#Cáncer de mama en Wisconsin (diagnóstico)

1) ID number 2) Diagnosis (M = malignant, B = benign) 3-32)

Ten real-valued features are computed for each cell nucleus:

```
a) radius (mean of distances from center to points on the perimeter)
b) texture (standard deviation of gray-scale values)
c) perimeter
d) area
e) smoothness (local variation in radius lengths)
f) compactness (perimeter^2 / area - 1.0)
g) concavity (severity of concave portions of the contour)
h) concave points (number of concave portions of the contour)
i) symmetry
j) fractal dimension ("coastline approximation" - 1)
```

ESPAÑOL 1) Número de identificación 2) Diagnóstico (M = maligno, B = benigno) 3-32) Se calculan diez características de valor real para cada núcleo celular:

- a) radio (media de las distancias desde el centro hasta los puntos del perímetro)
- b) textura (desviación estándar de los valores de la escala de grises)
- c) perímetro
- d) área
- e) suavidad (variación local en las longitudes de los radios)
- f) compacidad (perímetro^2 / área - 1,0)
- g) concavidad (severidad de las porciones cóncavas del contorno)
- h) puntos cóncavos (número de porciones cóncavas del contorno)
- i) simetría
- j) dimensión fractal ("aproximación de la línea de costa" - 1)

Datos

UCI Machine Learning Repository. (2019). Retrieved January 3, 2025, from Uci.edu website: https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic

```
pip install ucimlrepo

Collecting ucimlrepo
  Downloading ucimlrepo-0.0.7-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: pandas>=1.0.0 in
/usr/local/lib/python3.11/dist-packages (from ucimlrepo) (2.2.2)
Requirement already satisfied: certifi>=2020.12.5 in
/usr/local/lib/python3.11/dist-packages (from ucimlrepo) (2024.12.14)
Requirement already satisfied: numpy>=1.23.2 in
```

```
/usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0-
>ucimlrepo) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0-
>ucimlrepo) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0-
>ucimlrepo) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0-
>ucimlrepo) (2025.1)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2-
>pandas>=1.0.0->ucimlrepo) (1.17.0)
Downloading ucimlrepo-0.0.7-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.7

from ucimlrepo import fetch_ucirepo

# fetch dataset
breast_cancer_wisconsin_diagnostic = fetch_ucirepo(id=17)

# data (as pandas dataframes)
X = breast_cancer_wisconsin_diagnostic.data.features
y = breast_cancer_wisconsin_diagnostic.data.targets

# metadata
print(breast_cancer_wisconsin_diagnostic.metadata)

# variable information
print(breast_cancer_wisconsin_diagnostic.variables)

{'uci_id': 17, 'name': 'Breast Cancer Wisconsin (Diagnostic)',
'repository_url':
'https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagno
stic', 'data_url':
'https://archive.ics.uci.edu/static/public/17/data.csv', 'abstract':
'Diagnostic Wisconsin Breast Cancer Database.', 'area': 'Health and
Medicine', 'tasks': ['Classification'], 'characteristics':
['Multivariate'], 'num_instances': 569, 'num_features': 30,
'feature_types': ['Real'], 'demographics': [], 'target_col':
['Diagnosis'], 'index_col': ['ID'], 'has_missing_values': 'no',
'missing_values_symbol': None, 'year_of_dataset_creation': 1993,
'last_updated': 'Fri Nov 03 2023', 'dataset_doi': '10.24432/C5DW2B',
'creators': ['William Wolberg', 'Olvi Mangasarian', 'Nick Street', 'W.
Street'], 'intro_paper': {'ID': 230, 'type': 'NATIVE', 'title':
'Nuclear feature extraction for breast tumor diagnosis', 'authors':
'W. Street, W. Wolberg, O. Mangasarian', 'venue': 'Electronic
imaging', 'year': 1993, 'journal': None, 'DOI': '10.1117/12.148698',
```

'URL': 'https://www.semanticscholar.org/paper/53f0fbb425bc14468eb3bf96b2e1d41ba8087f36', 'sha': None, 'corpus': None, 'arxiv': None, 'mag': None, 'acl': None, 'pmid': None, 'pmcid': None}, 'additional_info': {'summary': 'Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. A few of the images can be found at http://www.cs.wisc.edu/~street/images/\r\n\r\nSeparating plane described above was obtained using Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree Construction Via Linear Programming." Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society, pp. 97-101, 1992], a classification method which uses linear programming to construct a decision tree. Relevant features were selected using an exhaustive search in the space of 1-4 features and 1-3 separating planes.\r\n\r\nThe actual linear program used to obtain the separating plane in the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].\r\n\r\nThis database is also available through the UW CS ftp server:\r\nftp ftp.cs.wisc.edu\r\ncd math-prog/cpo-dataset/machine-learn/WDBC/', 'purpose': None, 'funded_by': None, 'instances_represent': None, 'recommended_data_splits': None, 'sensitive_data': None, 'preprocessing_description': None, 'variable_info': '1) ID number\r\n2) Diagnosis (M = malignant, B = benign)\r\n3-32)\r\n\r\nTen real-valued features are computed for each cell nucleus:\r\n\r\n\ta) radius (mean of distances from center to points on the perimeter)\r\n\tb) texture (standard deviation of gray-scale values)\r\n\tc) perimeter\r\n\td) area\r\n\te) smoothness (local variation in radius lengths)\r\n\tf) compactness (perimeter^2 / area - 1.0)\r\n\tg) concavity (severity of concave portions of the contour)\r\n\th) concave points (number of concave portions of the contour)\r\n\ti) symmetry \r\n\tj) fractal dimension ("coastline approximation" - 1)', 'citation': None}}

	name	role	type	demographic	description	units
0	ID	ID	Categorical	None	None	None
1	Diagnosis	Target	Categorical	None	None	None
2	radius1	Feature	Continuous	None	None	None
3	texture1	Feature	Continuous	None	None	None
4	perimeter1	Feature	Continuous	None	None	None
5	area1	Feature	Continuous	None	None	None
6	smoothness1	Feature	Continuous	None	None	None

```
None
7         compactness1  Feature   Continuous         None         None
None
8           concavity1  Feature   Continuous         None         None
None
9      concave_points1  Feature   Continuous         None         None
None
10           symmetry1  Feature   Continuous         None         None
None
11  fractal_dimension1  Feature   Continuous         None         None
None
12             radius2  Feature   Continuous         None         None
None
13            texture2  Feature   Continuous         None         None
None
14          perimeter2  Feature   Continuous         None         None
None
15               area2  Feature   Continuous         None         None
None
16         smoothness2  Feature   Continuous         None         None
None
17        compactness2  Feature   Continuous         None         None
None
18          concavity2  Feature   Continuous         None         None
None
19     concave_points2  Feature   Continuous         None         None
None
20           symmetry2  Feature   Continuous         None         None
None
21  fractal_dimension2  Feature   Continuous         None         None
None
22             radius3  Feature   Continuous         None         None
None
23            texture3  Feature   Continuous         None         None
None
24          perimeter3  Feature   Continuous         None         None
None
25               area3  Feature   Continuous         None         None
None
26         smoothness3  Feature   Continuous         None         None
None
27        compactness3  Feature   Continuous         None         None
None
28          concavity3  Feature   Continuous         None         None
None
29     concave_points3  Feature   Continuous         None         None
None
30           symmetry3  Feature   Continuous         None         None
None
```

```
31  fractal_dimension3  Feature   Continuous        None        None
None
```

	missing_values
0	no
1	no
2	no
3	no
4	no
5	no
6	no
7	no
8	no
9	no
10	no
11	no
12	no
13	no
14	no
15	no
16	no
17	no
18	no
19	no
20	no
21	no
22	no
23	no
24	no
25	no
26	no
27	no
28	no
29	no
30	no
31	no

X


```
{"type":"dataframe","variable_name":"X"}
```


y


```
{"summary":"{\n  \"name\": \"y\",\n  \"rows\": 569,\n  \"fields\": [\n
{\n      \"column\": \"Diagnosis\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"B\",\n          \"M\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"y"}
```

Librerias

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import sklearn.datasets
import matplotlib.pyplot as plt
import seaborn as sns
```

- NumPy (numpy): Utilizada para operaciones numéricas y manipulación de arreglos.
- Pandas (pandas): Facilita la manipulación y análisis de datos a través de estructuras de datos como DataFrames.
- Scikit-Learn (sklearn): Proporciona herramientas para crear modelos de machine learning, incluyendo preprocesamiento, entrenamiento y evaluación.
- LabelEncoder: Para convertir etiquetas categóricas en valores numéricos.
- train_test_split: Para dividir los datos en conjuntos de entrenamiento y prueba.
- LogisticRegression: Implementa el algoritmo de regresión logística. accuracy_score: Para calcular la precisión del modelo.
- Matplotlib (matplotlib.pyplot): Usada para crear visualizaciones gráficas.

Exploración visual

```
# El conjunto de datos sobre cáncer de mama se carga utilizando:
breast_cancer_dataset = sklearn.datasets.load_breast_cancer()

#Se convierte el conjunto de datos en un DataFrame de Pandas:
data_frame = pd.DataFrame(breast_cancer_dataset.data, columns =
breast_cancer_dataset.feature_names)

# Se añade una columna que representa la etiqueta (benigno o maligno):
data_frame['label'] = breast_cancer_dataset.target

# print the first 5 rows of the dataframe
data_frame.head()
```

```
{"type":"dataframe","variable_name":"data_frame"}
```

```
data_frame.shape
```

```
(569, 31)
```

```
data_frame.info()
```

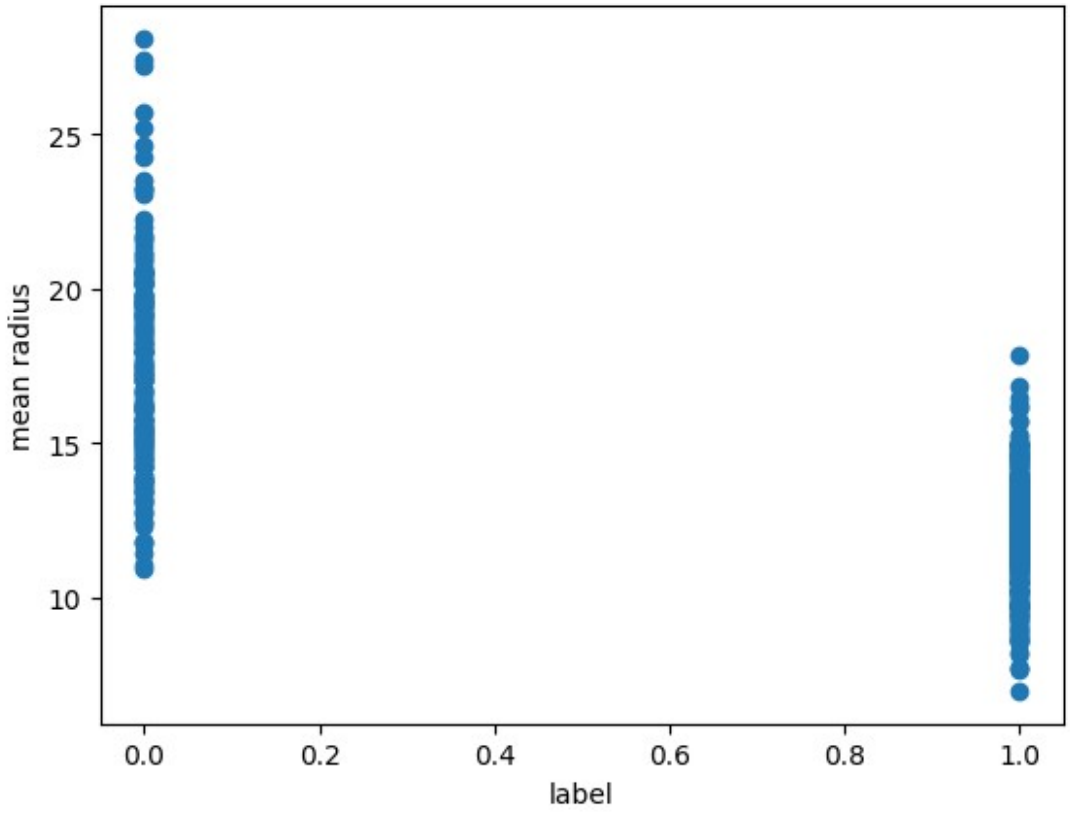
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
```

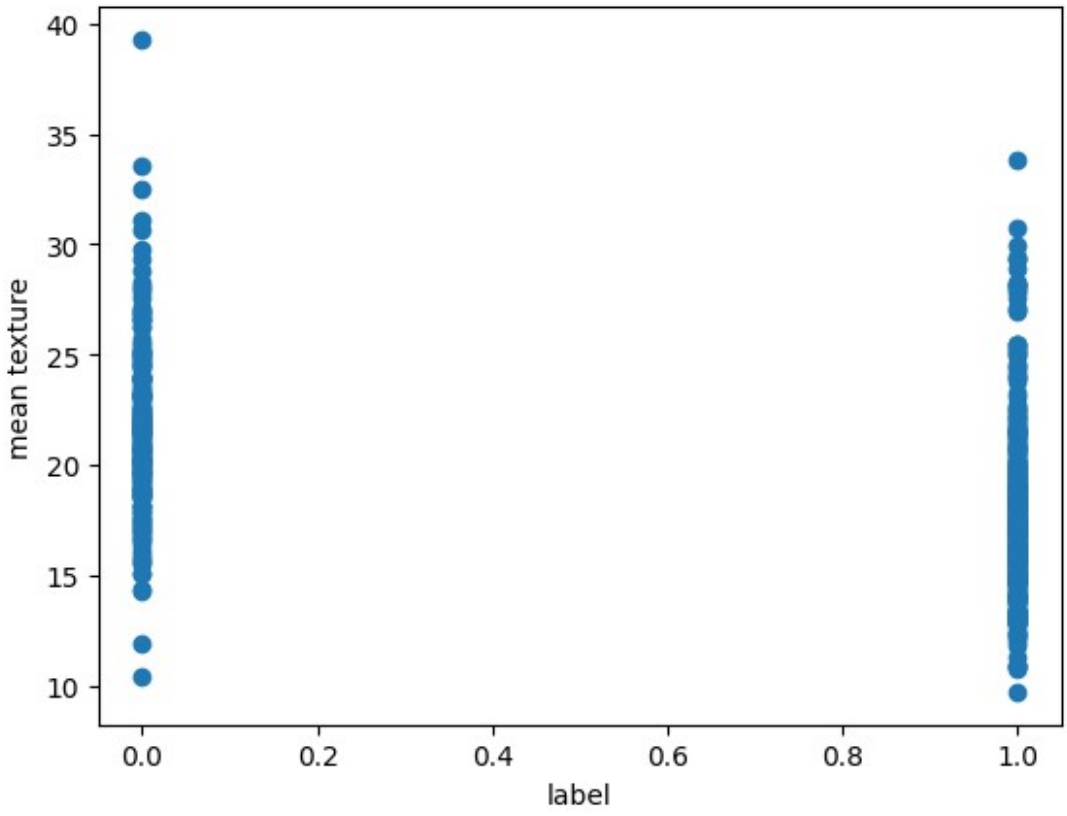
```
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 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  label                    569 non-null    int64  
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
```

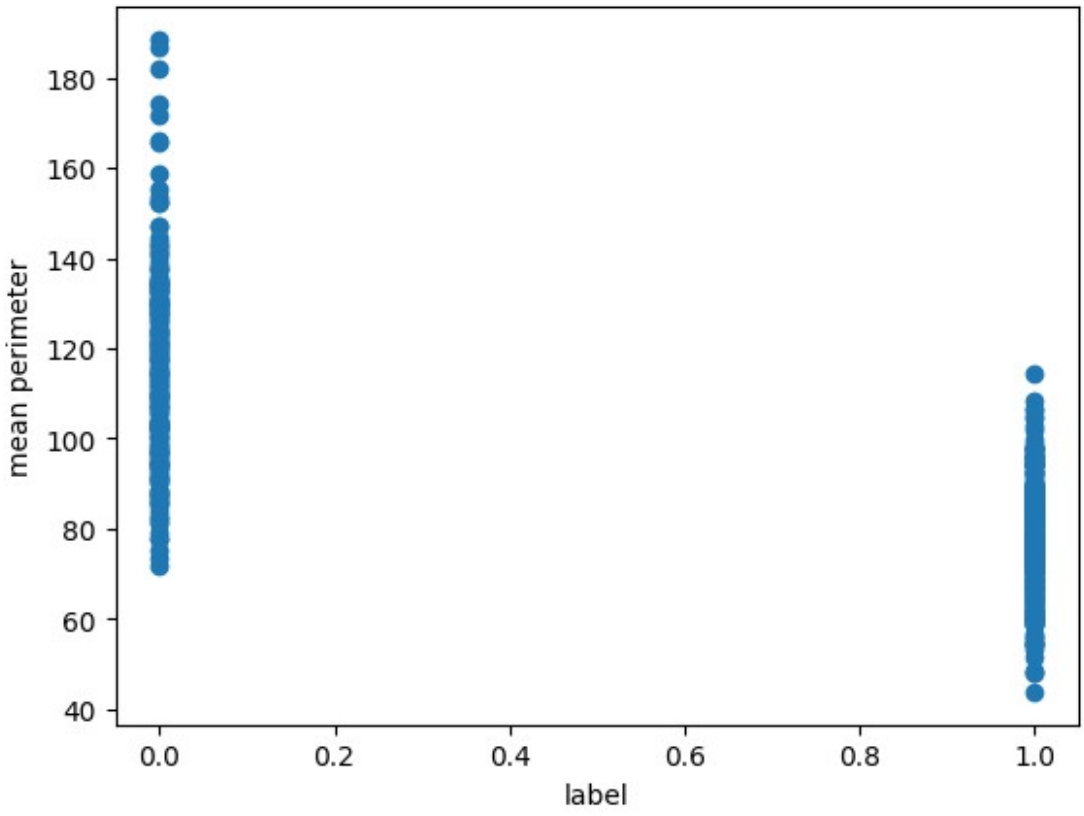
```
data_frame.columns
```

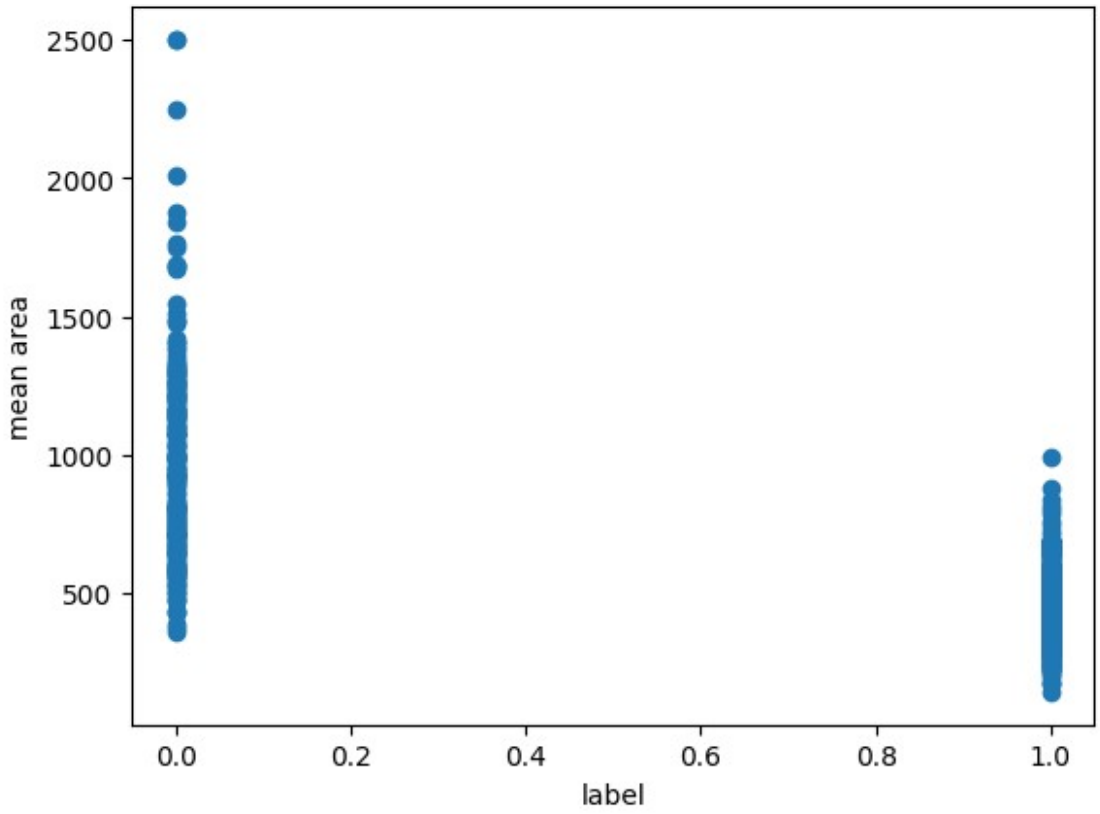
```
Index(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal
dimension',
       'radius error', 'texture error', 'perimeter error', 'area
error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error', 'fractal dimension
error',
       'worst radius', 'worst texture', 'worst perimeter', 'worst
area',
```

