.

Feature Selection: Remove redundant features with high correlation to simplify models and improve efficiency

```
# Separating the features and target

X = df.drop(columns = 'label', axis = 1)
y = df['label']

# Splitting data into training and testing data
# 80% - Training data
# 20% - Testing data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state = 42)

print(X.shape, X_train.shape, X_test.shape)
```

```
(569, 30) (455, 30) (114, 30)
```

```
print(y_train.shape, y_test.shape)
```

```
(455,) (114,)
```

Training a Logistic Regression Model

```
model = LogisticRegression()

# Training the model using Training
model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/
_logistic.py:465: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

LogisticRegression()
```

```
# Model evaluation - Accuracy on training data

train_pred = model.predict(X_train)
train_accuracy = accuracy_score(y_train, train_pred)
print("The Training accuracy: ", train_accuracy)

The Training accuracy:  0.9472527472527472

y_pred = model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)
print("The test accuracy: ", test_accuracy)

The test accuracy:  0.956140350877193
```

Predictive System

```
input_data = (19.81, 22.15, 130, 1260, 0.09831, 0.1027, 0.1479,
0.09498, 0.1582, 0.05395, 0.7582, 1.017, 5.865, 112.4, 0.006494,
0.01893, 0.03391, 0.01521, 0.01356, 0.001997, 27.32, 30.88, 186.8,
2398, 0.1512, 0.315, 0.5372, 0.2388, 0.2768, 0.07615)
input_array = np.asarray(input_data)

# Reshaping the data
input_array_reshaped = input_array.reshape(1, -1)

prediction = model.predict(input_array_reshaped)
print(prediction)

if prediction[0] == 0:
    print("The patient is Benign")
else:
    print("The patient is Malignant")

[0]
The patient is Benign

/usr/local/lib/python3.10/dist-packages/sklearn/utils/
validation.py:2739: UserWarning: X does not have valid feature names,
but LogisticRegression was fitted with feature names
  warnings.warn(
```

Building a Neural Network

```
# Standarise the data

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_standard = scaler.fit_transform(X_train)
X_test_standard = scaler.transform(X_test)
```