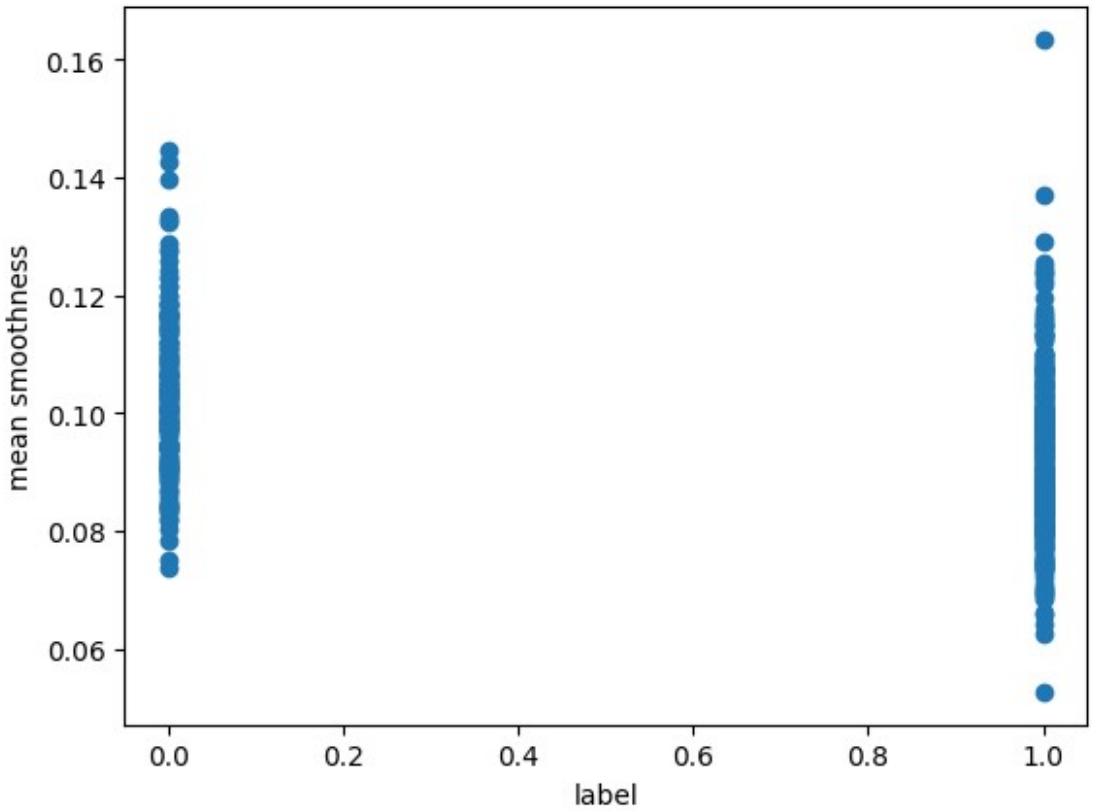
```
       'worst smoothness', 'worst compactness', 'worst concavity',
       'worst concave points', 'worst symmetry', 'worst fractal
dimension',
       'label'],
      dtype='object')

# Crear un diagrama de dispersión para cada grafico

for i in data_frame.columns:
    if i != 'label':
        plt.scatter(data_frame['label'], data_frame[i])
        plt.xlabel("label")
        plt.ylabel(i)
        plt.show()
```

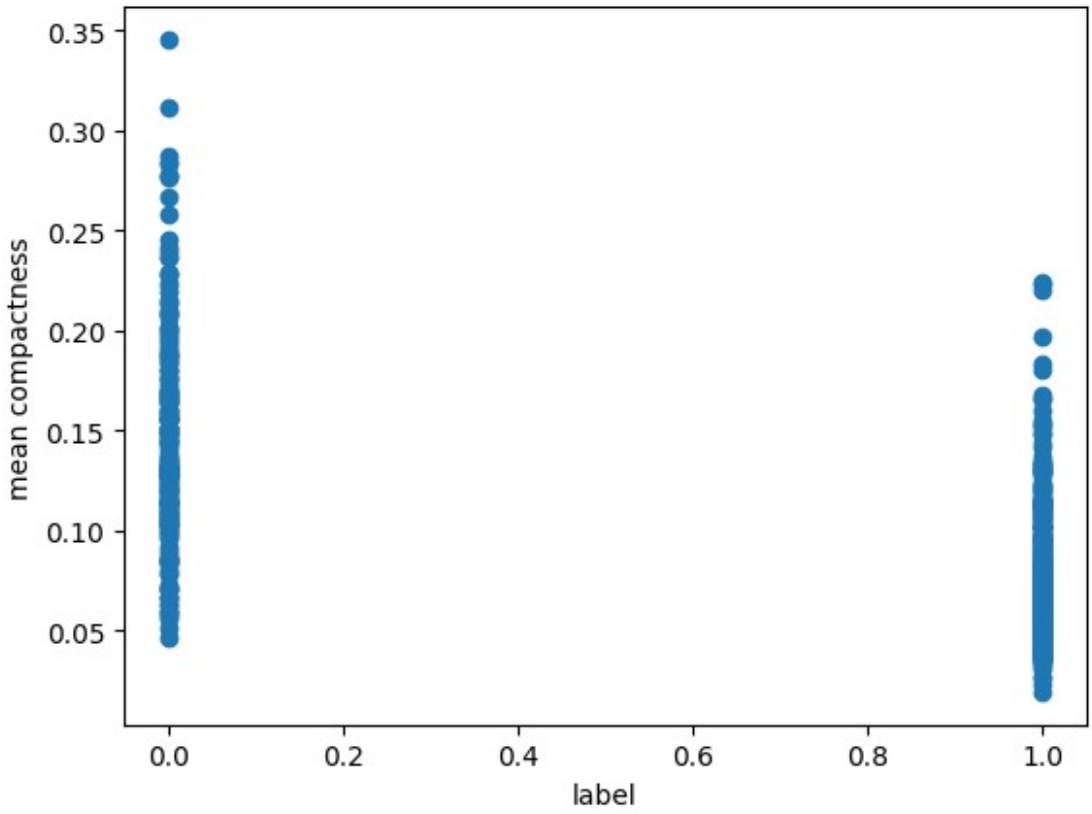


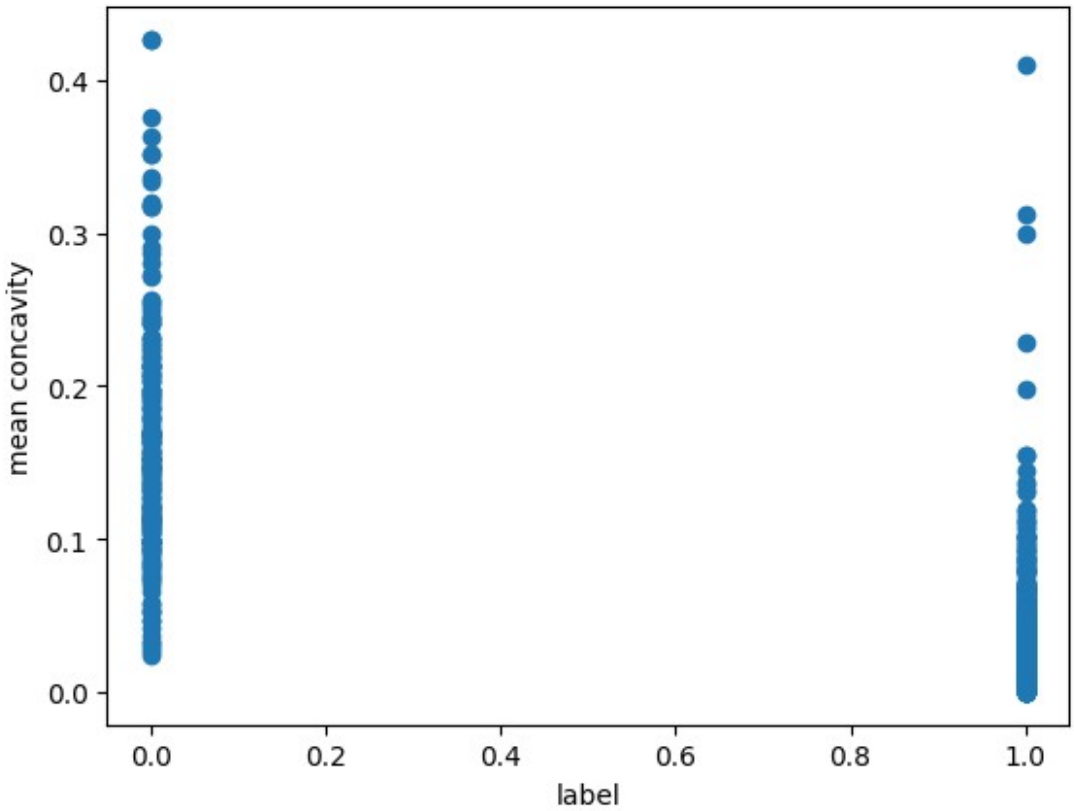

mean texture
label
40
35
30
25
20
15
10
0.0
0.2
0.4
0.6
0.8
1.0

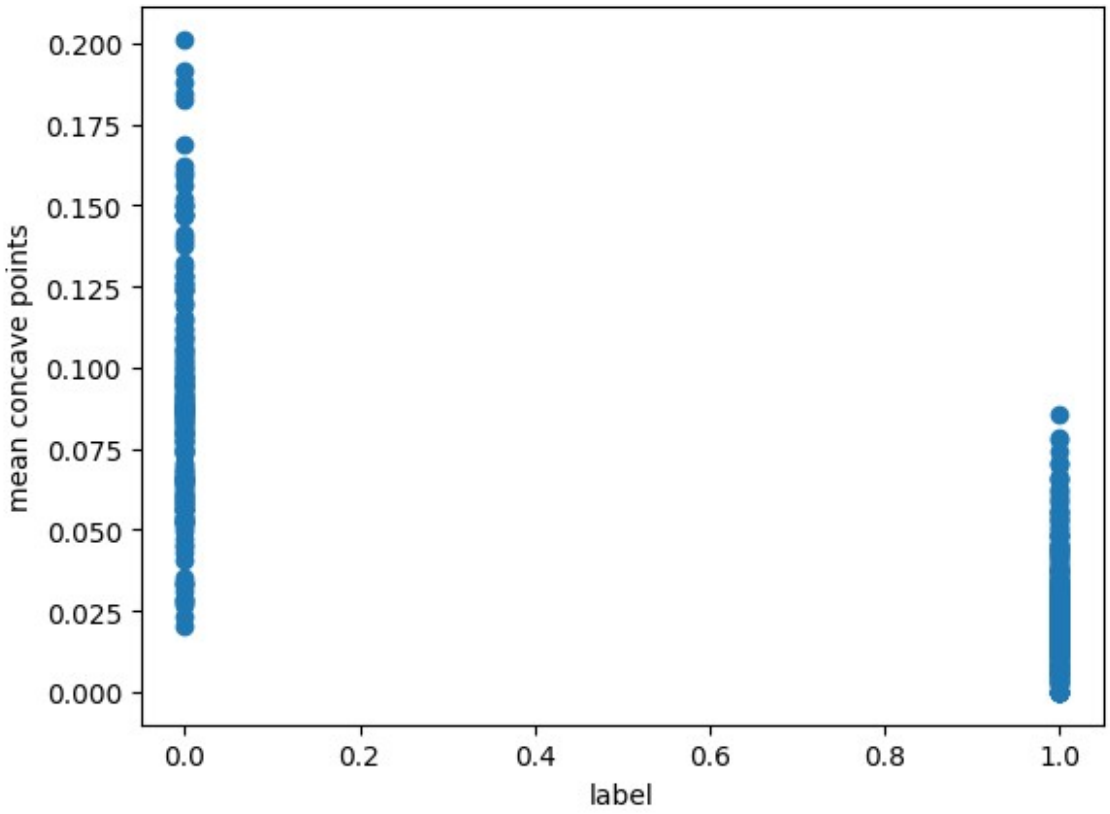

mean perimeter
label
180
160
140
120
100
80
60
40
0.0
0.2
0.4
0.6
0.8
1.0

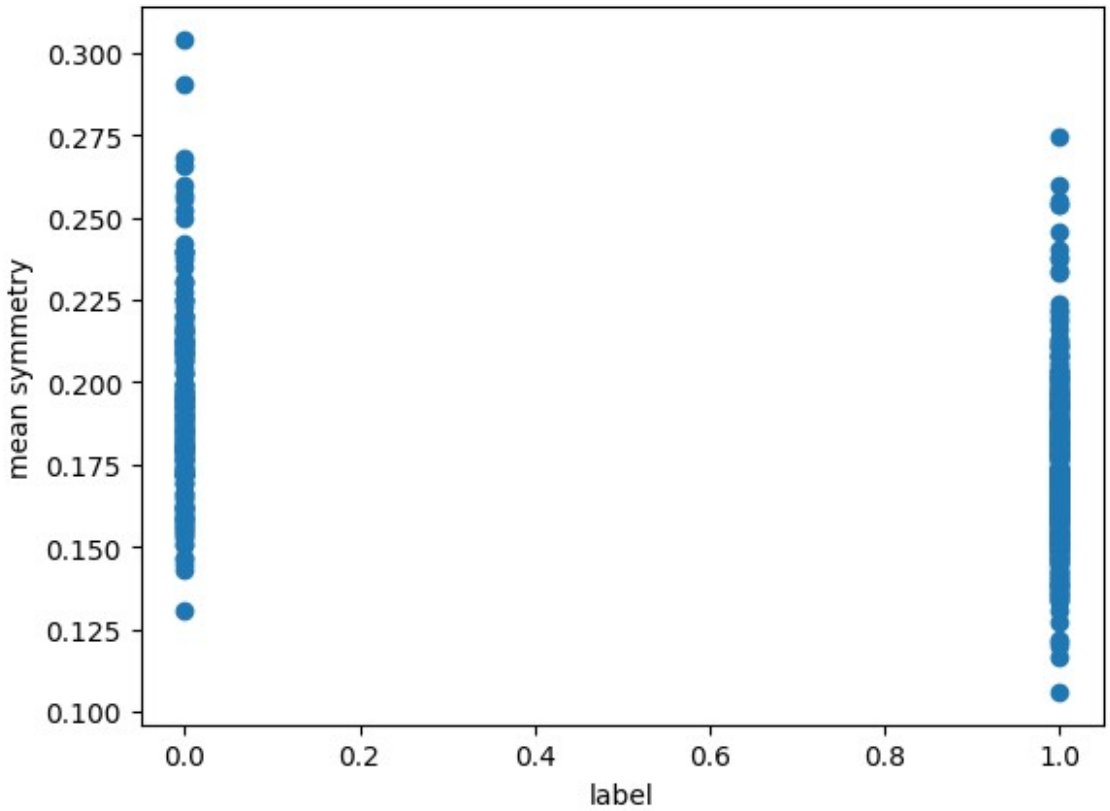

2500
2000
1500
1000
500
mean area
0.0
0.2
0.4
0.6
0.8
1.0
label

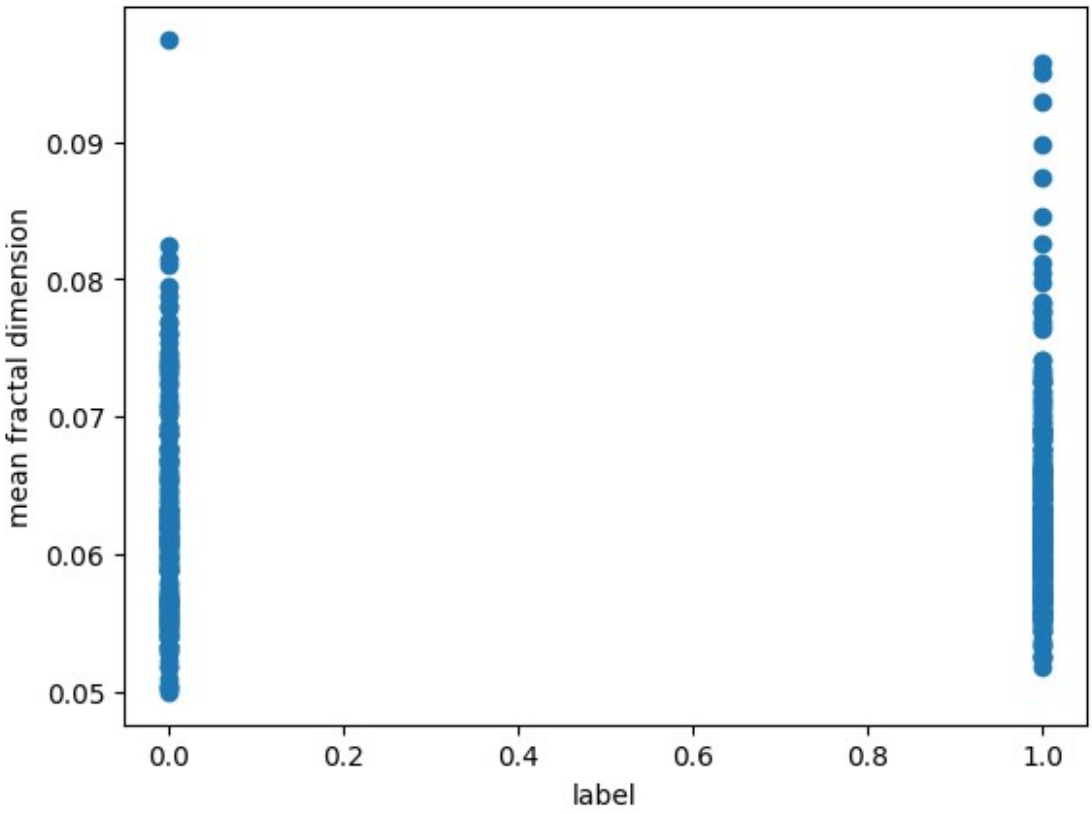

0.16
0.14
0.12
0.10
0.08
0.06
mean smoothness
0.0
0.2
0.4
0.6
0.8
1.0
label

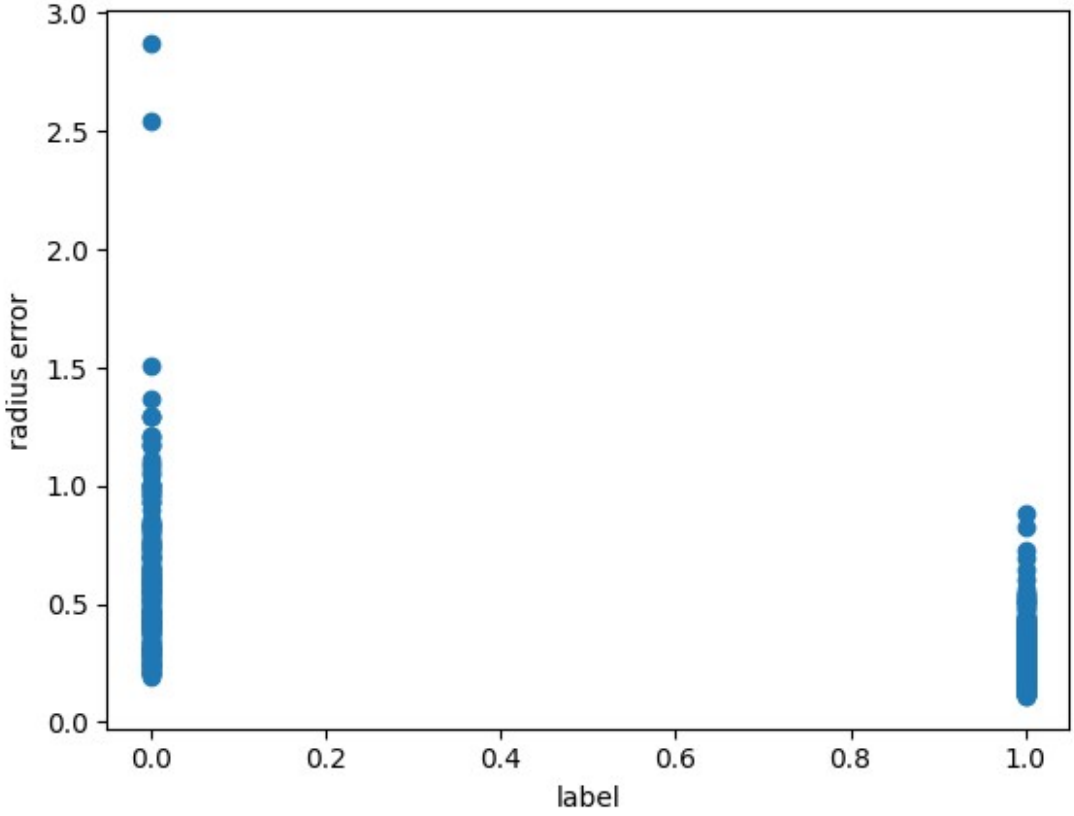

0.35
0.30
0.25
0.20
0.15
0.10
0.05
mean compactness
0.0
0.2
0.4
0.6
0.8
1.0
label

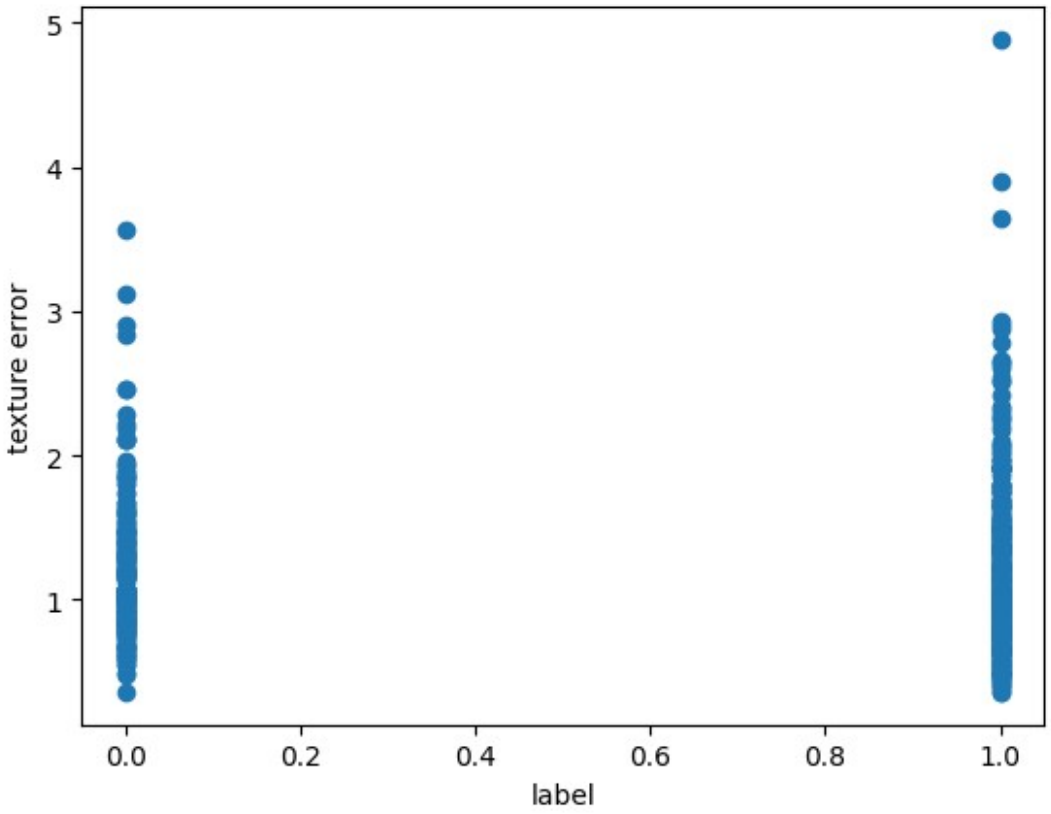

0.4
0.3
0.2
0.1
0.0
mean concavity
0.0
0.2
0.4
0.6
0.8
1.0
label

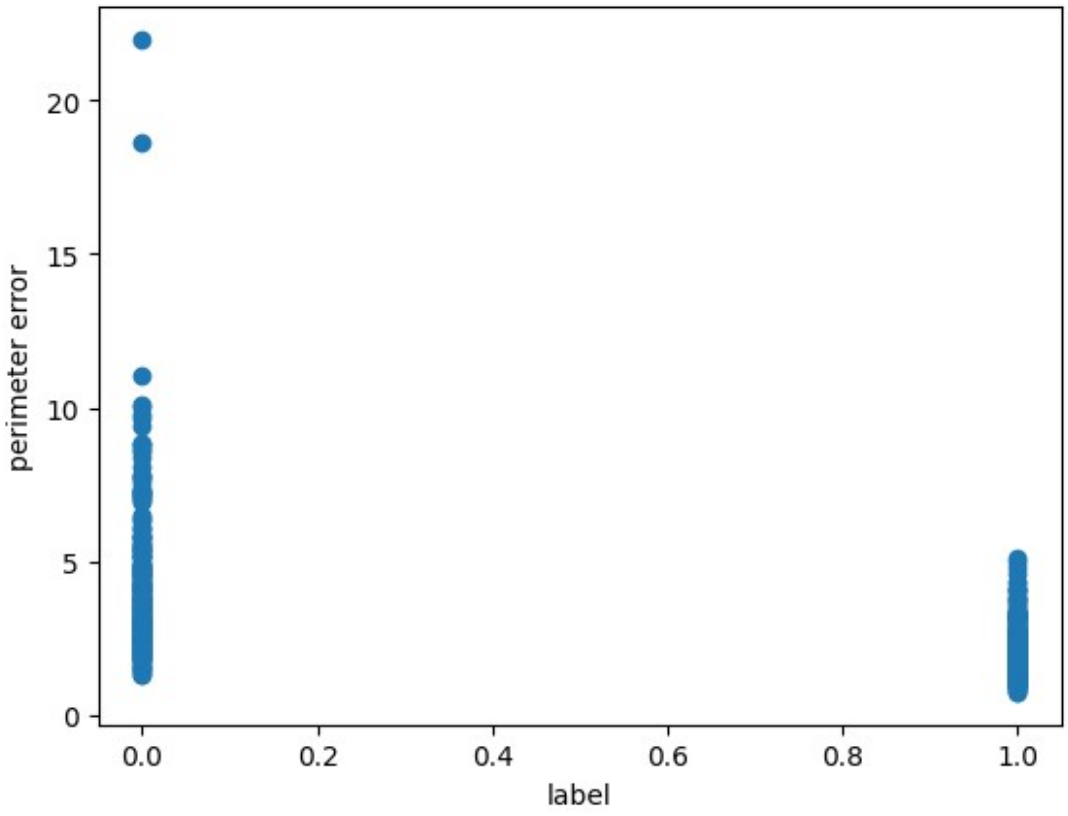

mean concave points
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0
0.2
0.4
0.6
0.8
1.0
label

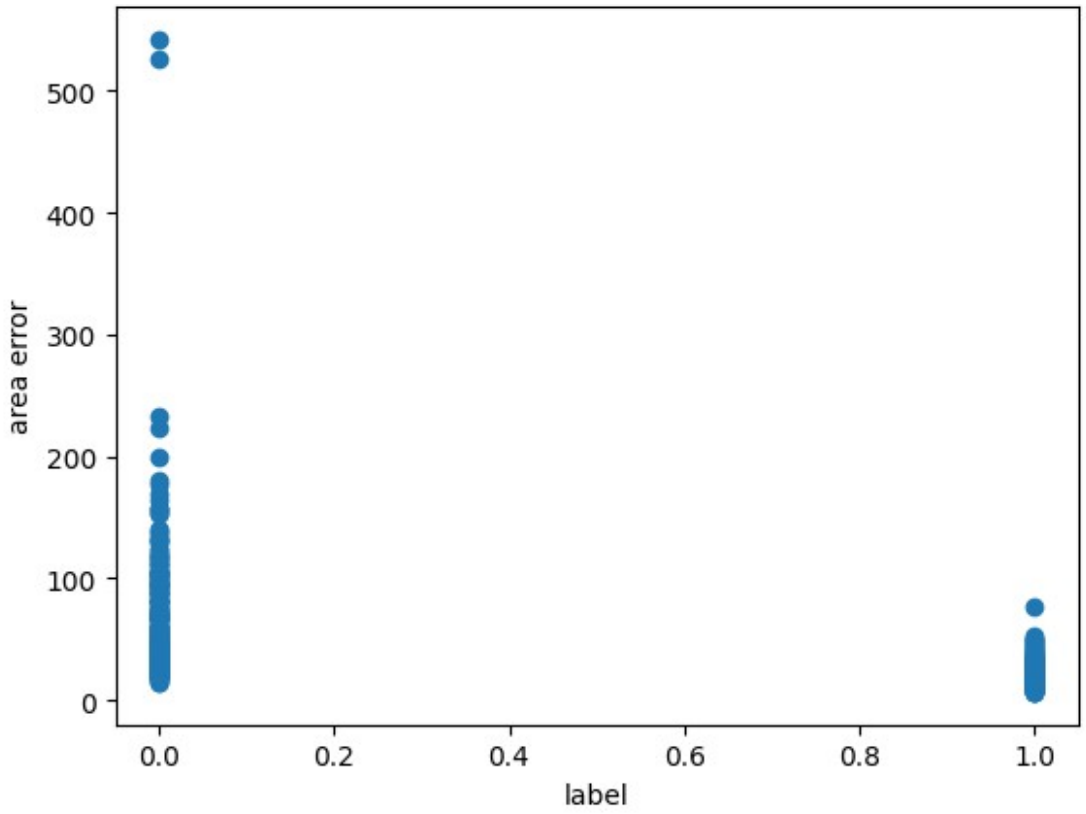

mean symmetry
0.300
0.275
0.250
0.225
0.200
0.175
0.150
0.125
0.100
0.0
0.2
0.4
0.6
0.8
1.0
label