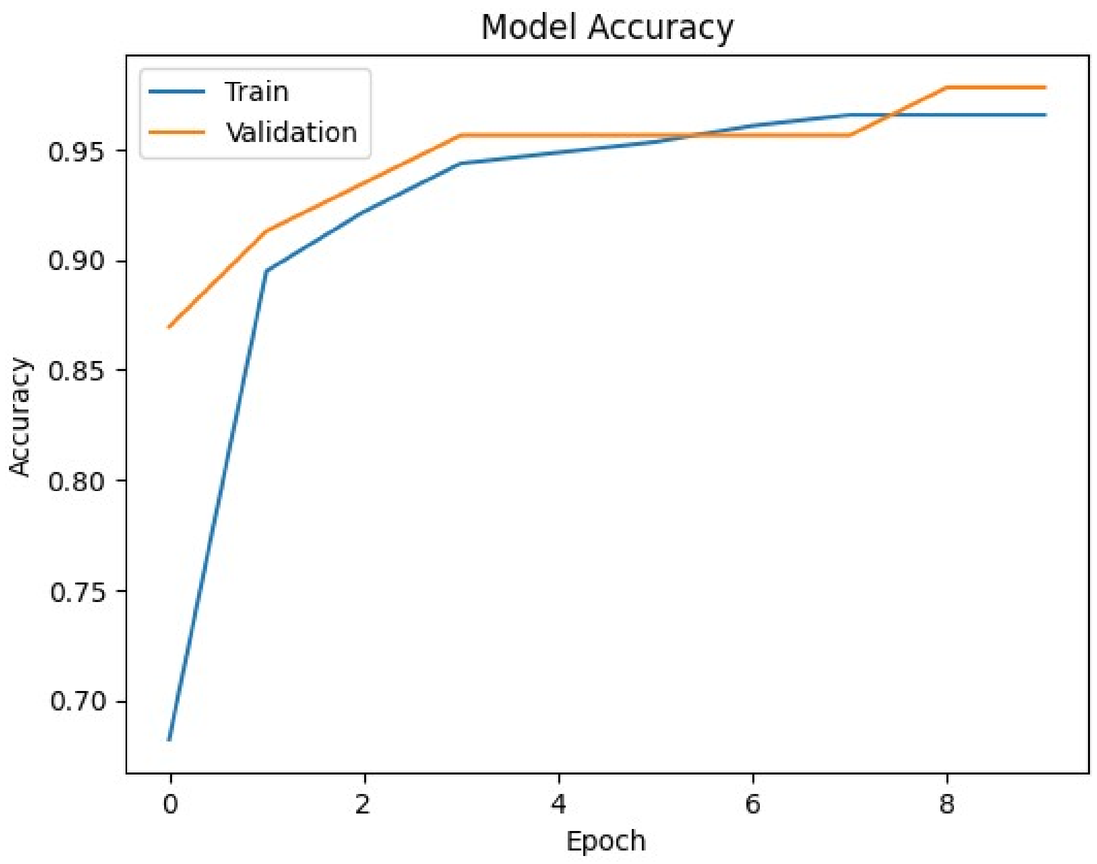
```
import tensorflow as tf
# Setting seed
tf.random.set_seed(3) # For reproducability

from tensorflow import keras

X.shape

(569, 30)

# Setting the layers for the neural network

model = keras.Sequential([
                            keras.layers.Flatten(input_shape = (30,)),
# layer 1 - input layer
                            keras.layers.Dense(20, activation =
'relu'),  # layer 2 - hidden layer with 20 neurons
                            keras.layers.Dense(2, activation =
'sigmoid')  # layer 3 - output layer
])

# The input is Flattened to a single dimensional array
# Dense means that all the neurons in the particular layer is
connected to all the neurons in the previous layer.
# Number of neurons in the output layer should be equal to the number
of class in the Target.

# Compiling the neural network

model.compile(optimizer = 'adam',
              loss = 'sparse_categorical_crossentropy', # Label
encoding
              metrics = ['accuracy']
              )

# Training the neural network

history = model.fit(X_train_standard, y_train, validation_split = 0.1,
epochs = 10)

Epoch 1/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5753 - loss:
0.7472 - val_accuracy: 0.8696 - val_loss: 0.3590
Epoch 2/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8758 - loss:
0.3308 - val_accuracy: 0.9130 - val_loss: 0.2176
Epoch 3/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9180 - loss:
0.2126 - val_accuracy: 0.9348 - val_loss: 0.1796
Epoch 4/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9493 - loss:
```

```
0.1670 - val_accuracy: 0.9565 - val_loss: 0.1623
Epoch 5/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9520 - loss:
0.1437 - val_accuracy: 0.9565 - val_loss: 0.1514
Epoch 6/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9580 - loss:
0.1298 - val_accuracy: 0.9565 - val_loss: 0.1427
Epoch 7/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9620 - loss:
0.1199 - val_accuracy: 0.9565 - val_loss: 0.1355
Epoch 8/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9683 - loss:
0.1120 - val_accuracy: 0.9565 - val_loss: 0.1292
Epoch 9/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9683 - loss:
0.1053 - val_accuracy: 0.9783 - val_loss: 0.1238
Epoch 10/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9662 - loss:
0.0994 - val_accuracy: 0.9783 - val_loss: 0.1191
```

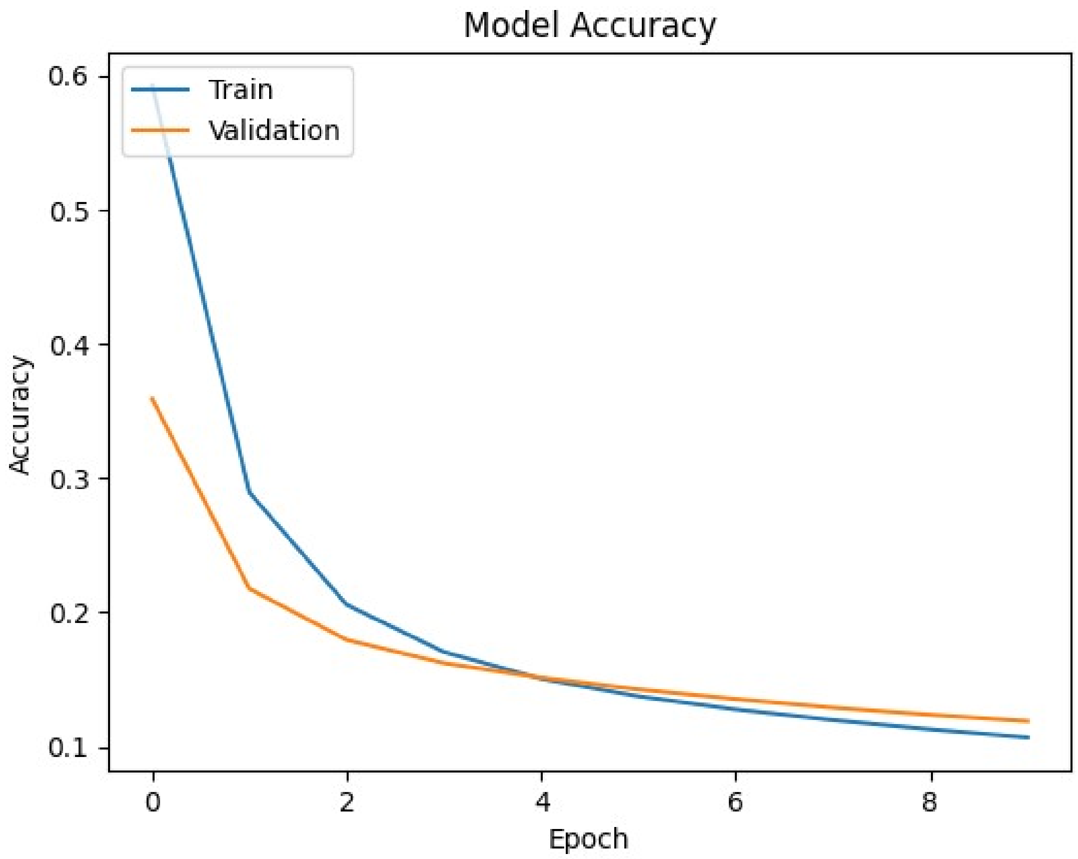
We can observe that at each epoch, the accuracy score is increasing and the loss is decreasing.