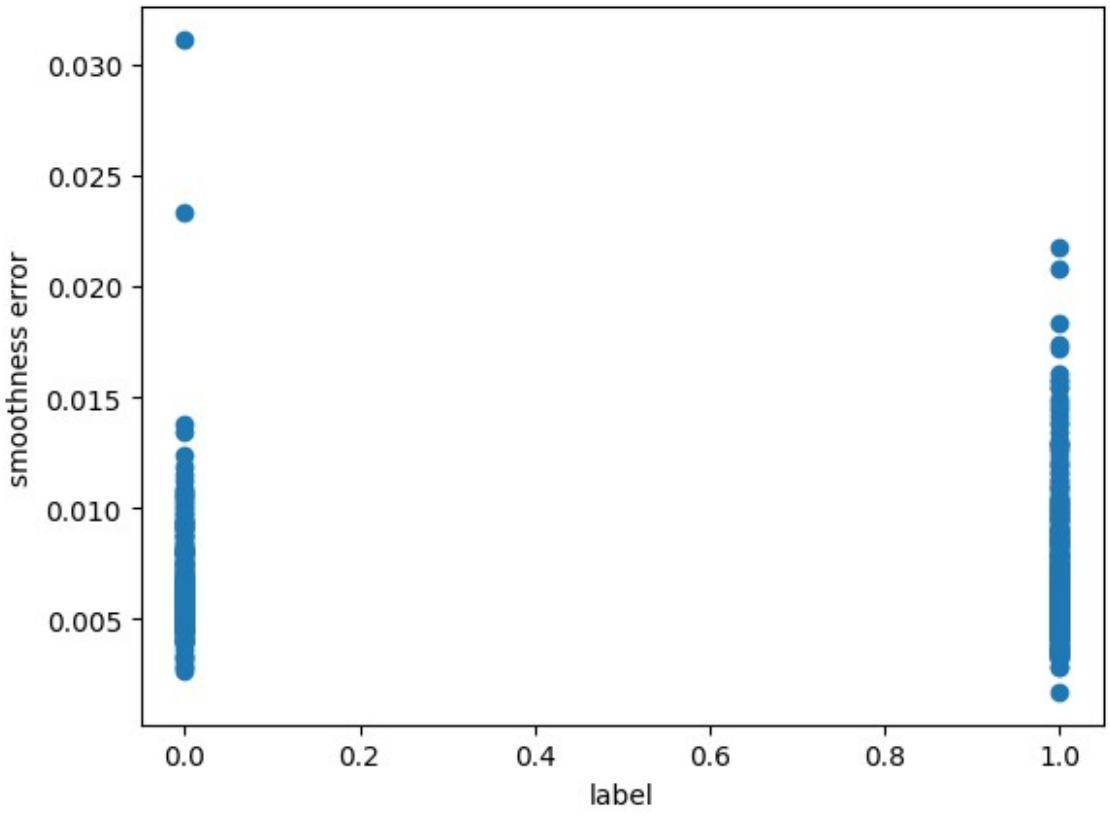

mean fractal dimension
0.09
0.08
0.07
0.06
0.05
0.0
0.2
0.4
0.6
0.8
1.0
label

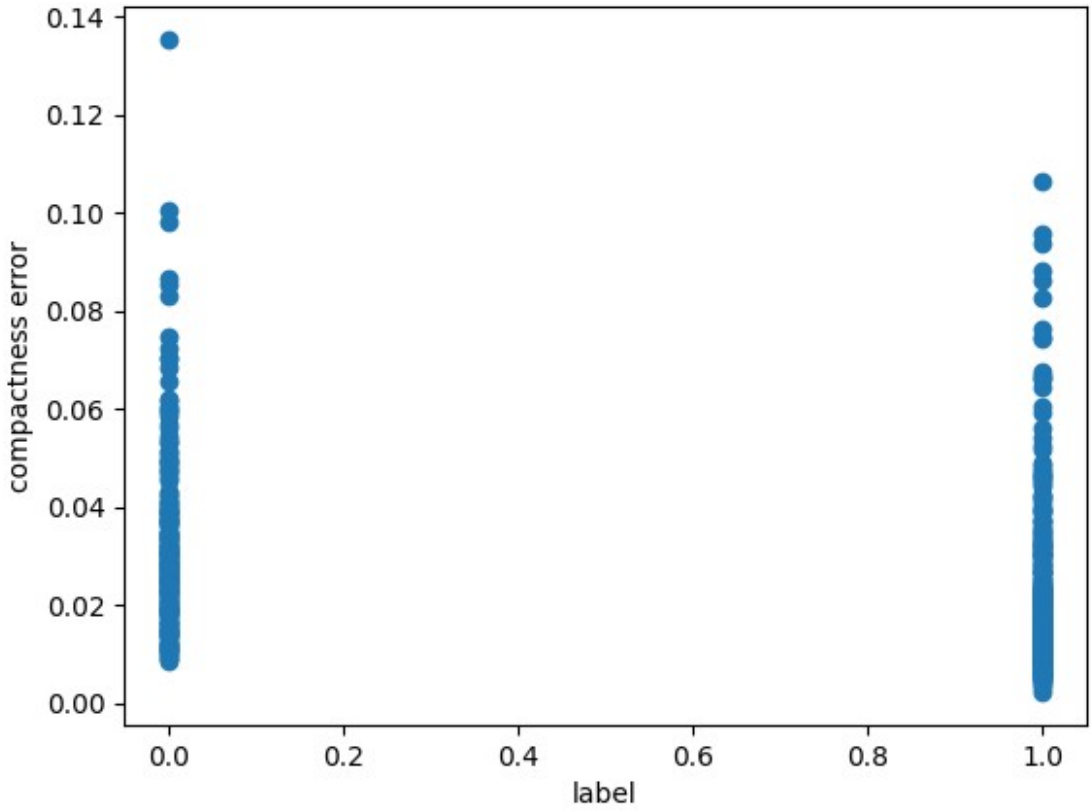

radius error
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0
label

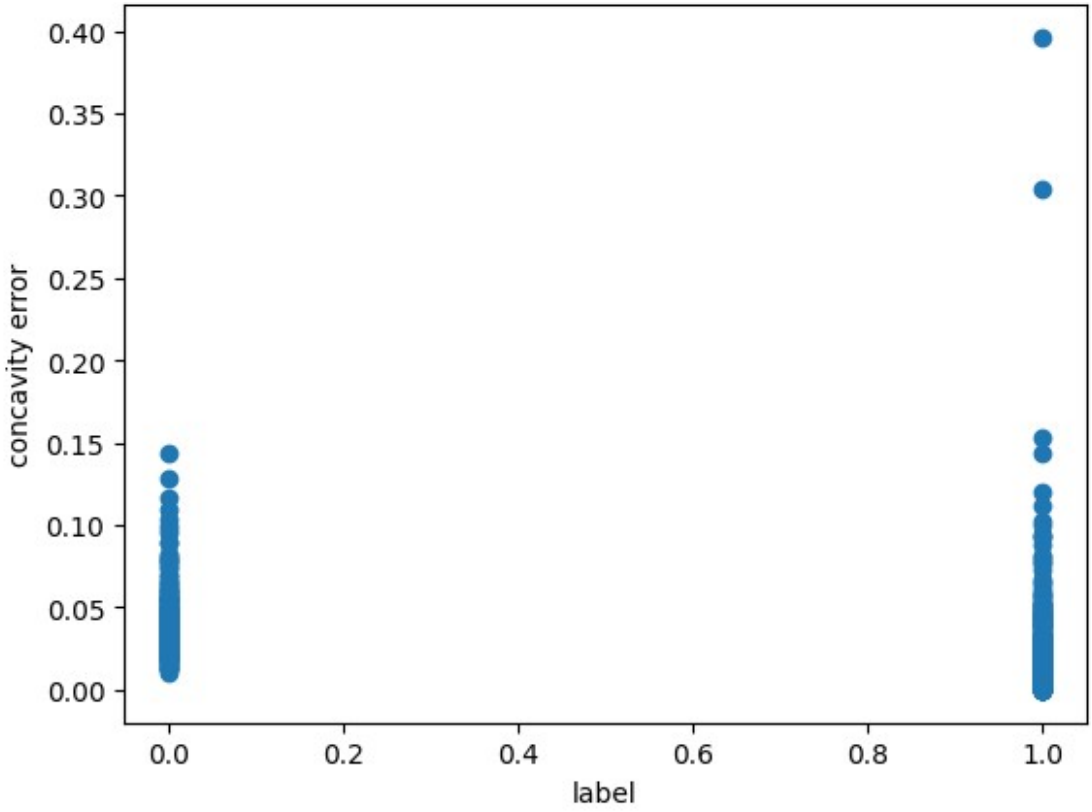

texture error
label
0.0
0.2
0.4
0.6
0.8
1.0
1
2
3
4
5

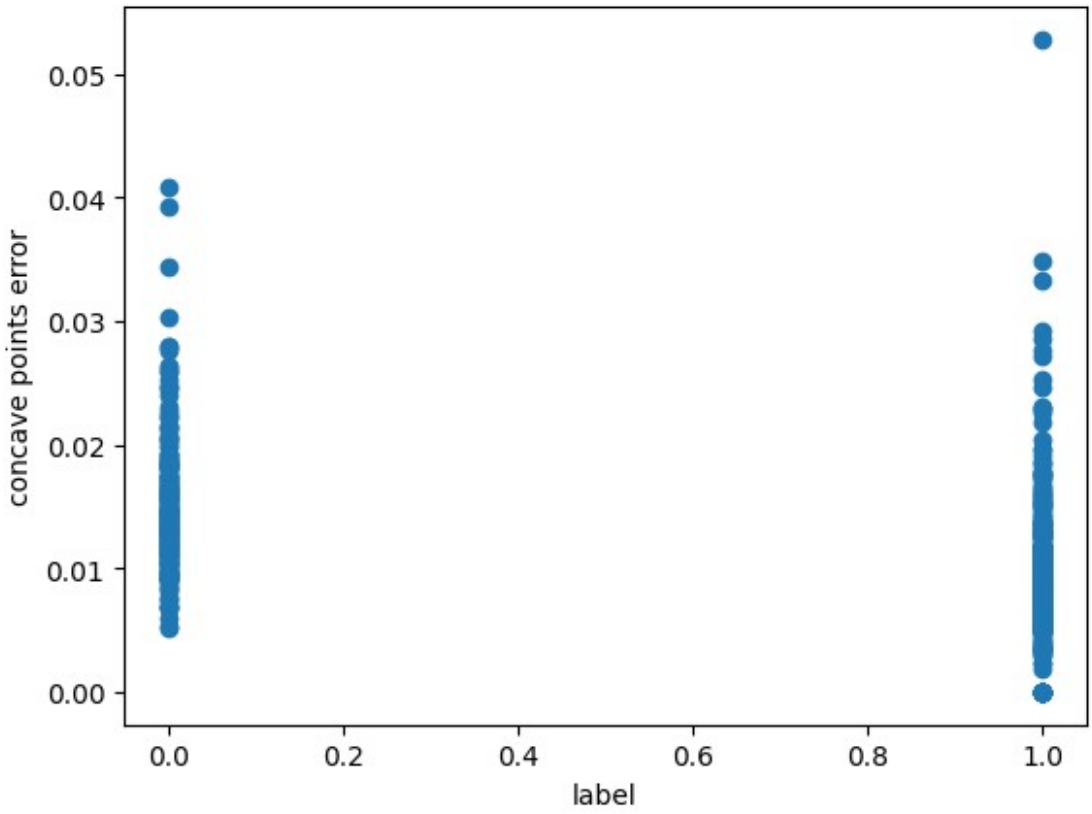

perimeter error
label
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20

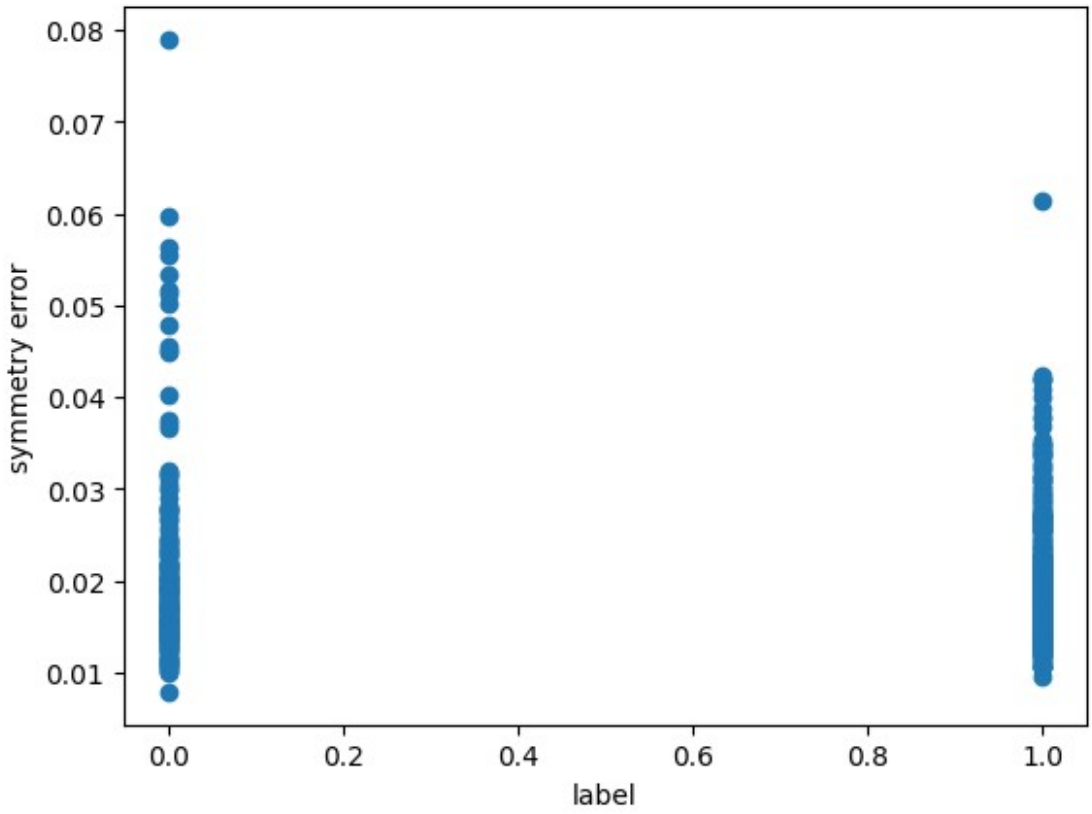

area error
500
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0
label

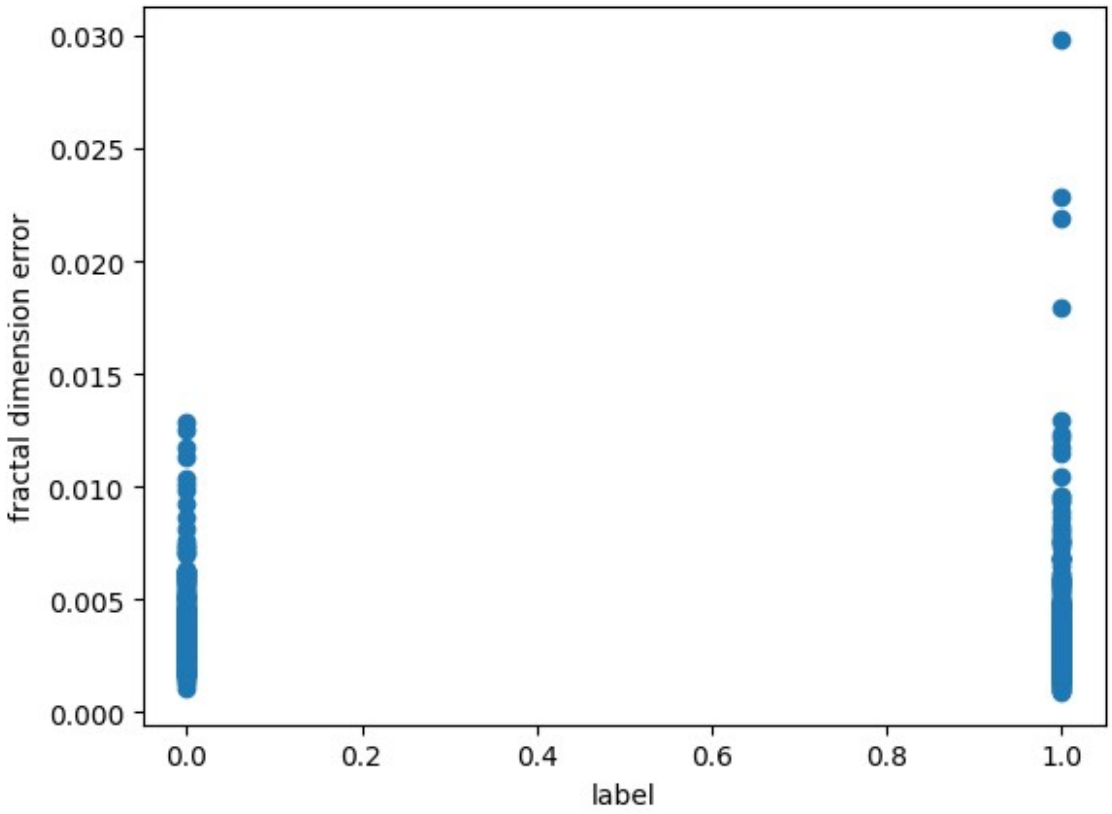

smoothness error
0.030
0.025
0.020
0.015
0.010
0.005
0.0
0.2
0.4
0.6
0.8
1.0
label

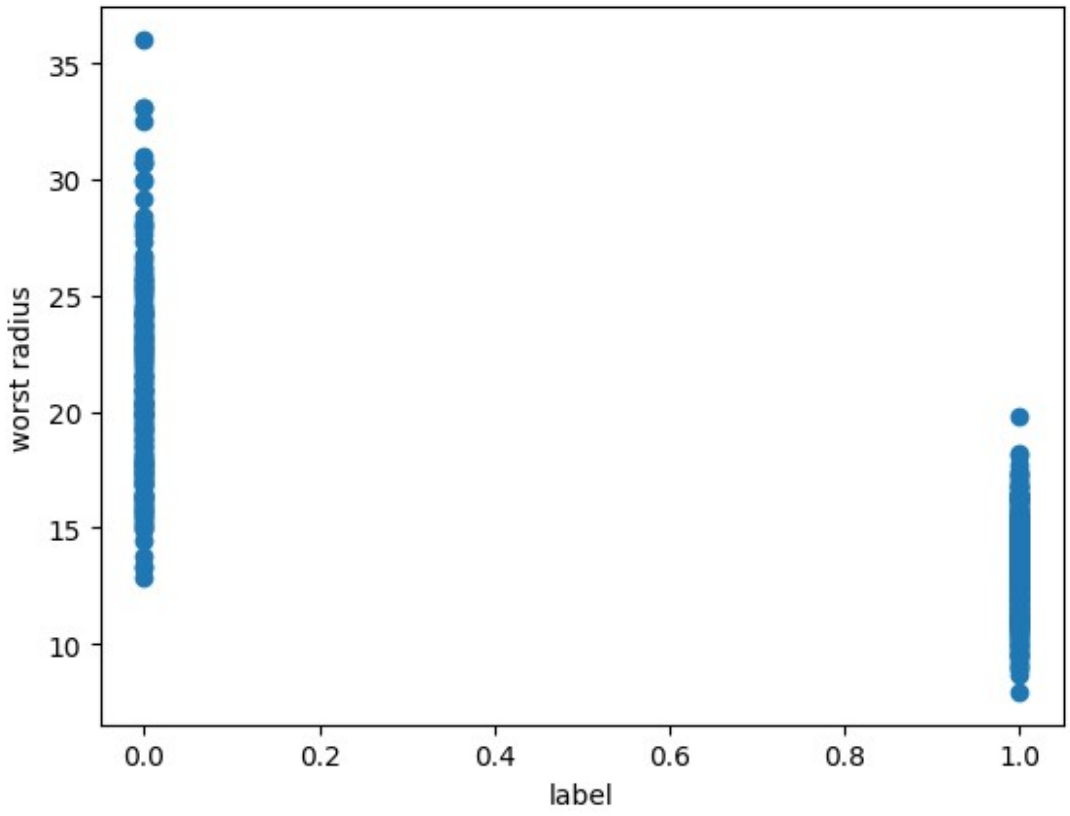

0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
compactness error
0.0
0.2
0.4
0.6
0.8
1.0
label

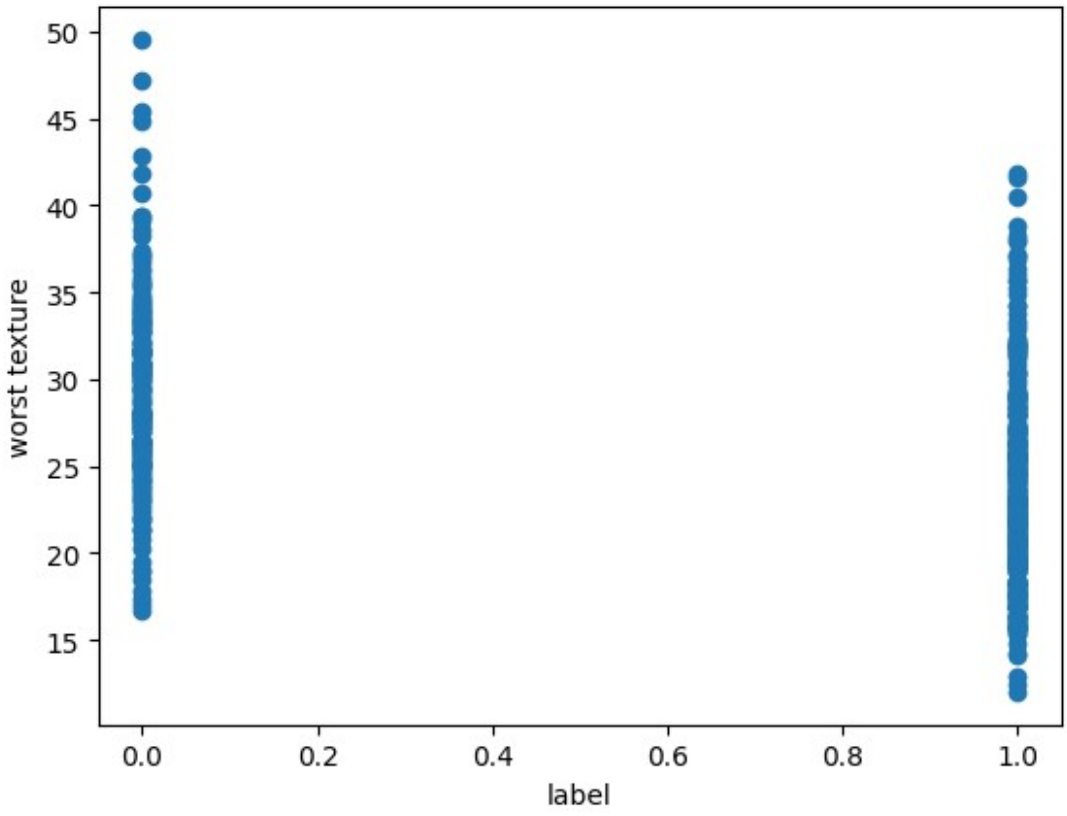

0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
concavity error
0.0
0.2
0.4
0.6
0.8
1.0
label

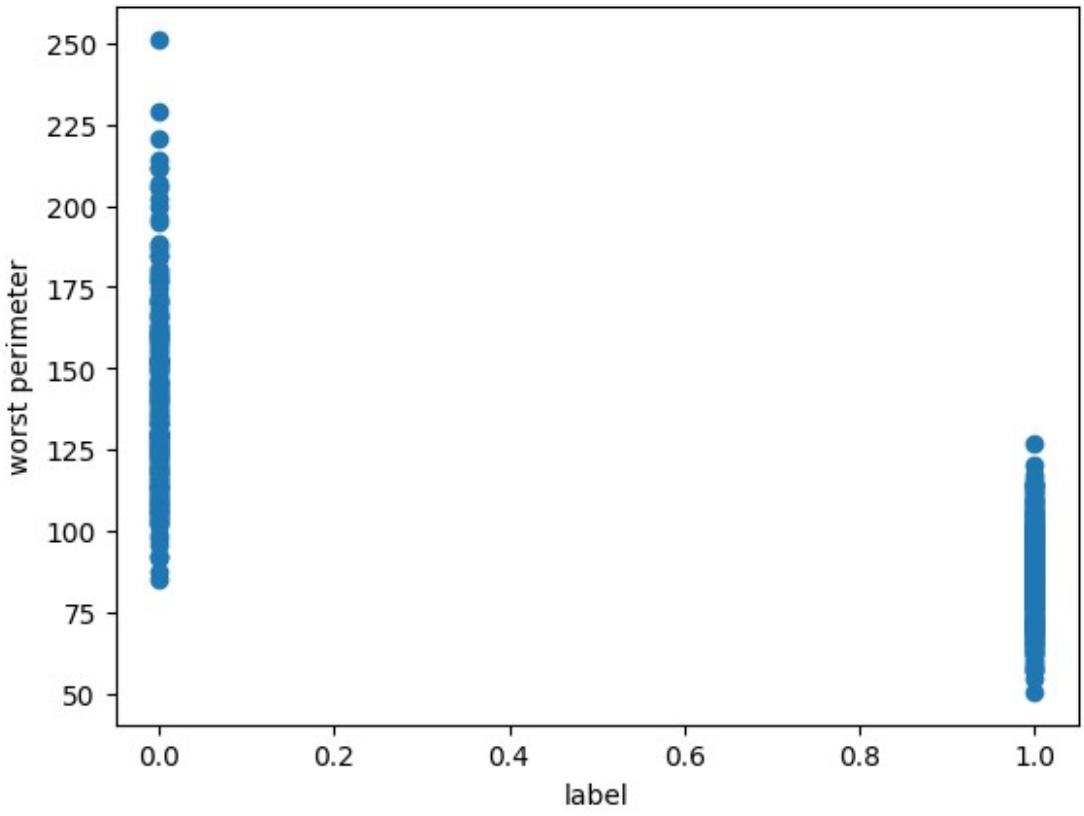

concave points error
0.05
0.04
0.03
0.02
0.01
0.00
0.0
0.2
0.4
0.6
0.8
1.0
label

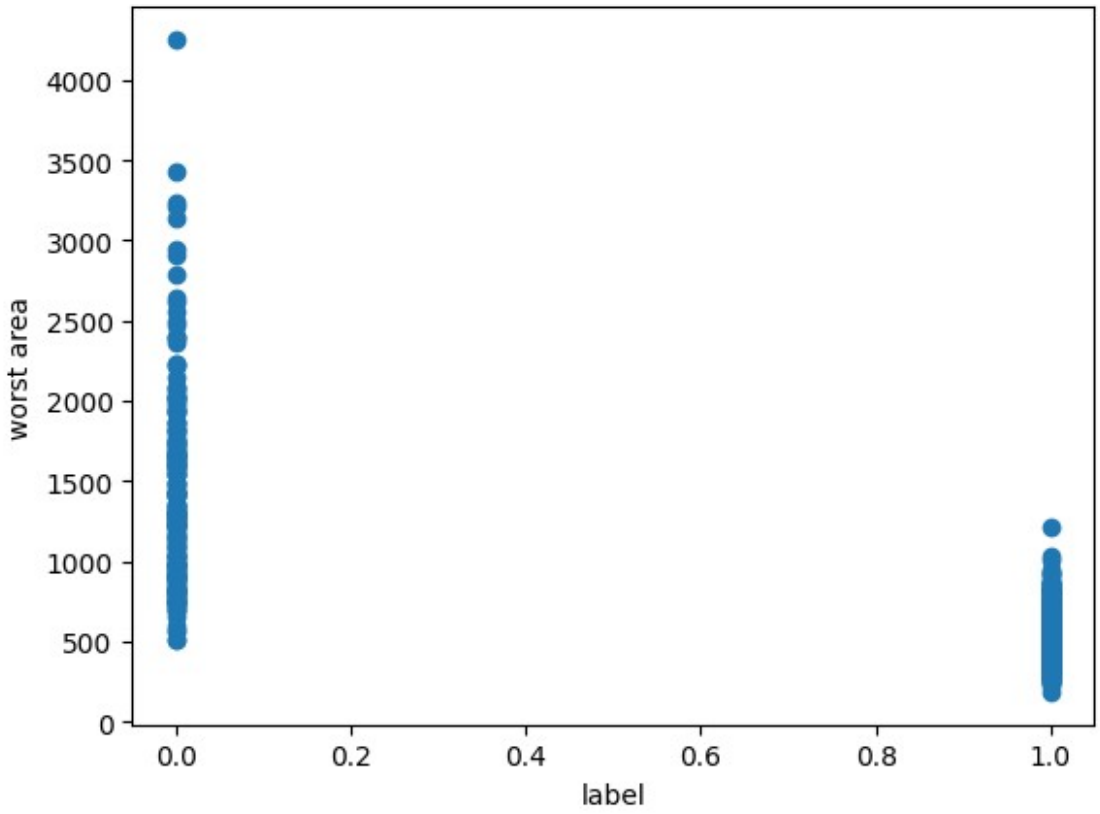

symmetry error
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.0
0.2
0.4
0.6
0.8
1.0
label

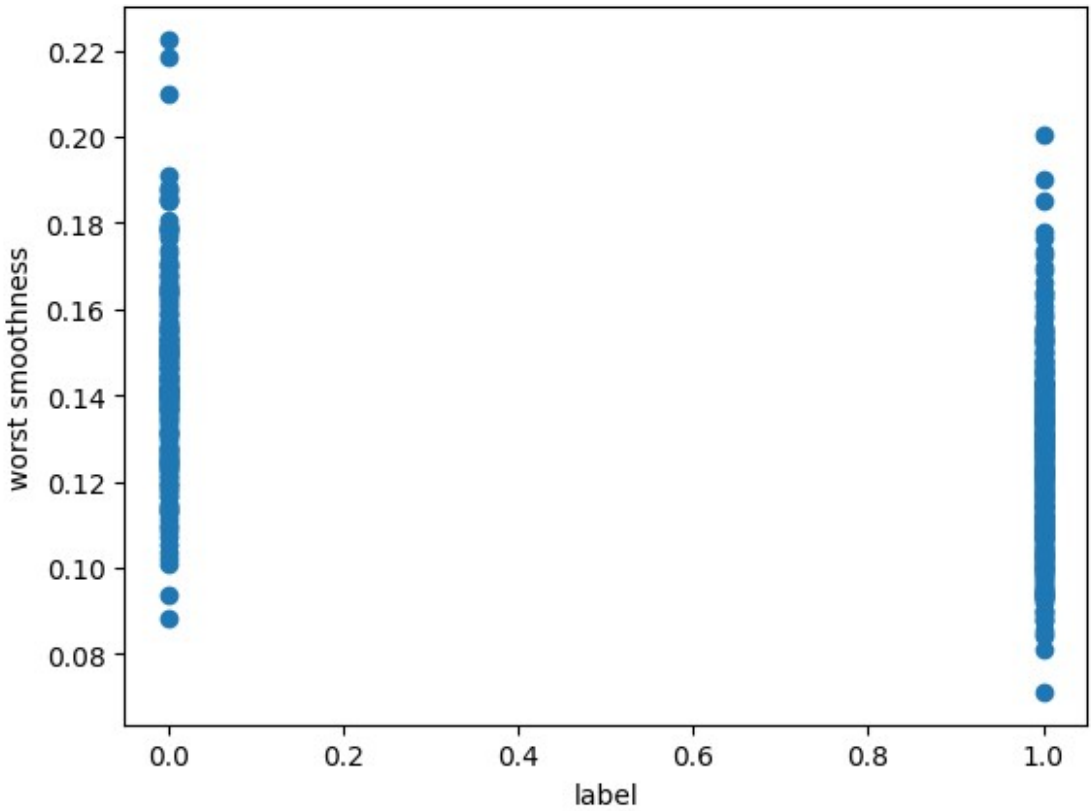

0.030
0.025
0.020
0.015
0.010
0.005
0.000
fractal dimension error
0.0
0.2
0.4
0.6
0.8
1.0
label

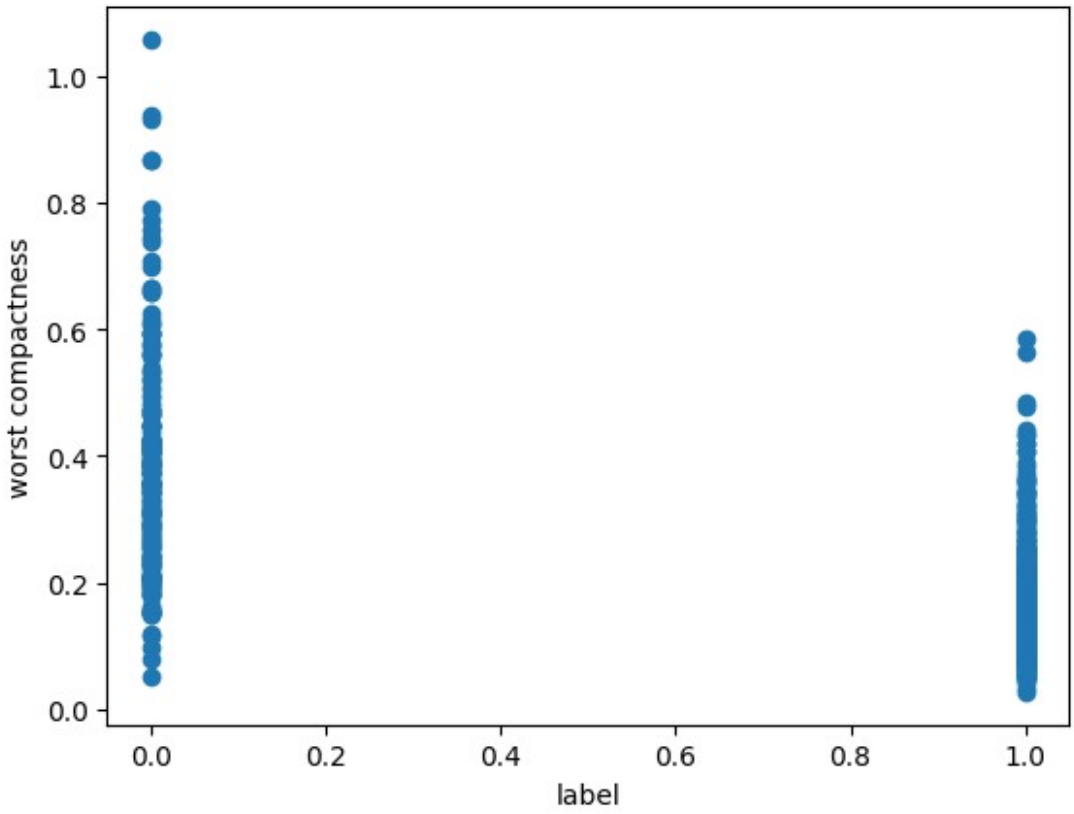

35
30
25
20
15
10
worst radius
0.0
0.2
0.4
0.6
0.8
1.0
label

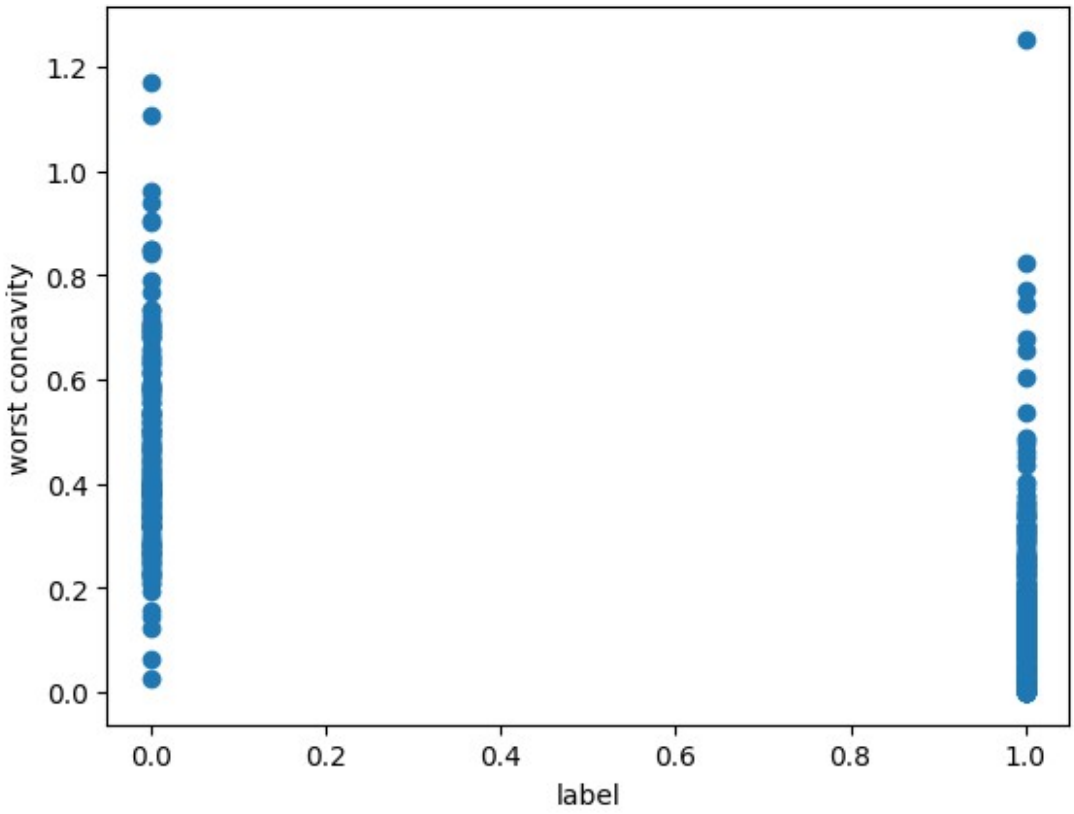

worst texture
50
45
40
35
30
25
20
15
0.0
0.2
0.4
0.6
0.8
1.0
label

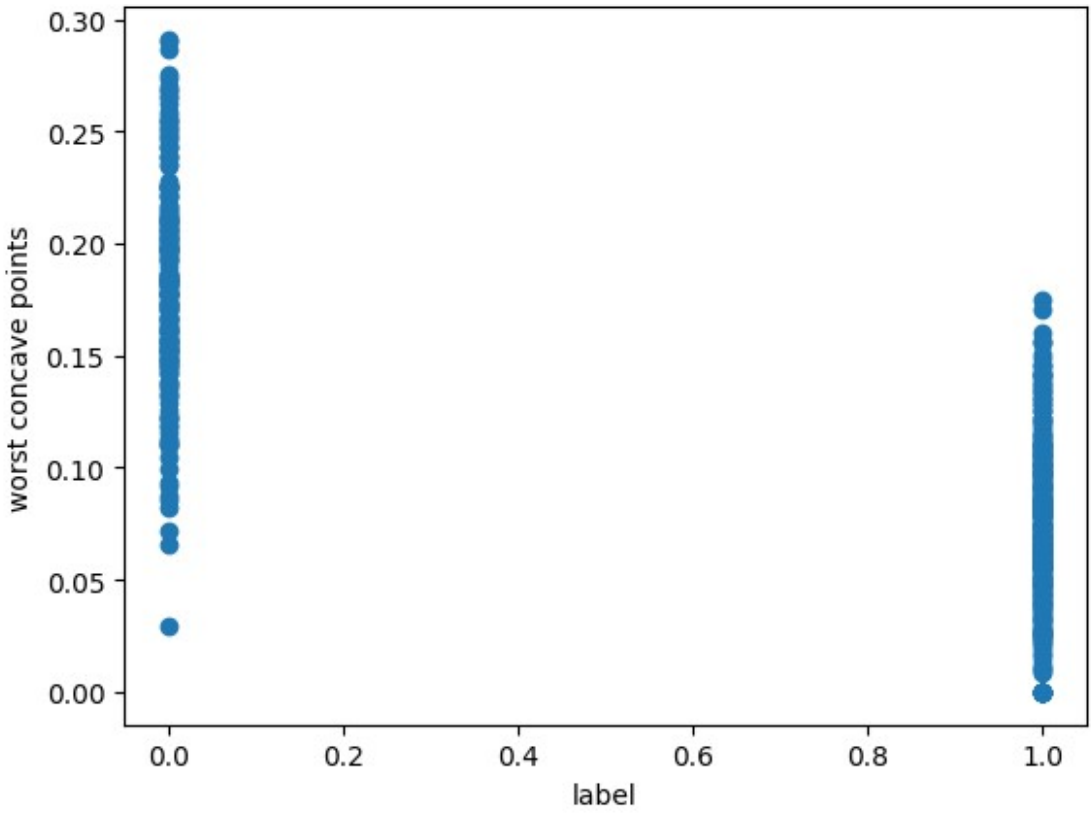

worst perimeter
250
225
200
175
150
125
100
75
50
0.0
0.2
0.4
0.6
0.8
1.0
label

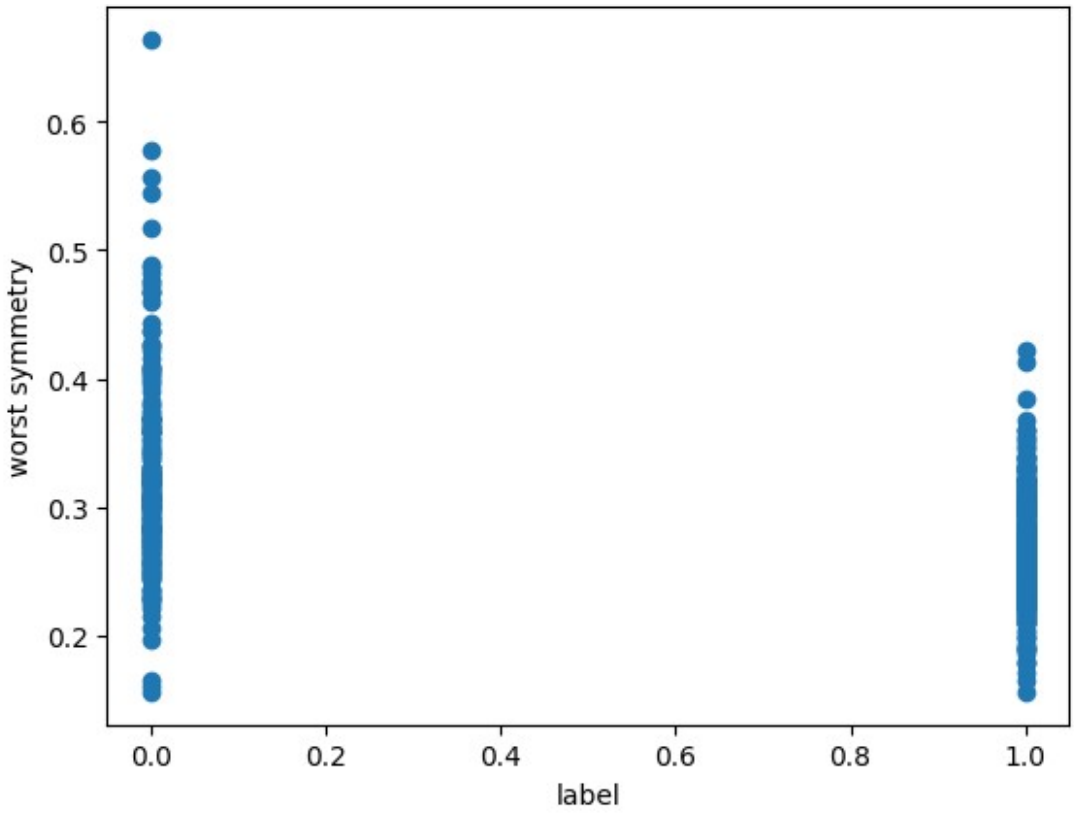

worst area
4000
3500
3000
2500
2000
1500
1000
500
0
0.0
0.2
0.4
0.6
0.8
1.0
label

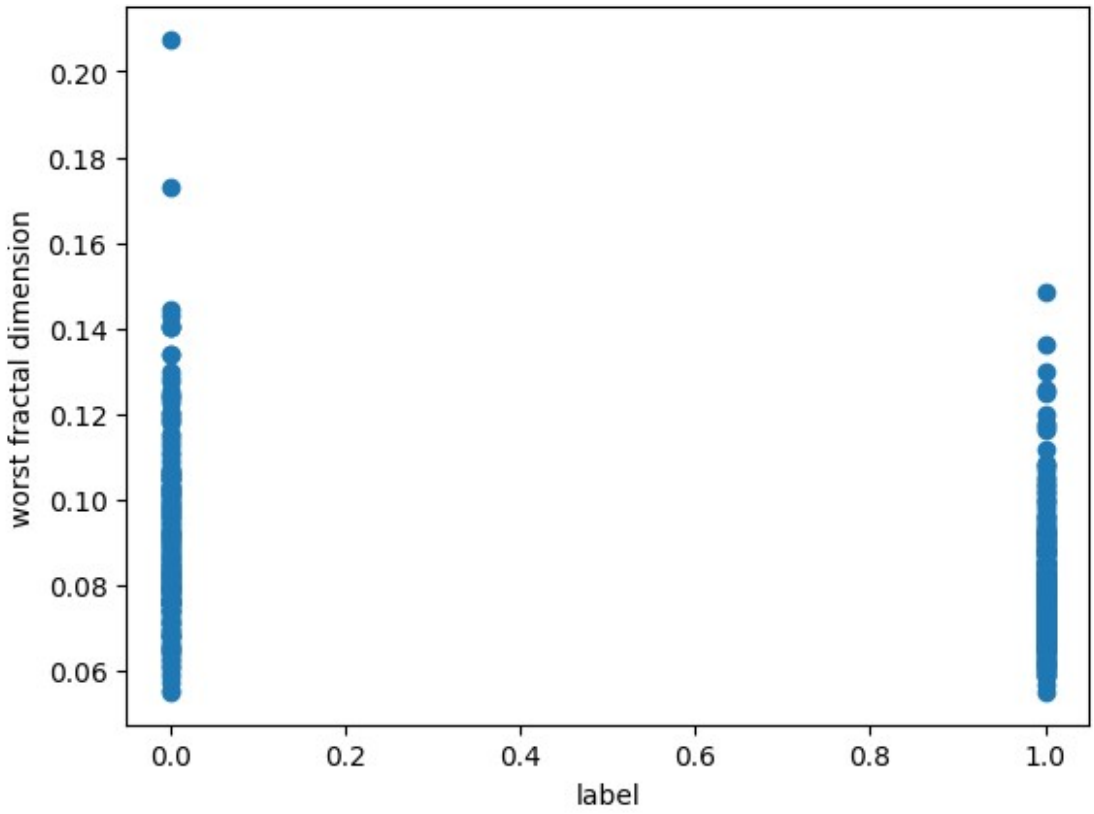

worst smoothness
0.22
0.20
0.18
0.16
0.14
0.12
0.10
0.08
0.0
0.2
0.4
0.6
0.8
1.0
label