=> Loss function and accuracy are inversely proportional.

```python
# Plotting the accuracy and loss

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])

plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()
```



```
# Plotting the accuracy and loss

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()
```



```
# Accuracy of the model with the test data

# The model uses the test data to find y_pred(Prediction) and compares
it with y_test(True Value) and gives us the loss value and accuracy
loss, accuracy = model.evaluate(X_test_standard, y_test)
print(accuracy)

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9491 - loss: 0.1161

0.9561403393745422

print(X_test_standard.shape)
print("\nThe first sample is:")
print(X_test_standard[0])

(114, 30)

The first sample is:
[-0.46649743 -0.13728933 -0.44421138 -0.48646498  0.28085007
0.04160589
 -0.11146496 -0.26486866  0.41524141  0.13513744 -0.02091509 -
0.29323907
 -0.17460869 -0.2072995  -0.01181432 -0.35108921 -0.1810535  -
0.24238831
```

```
 -0.33731758 -0.0842133  -0.2632354  -0.14784208 -0.33154752 -
0.35109337
  0.48001942 -0.09649594 -0.03583041 -0.19435087  0.17275669
0.20372995]
```

```
y_pred = model.predict(X_test_standard) # gives the probability of
each class
print(y_pred.shape)
print("The prediction for the first sampleis:", y_pred[0])
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
(114, 2)
The prediction for the first sampleis: [0.23526107 0.6011053 ]
```