worst compactness
label
0.0
0.2
0.4
0.6
0.8
1.0

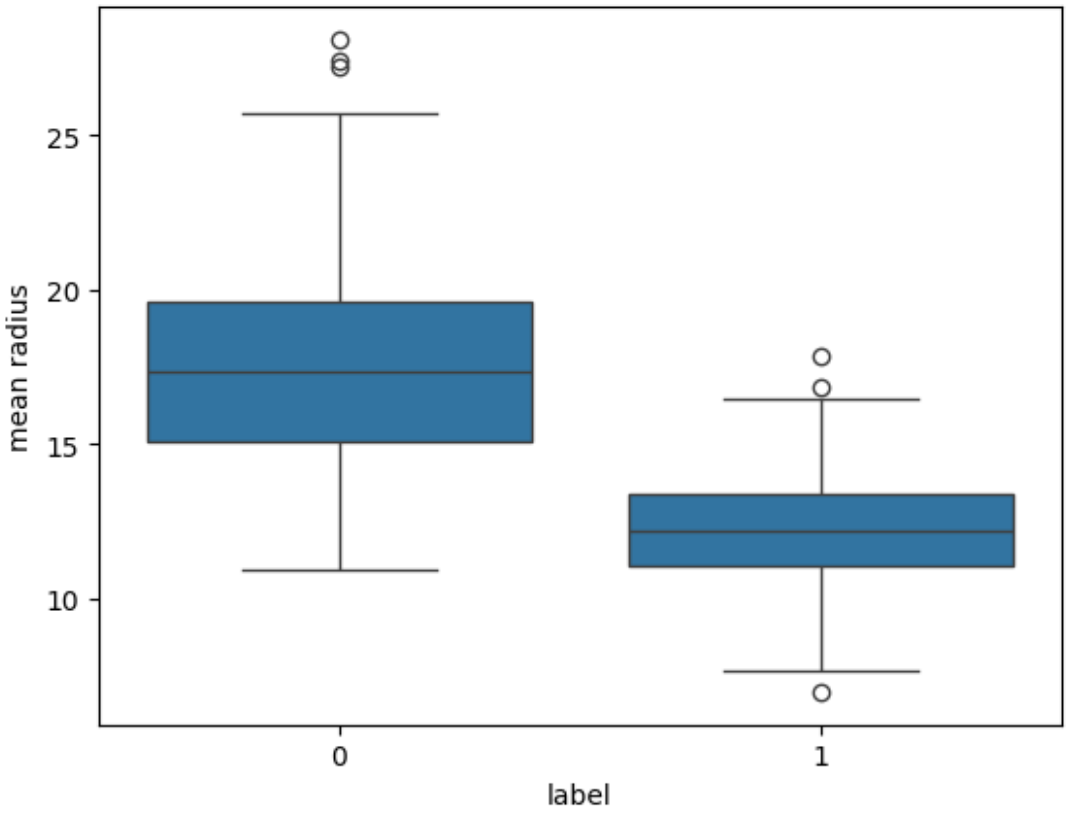

worst concavity
label
0.0
0.2
0.4
0.6
0.8
1.0
1.2

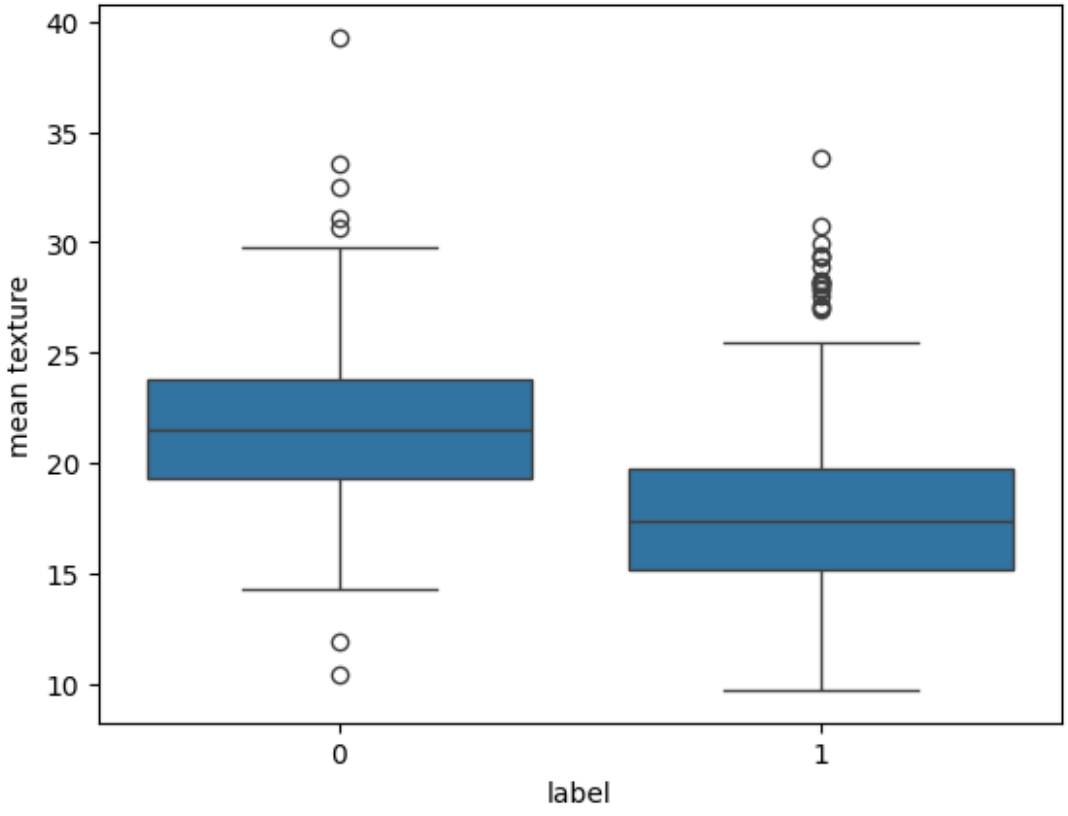

0.30
0.25
0.20
0.15
0.10
0.05
0.00
worst concave points
0.0
0.2
0.4
0.6
0.8
1.0
label

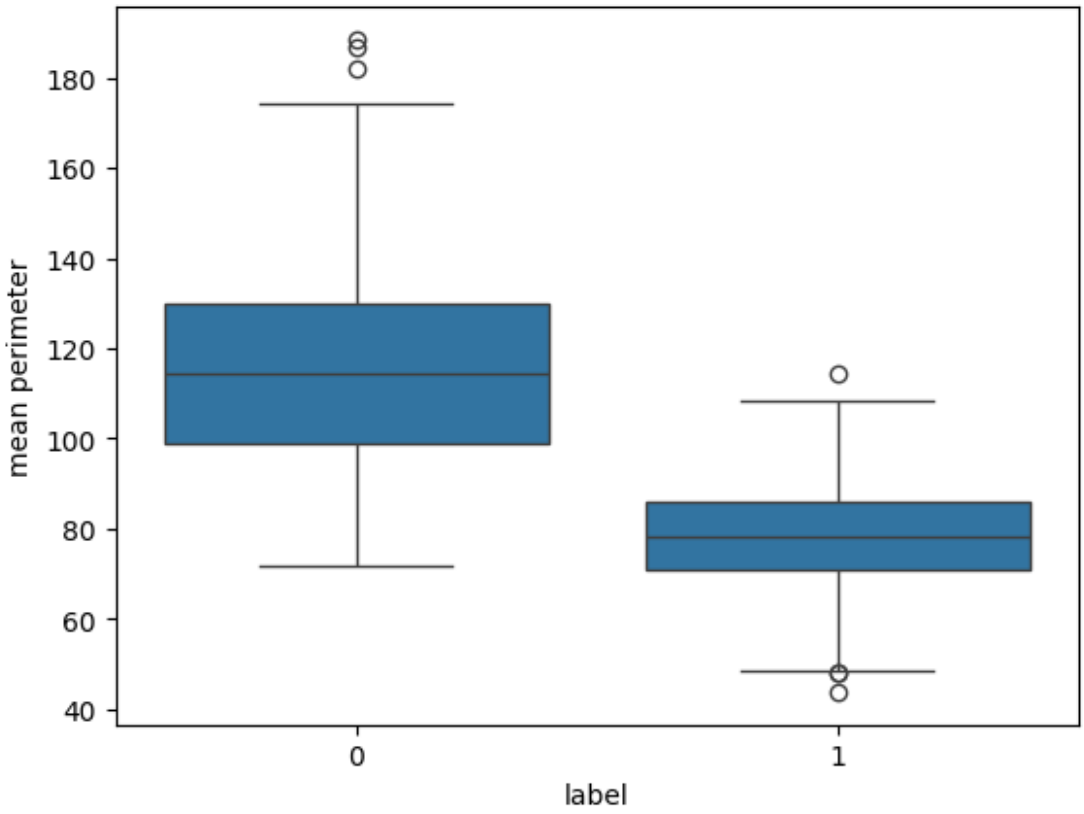

0.6
0.5
0.4
0.3
0.2
worst symmetry
0.0
0.2
0.4
0.6
0.8
1.0
label

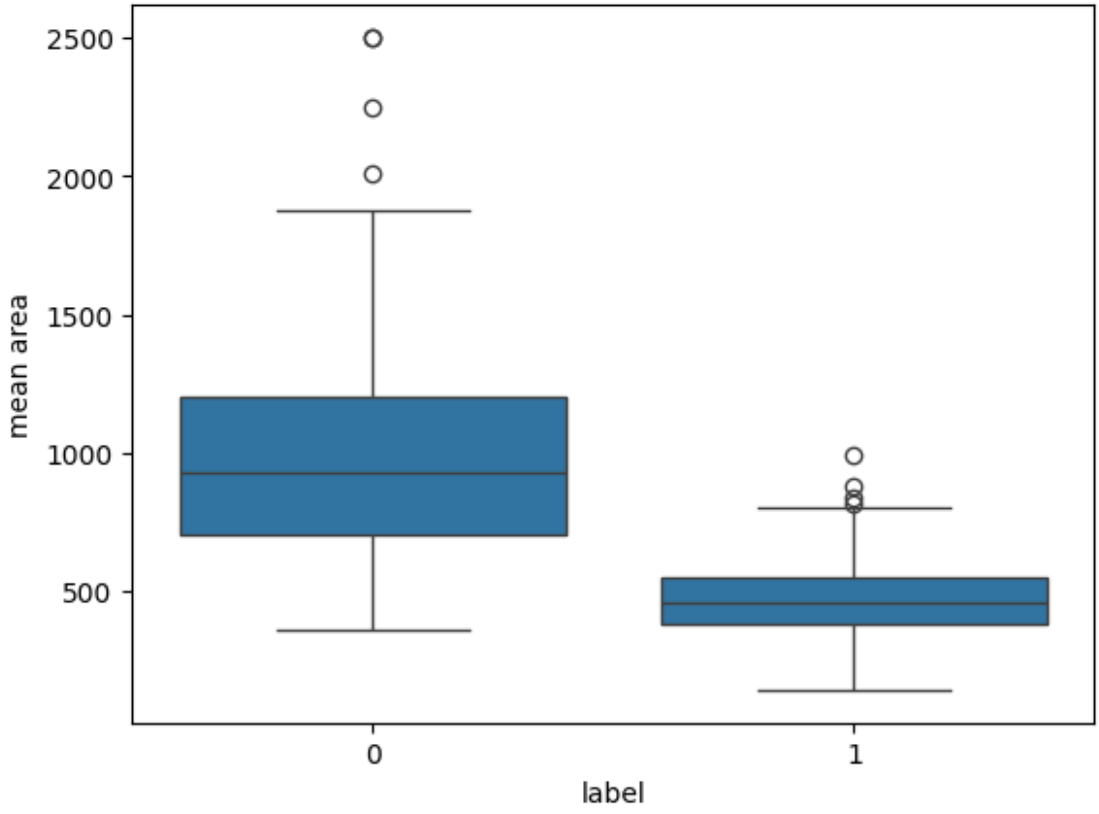


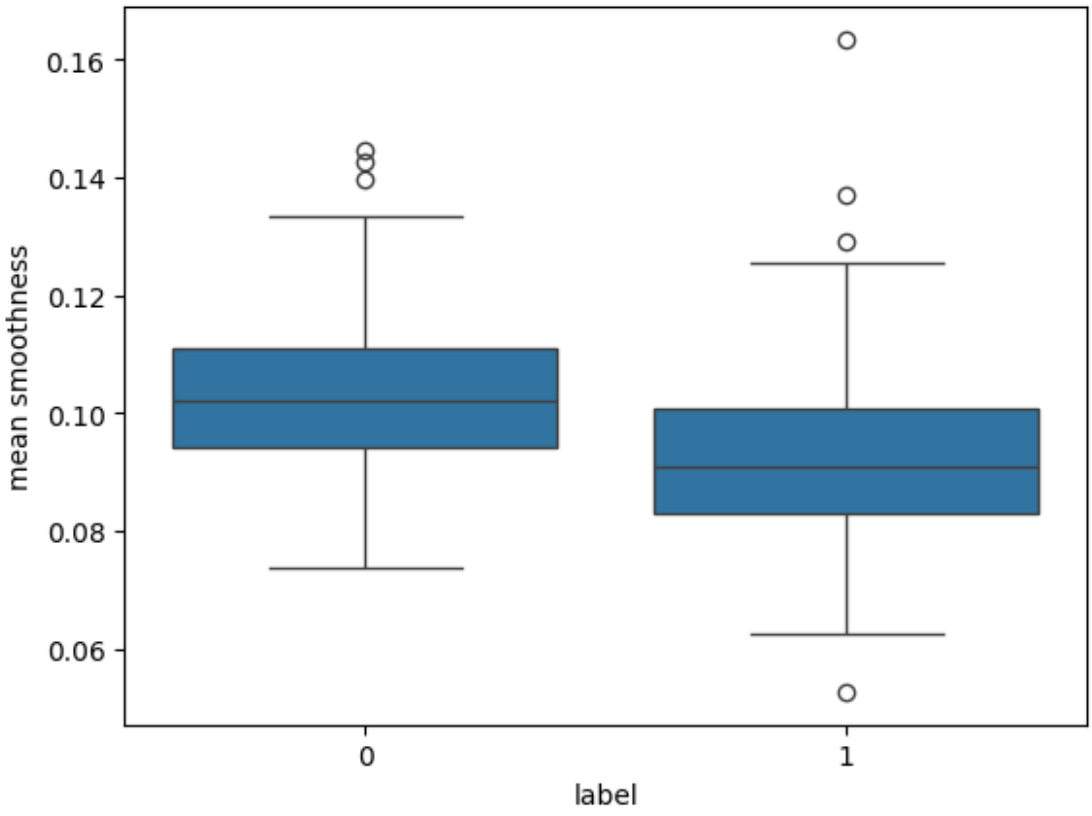

```
for i in data_frame.columns:
    if i != 'label':
        sns.boxplot(x='label', y=i, data=data_frame)
        plt.show()
```

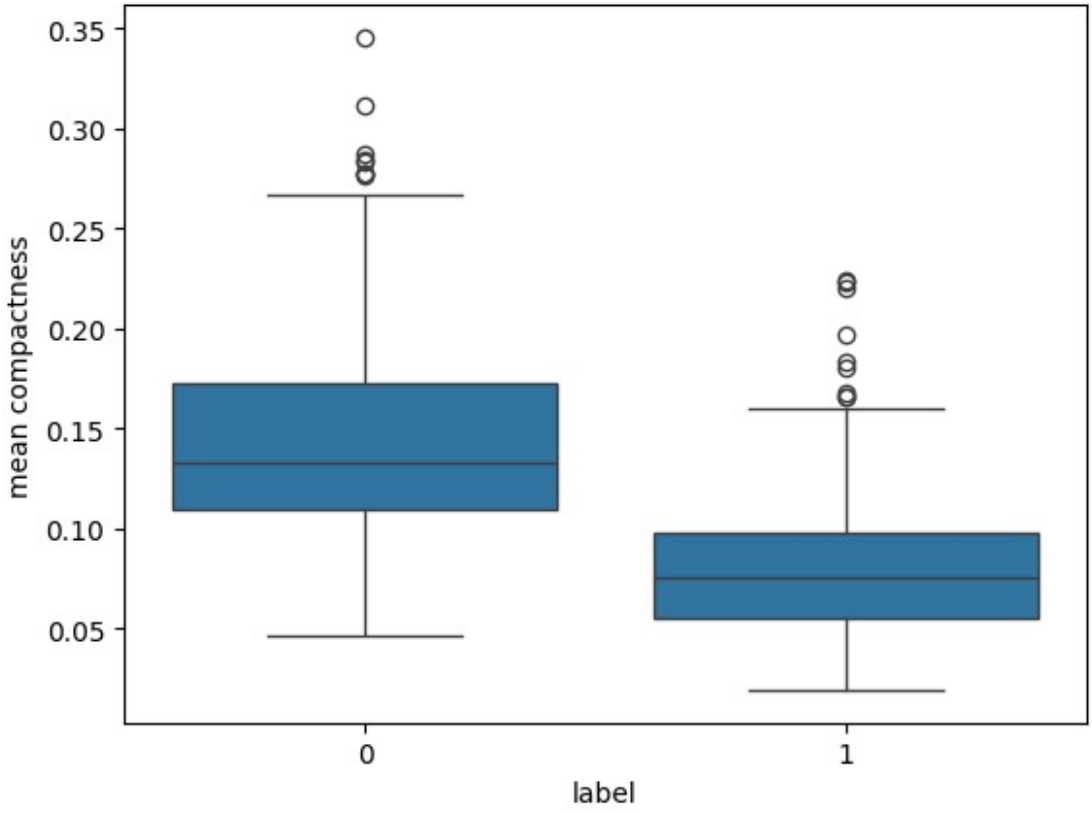

mean radius
25
20
15
10
0
1
label

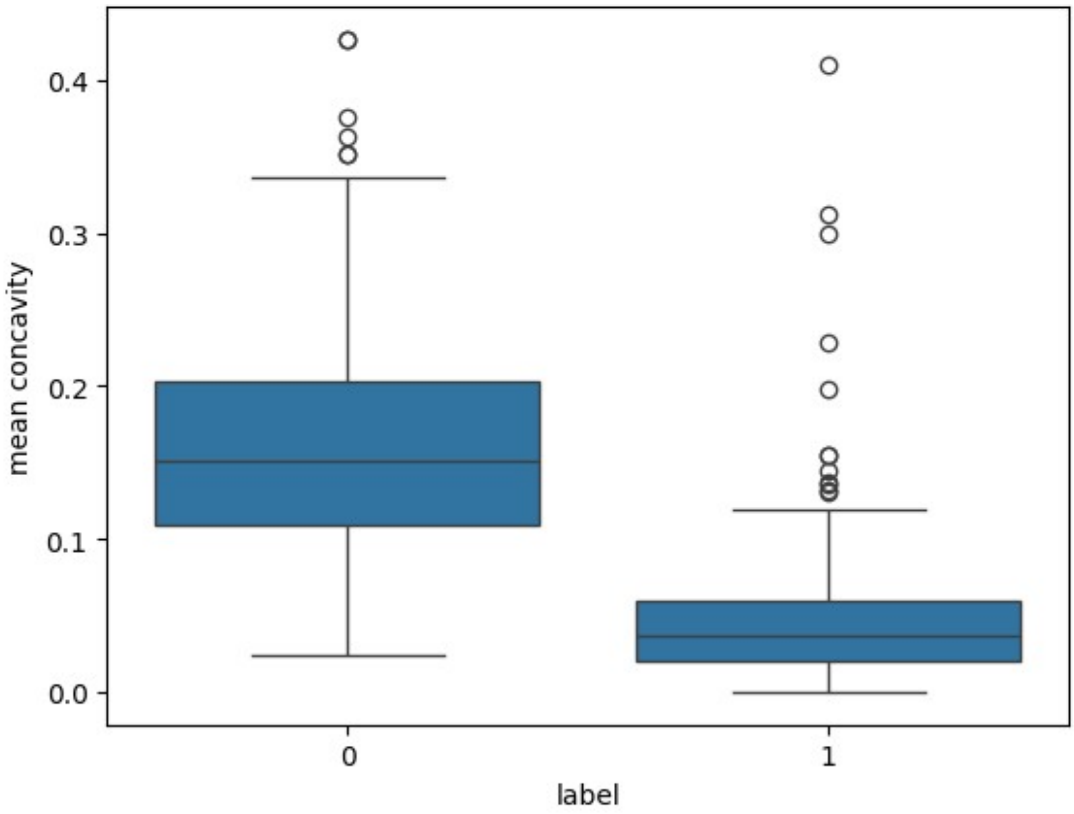

mean texture
40
35
30
25
20
15
10
0
1
label

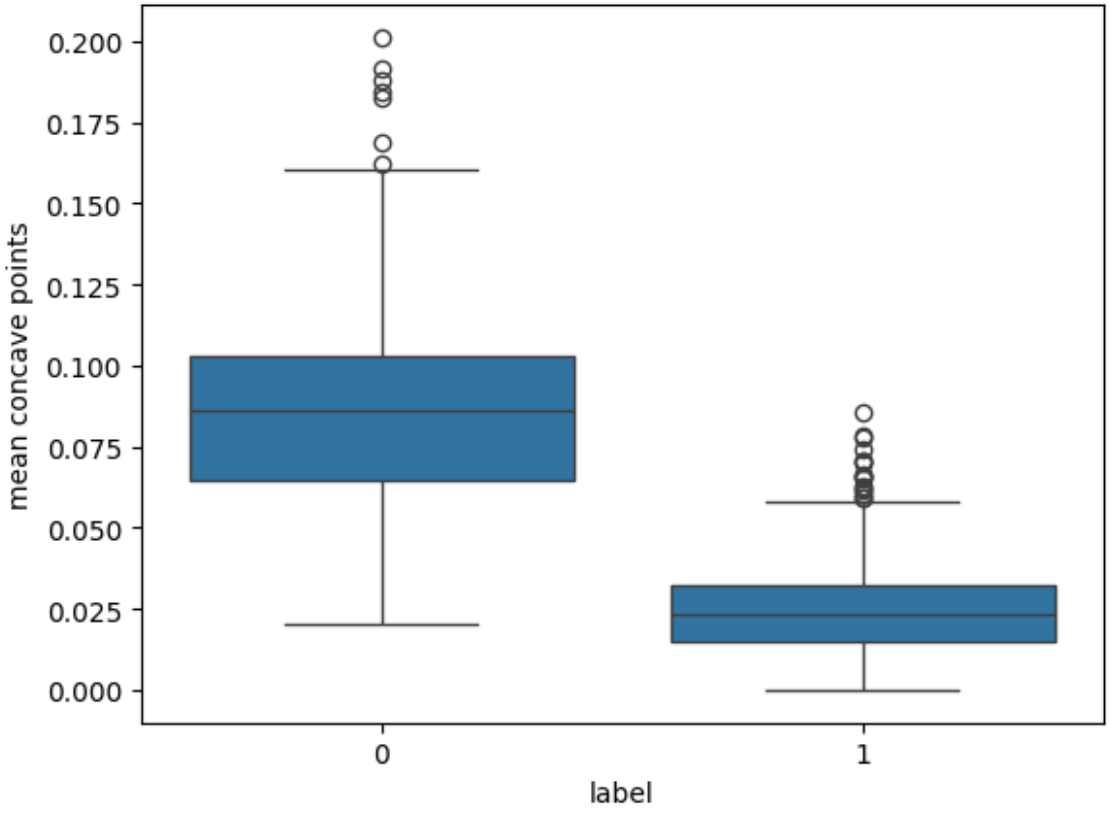

mean perimeter
180
160
140
120
100
80
60
40
0
1
label

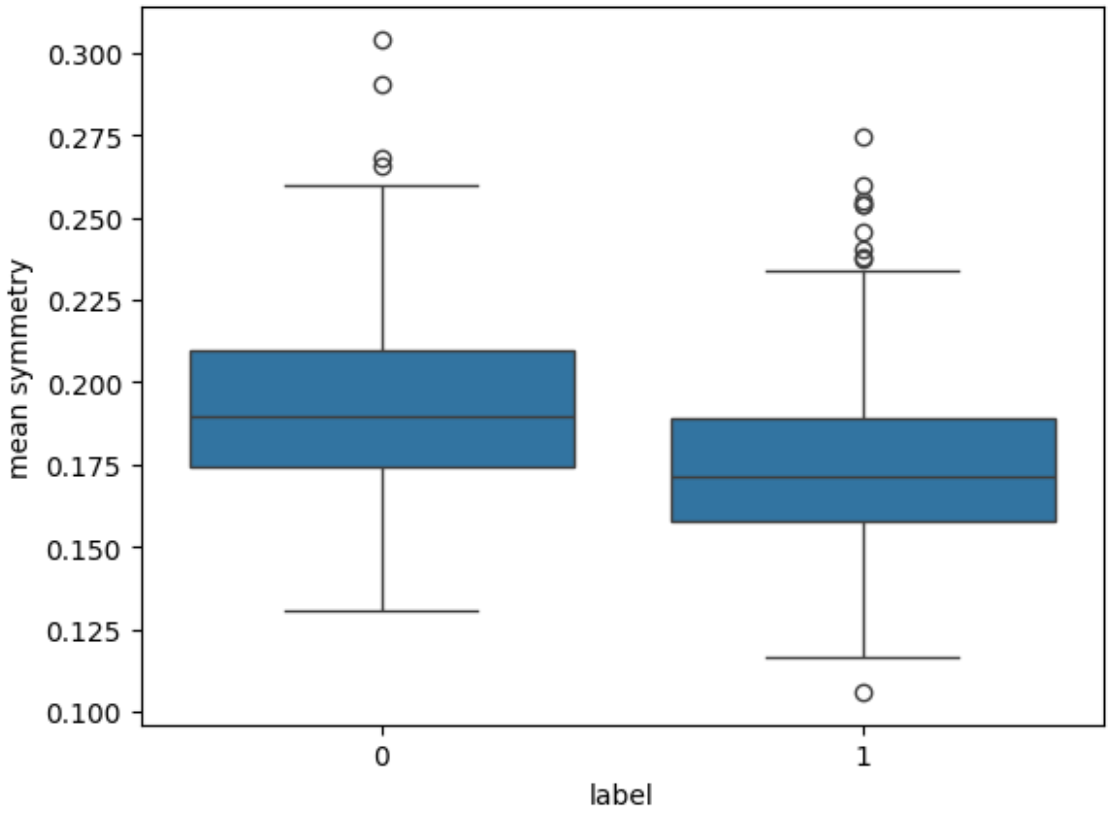

mean area
2500
2000
1500
1000
500
0
1
label

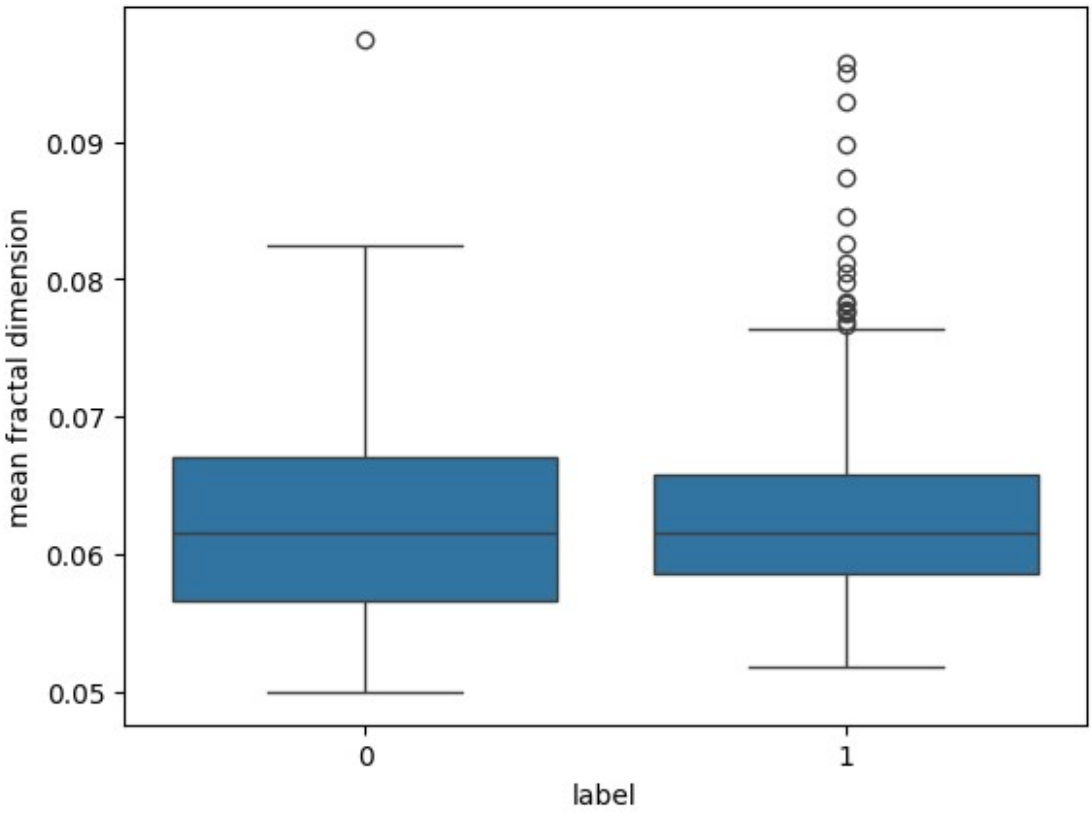

0.16
0.14
0.12
0.10
0.08
0.06
mean smoothness
0
1
label

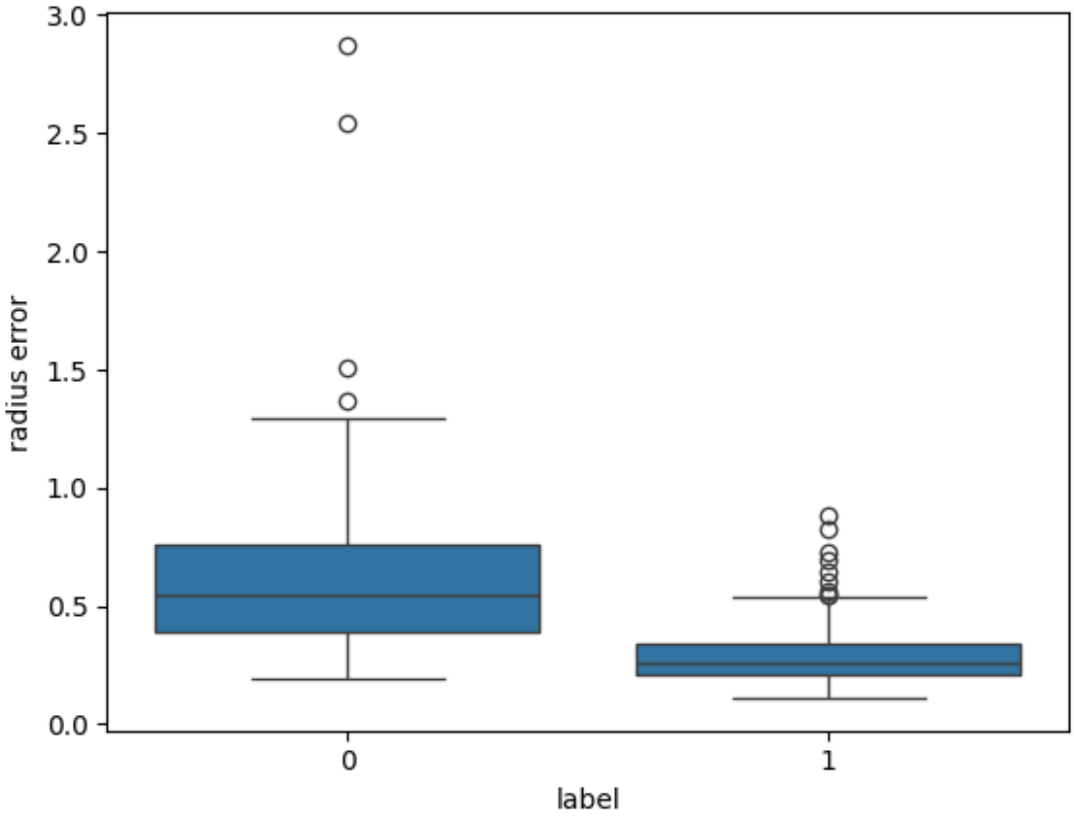

0.35
0.30
0.25
0.20
0.15
0.10
0.05
mean compactness
0
1
label

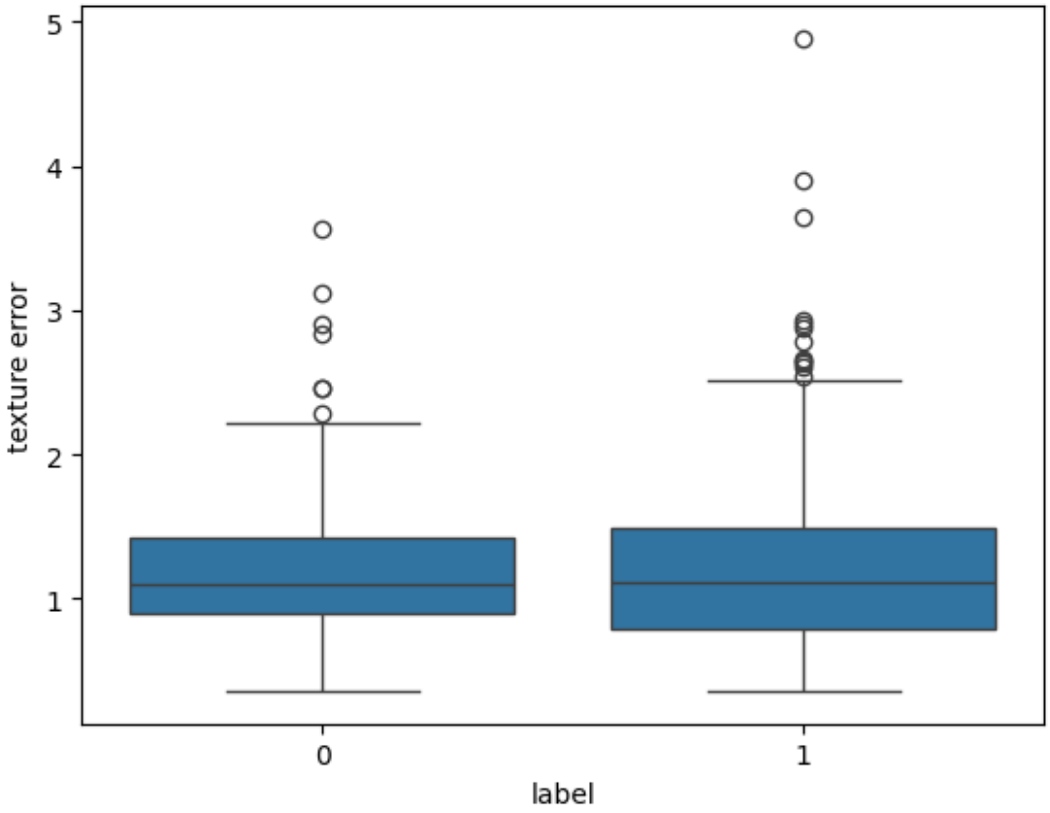

mean concavity
0.4
0.3
0.2
0.1
0.0
0
1
label

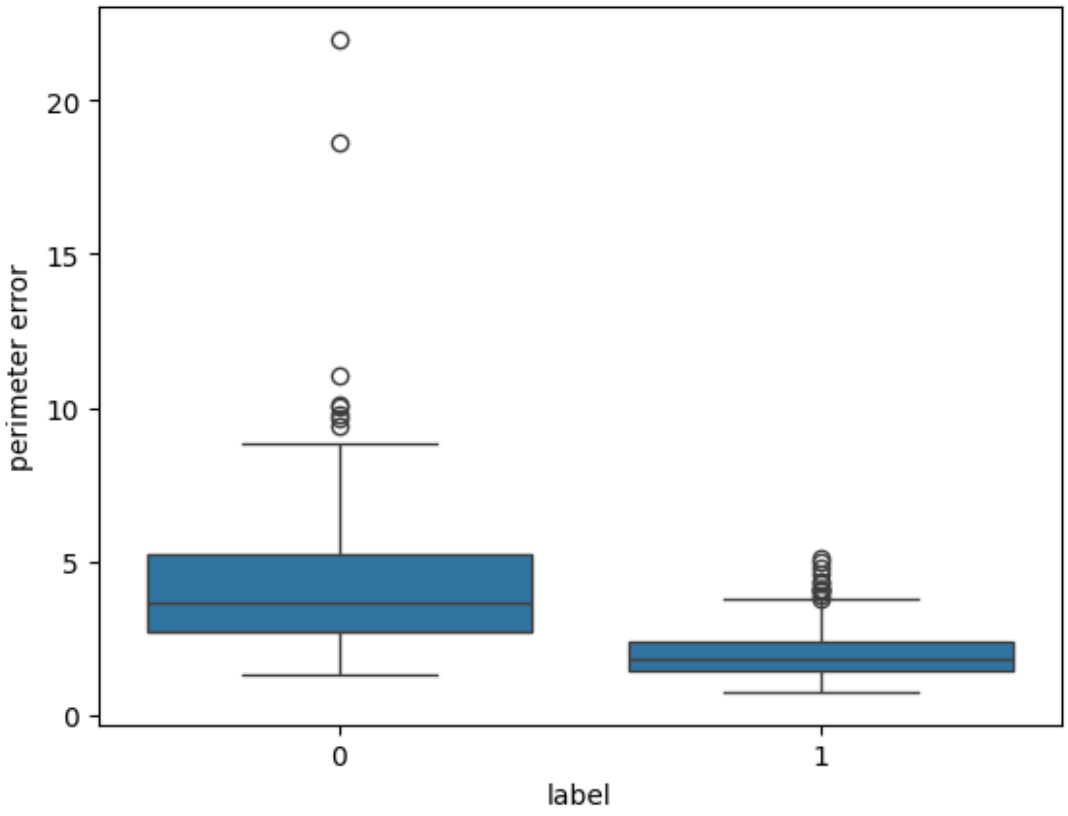

mean concave points
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
1
label