The model has 2 columns, the probablities of the datapoint being in each of the classes, meaning the model is 23.5% sure that the datapoint is Malignant and 60.1% sure that it is Benign.

```
print(y_pred)
```

```
[[0.23526107 0.6011053 ]
 [0.967012   0.17916565]
 [0.8809996  0.39203843]
 [0.09163497 0.76184845]
 [0.0518633  0.87765497]
 [0.997543   0.03363311]
 [0.9975007  0.03328505]
 [0.826613   0.42277163]
 [0.6363133  0.65119433]
 [0.04099468 0.7786242 ]
 [0.08233156 0.5207133 ]
 [0.76948696 0.28281498]
 [0.05472853 0.5836183 ]
 [0.4352791  0.5454692 ]
 [0.05069268 0.83709496]
 [0.5420243  0.16779551]
 [0.04411402 0.7464872 ]
 [0.00502491 0.81901234]
 [0.00185251 0.8099445 ]
 [0.9899725  0.08876199]
 [0.39394748 0.56767327]
 [0.09328163 0.71416384]
 [0.99696106 0.08862436]
 [0.00634144 0.6039927 ]
 [0.03394335 0.79270506]
 [0.08002656 0.82699513]
 [0.06072469 0.8766667 ]
 [0.06778082 0.6532684 ]
 [0.05446557 0.62938887]
 [0.9559362  0.27618167]
 [0.05411643 0.754101  ]
 [0.01126605 0.7540706 ]
 [0.0531918  0.8202688 ]
 [0.05752247 0.5818418 ]
 [0.01597529 0.7029848 ]
 [0.01669249 0.76145005]
 [0.45888856 0.57793117]
 [0.03493489 0.82175535]
 [0.96165407 0.3150286 ]
 [0.12034945 0.55135024]
 [0.02004644 0.8215898 ]
 [0.5882047  0.25820234]
 [0.06402855 0.7974846 ]
 [0.02627368 0.68792146]
 [0.0835228  0.549306  ]
 [0.10372543 0.6661686 ]
 [0.02972844 0.8801018 ]
 [0.00777749 0.76747096]
 [0.15875538 0.6356329 ]
 [0.08850199 0.7609377 ]
 [0.9634779  0.34864497]
 [0.9723907  0.06767032]
 [0.65766454 0.46065432]
 [0.33900797 0.58311003]
 [0.04361732 0.7901425 ]
 [0.12744749 0.67763054]
 [0.04798314 0.82949865]
 [0.99901146 0.00516789]
 [0.8659149  0.5371398 ]
 [0.01119028 0.7887057 ]
 [0.06462906 0.621182  ]
 [0.9917601  0.09564156]
 [0.9896463  0.0128862 ]
 [0.11215623 0.5670966 ]
 [0.01623746 0.6587871 ]
 [0.3202244  0.6376673 ]
 [0.9205148  0.14695016]
 [0.9960741  0.02753112]
 [0.01808445 0.760675  ]
 [0.19003838 0.711686  ]
 [0.56790584 0.2588996 ]
 [0.9079595  0.43957266]
 [0.06634501 0.56892574]
 [0.9246049  0.23878777]
 [0.00576476 0.7603435 ]
 [0.1334517  0.6001706 ]
 [0.10732    0.42613506]
 [0.6488602  0.6484444 ]
 [0.00486984 0.81886077]
 [0.15103422 0.7189332 ]
 [0.81270826 0.43475103]
 [0.00585179 0.7411847 ]
 [0.5792724  0.5233446 ]
 [0.9951409  0.03793609]
 [0.6241894  0.4972551 ]
 [0.9292105  0.13641322]
 [0.94384825 0.2213732 ]
 [0.8429959  0.08229268]
 [0.04146893 0.600876  ]
 [0.07614221 0.61682224]
 [0.09645507 0.791111  ]
 [0.4593728  0.60592794]
 [0.21548477 0.3536443 ]
 [0.00850809 0.5197519 ]
 [0.00946542 0.8385546 ]
 [0.01160217 0.84138703]
 [0.8301103  0.07711208]
 [0.9167709  0.14343035]
 [0.00574798 0.8308903 ]
 [0.8908912  0.19107935]
 [0.8219067  0.55518186]
 [0.00107736 0.8272977 ]
 [0.9718593  0.10672905]
 [0.9428982  0.24071299]
 [0.0553957  0.69913006]
 [0.13274953 0.46750888]
 [0.12862155 0.6439526 ]
 [0.9964425  0.03770087]
 [0.40135416 0.69930834]
 [0.16799852 0.67022586]
 [0.8898151  0.18716756]
 [0.03737961 0.73771644]
 [0.3370565  0.33216566]
 [0.96457285 0.0465486 ]]
```

If the first value is greater, then the output is 0. If the second value is greater then, the label is 1.

```
# Converting the probabilities into labels

y_labels = [np.argmax(i) for i in y_pred]
print(y_labels)
```

```
[1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0,
1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1,
1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1,
1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0]
```

Building a Predictive System

```
input_data = (19.81, 22.15, 130, 1260, 0.09831, 0.1027, 0.1479,
0.09498, 0.1582, 0.05395, 0.7582, 1.017, 5.865, 112.4, 0.006494,
0.01893, 0.03391, 0.01521, 0.01356, 0.001997, 27.32, 30.88, 186.8,
2398, 0.1512, 0.315, 0.5372, 0.2388, 0.2768, 0.07615)

# Converting the input data into numpy array
input_array = np.asarray(input_data)

# reshaping the input array
input_array_reshaped = input_array.reshape(1, -1)

# Standardise the input data
input_data_standard = scaler.transform(input_array_reshaped)

# Making the prediction
prediction = model.predict(input_data_standard)
print(prediction)
label = [np.argmax(prediction)]
print(label)

if label[0] == 0:
    print("The patient is Benign")
else:
    print("The patient is Malignant")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
[[0.99331516 0.06749561]]
[0]
The patient is Benign
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/
validation.py:2739: UserWarning: X does not have valid feature names,
but StandardScaler was fitted with feature names
  warnings.warn(
```