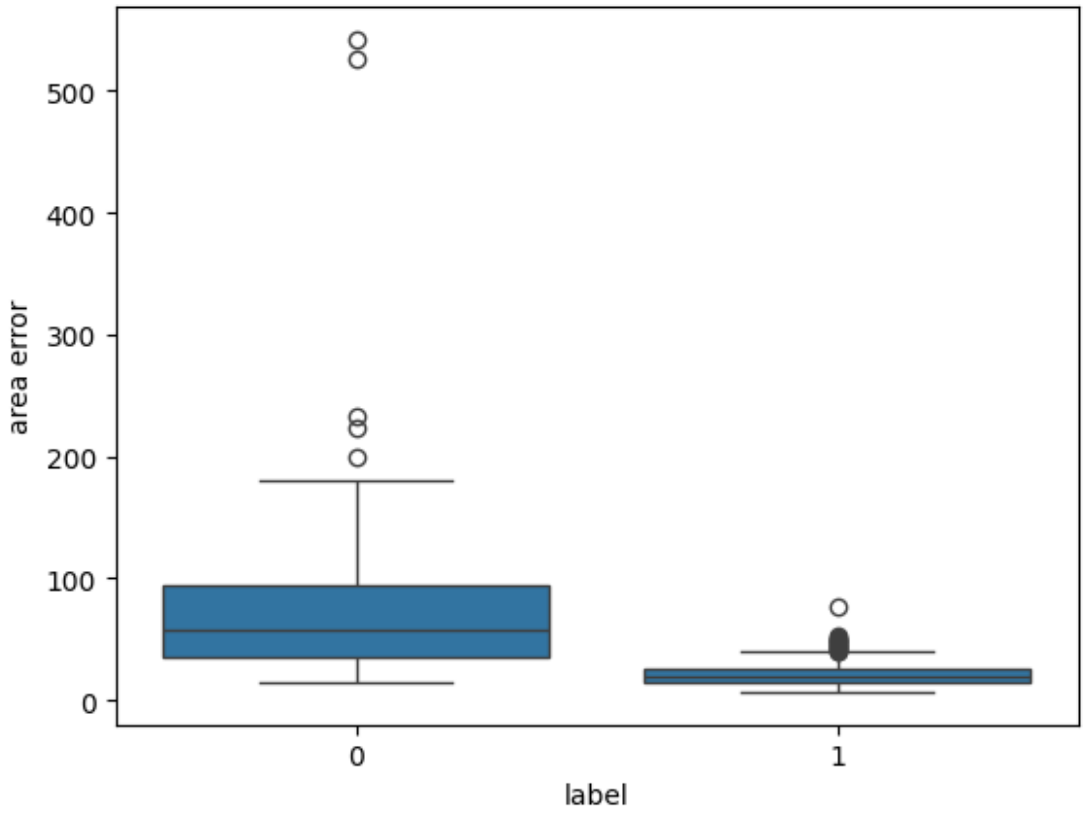

0.300
0.275
0.250
0.225
0.200
0.175
0.150
0.125
0.100
mean symmetry
0
1
label

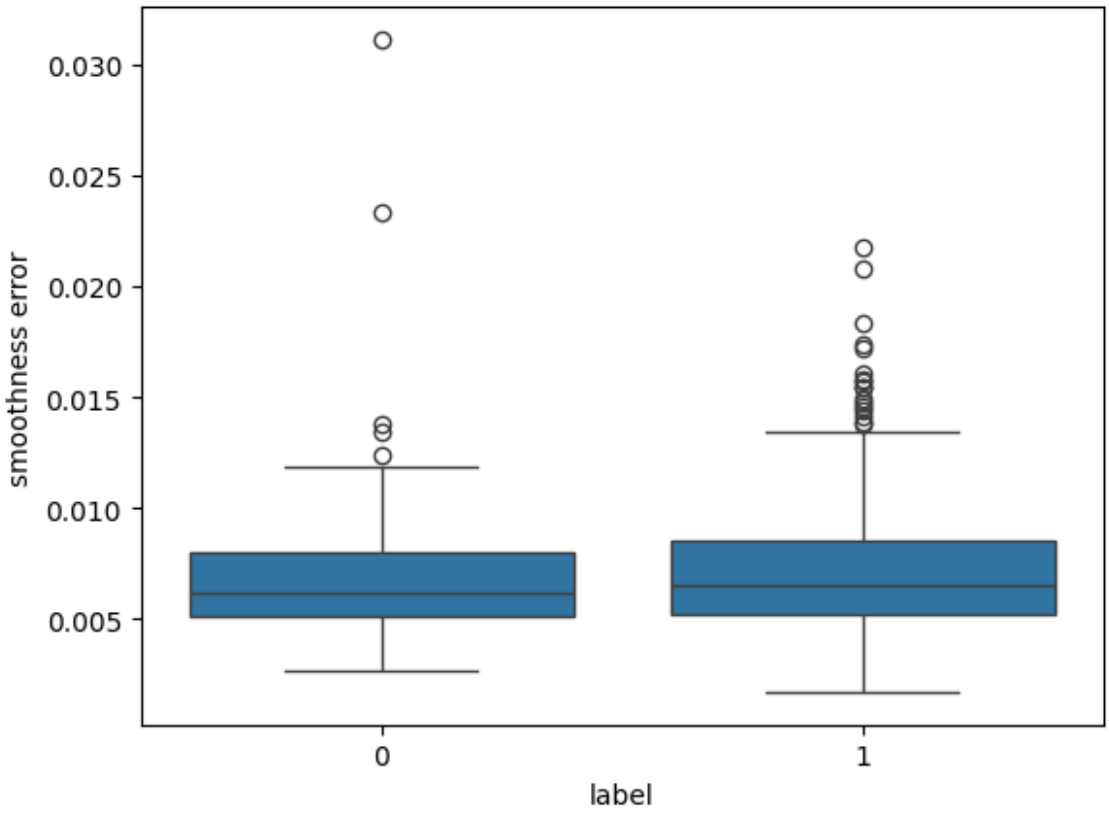

0.09
0.08
0.07
0.06
0.05
mean fractal dimension
0
1
label

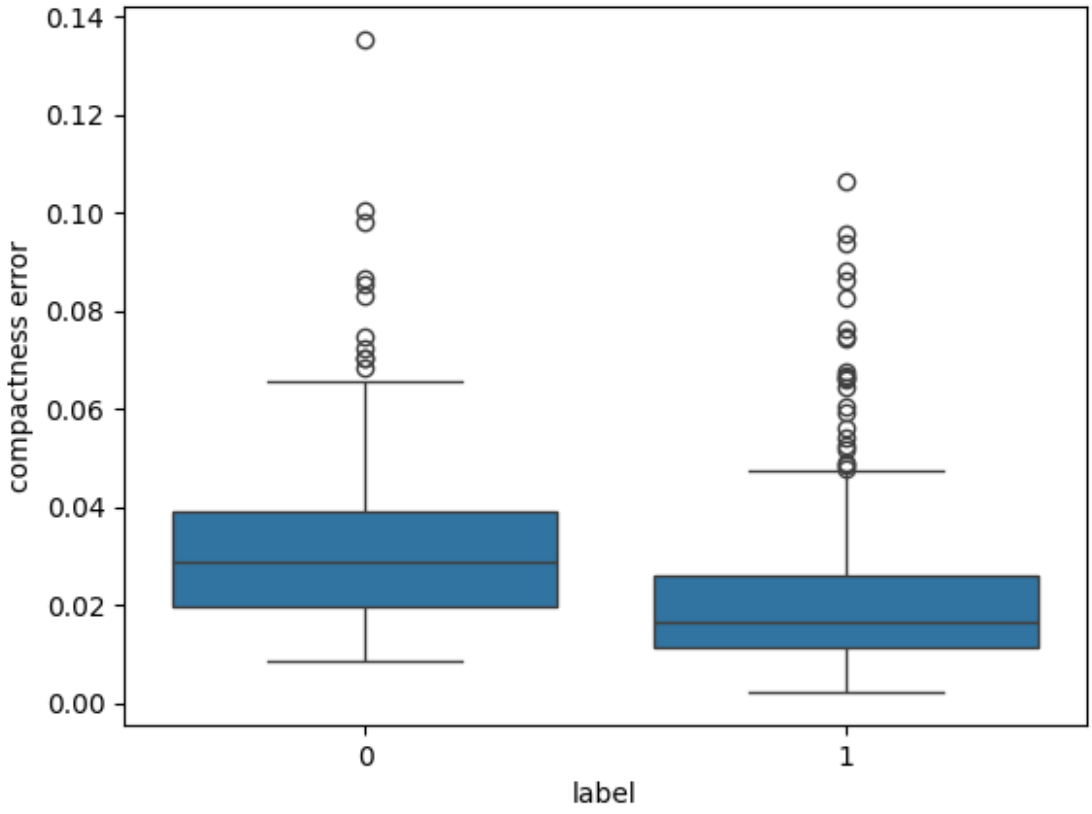

3.0
2.5
2.0
1.5
1.0
0.5
0.0
radius error
0
1
label

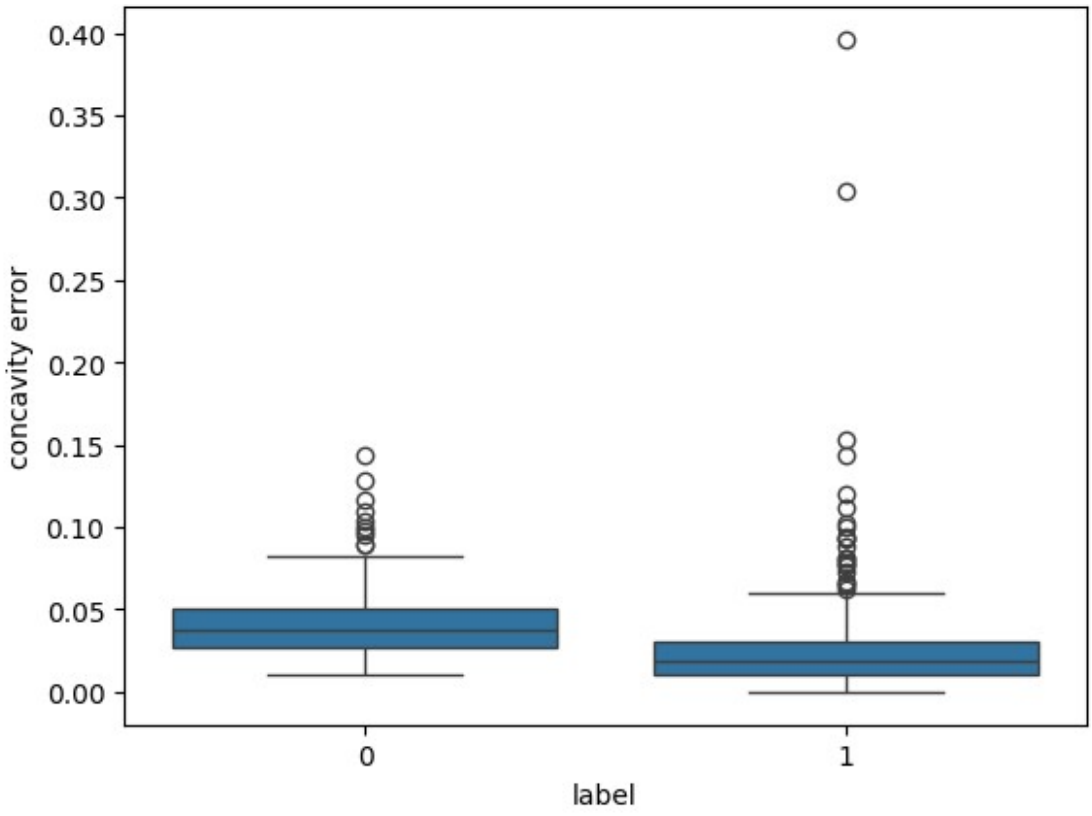

5
4
3
2
1
texture error
0
1
label

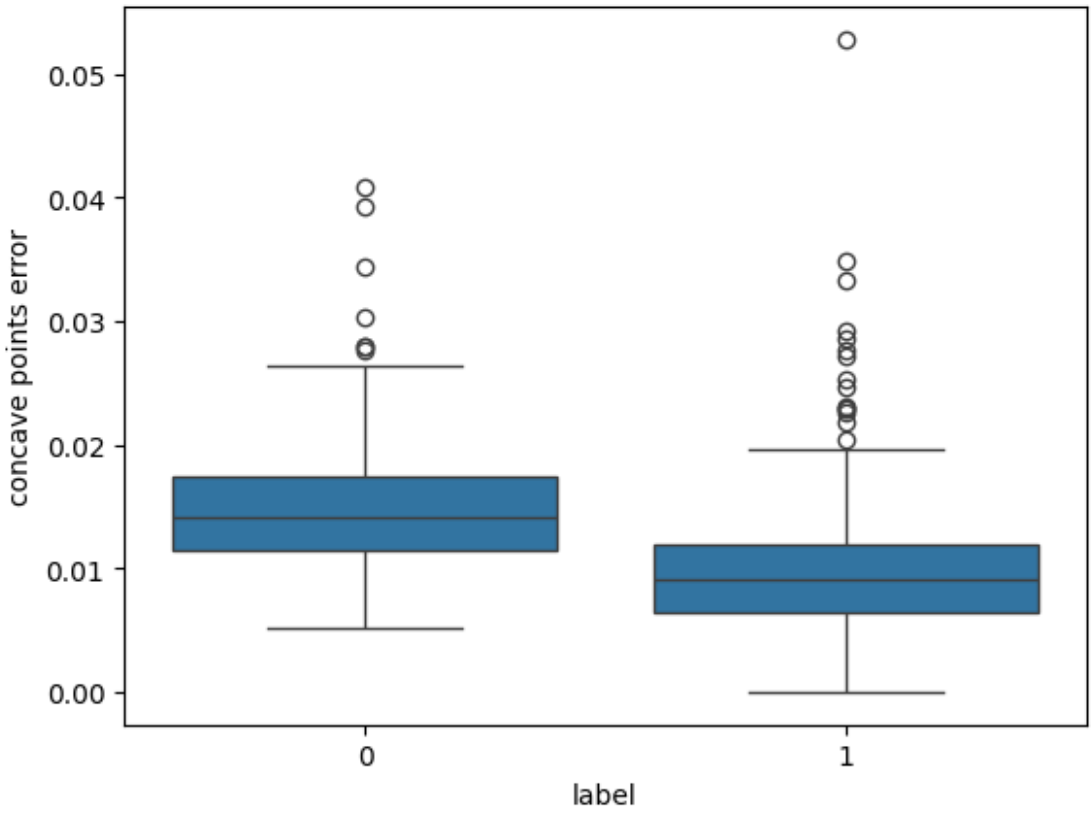

perimeter error
0
5
10
15
20
0
1
label

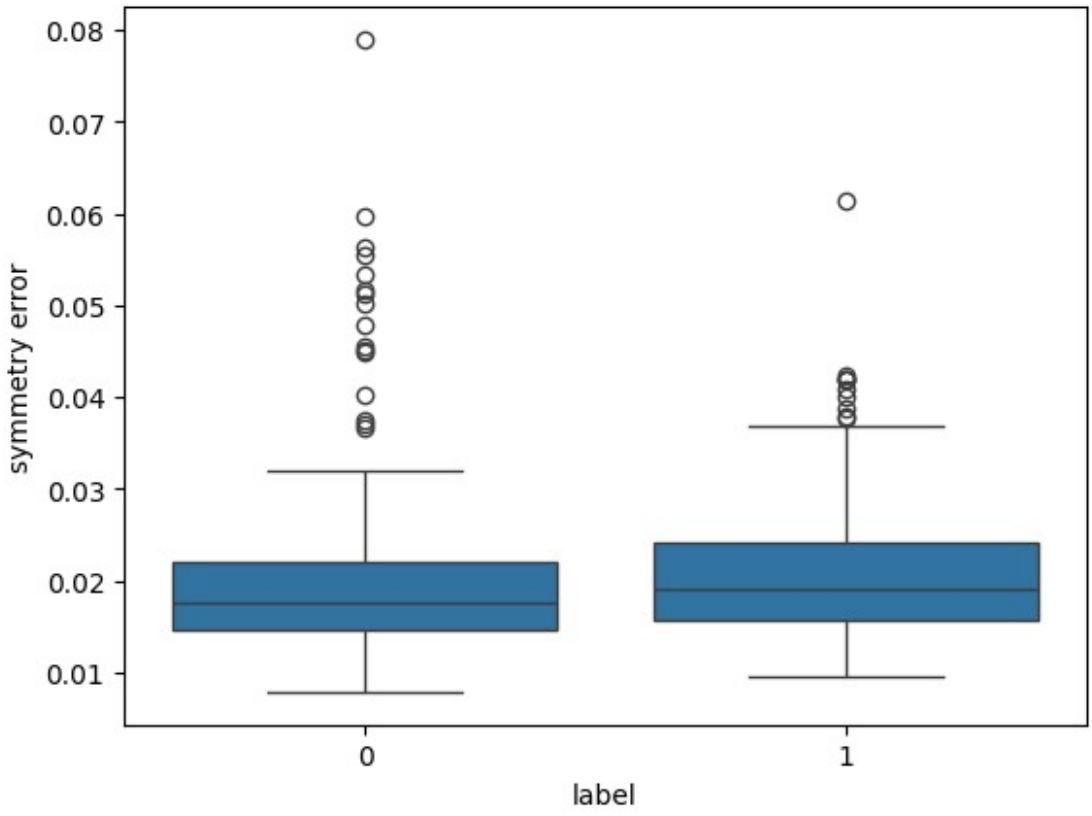

area error
0
100
200
300
400
500
0
1
label

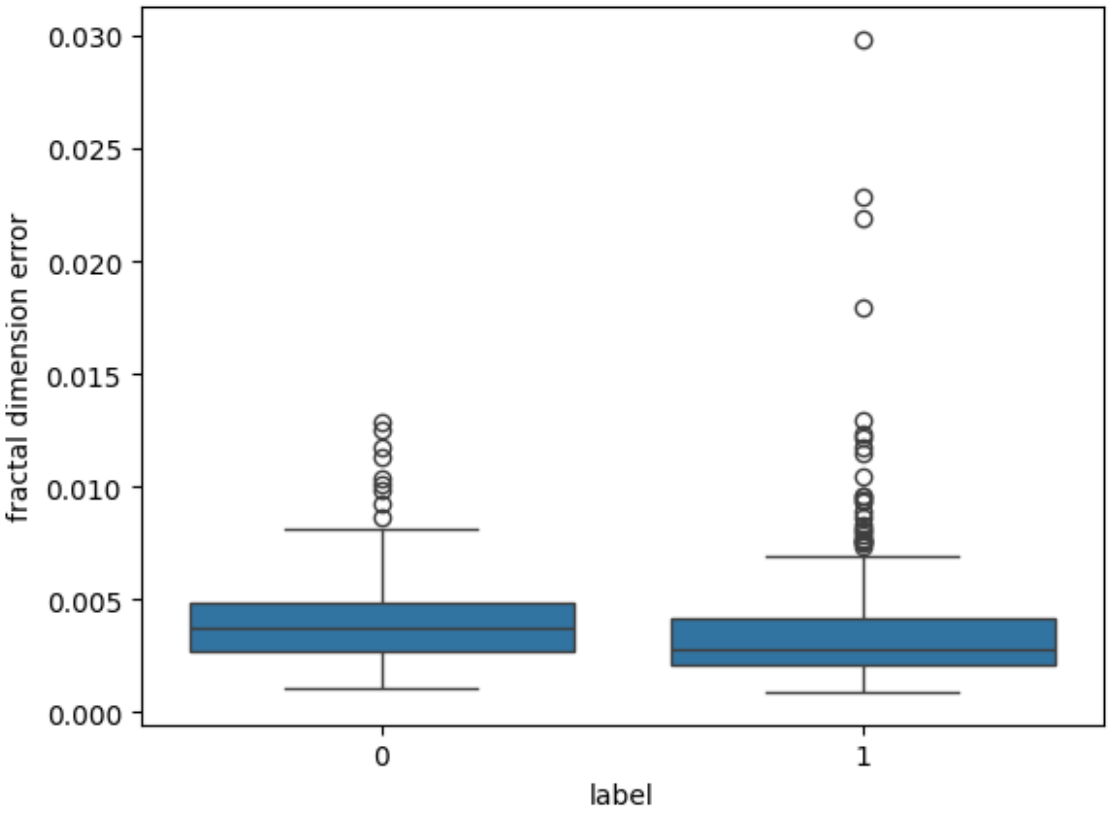

0.030
0.025
0.020
0.015
0.010
0.005
smoothness error
0
1
label

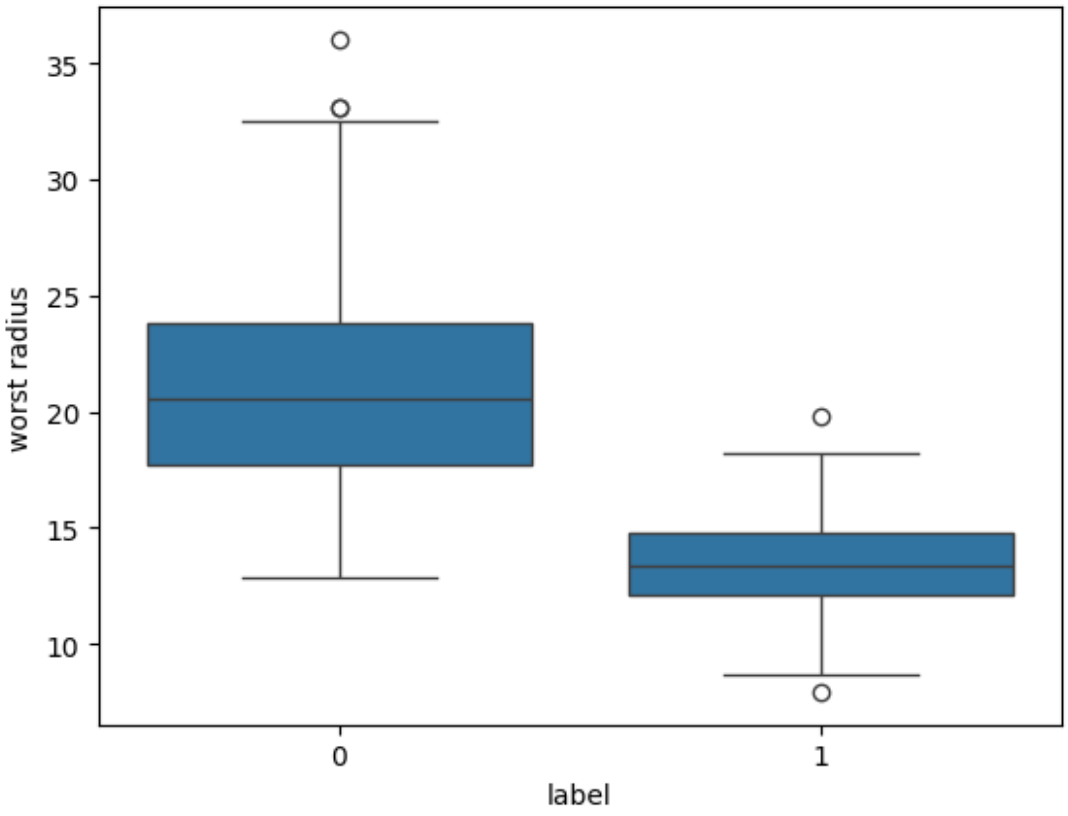

0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
compactness error
0
1
label

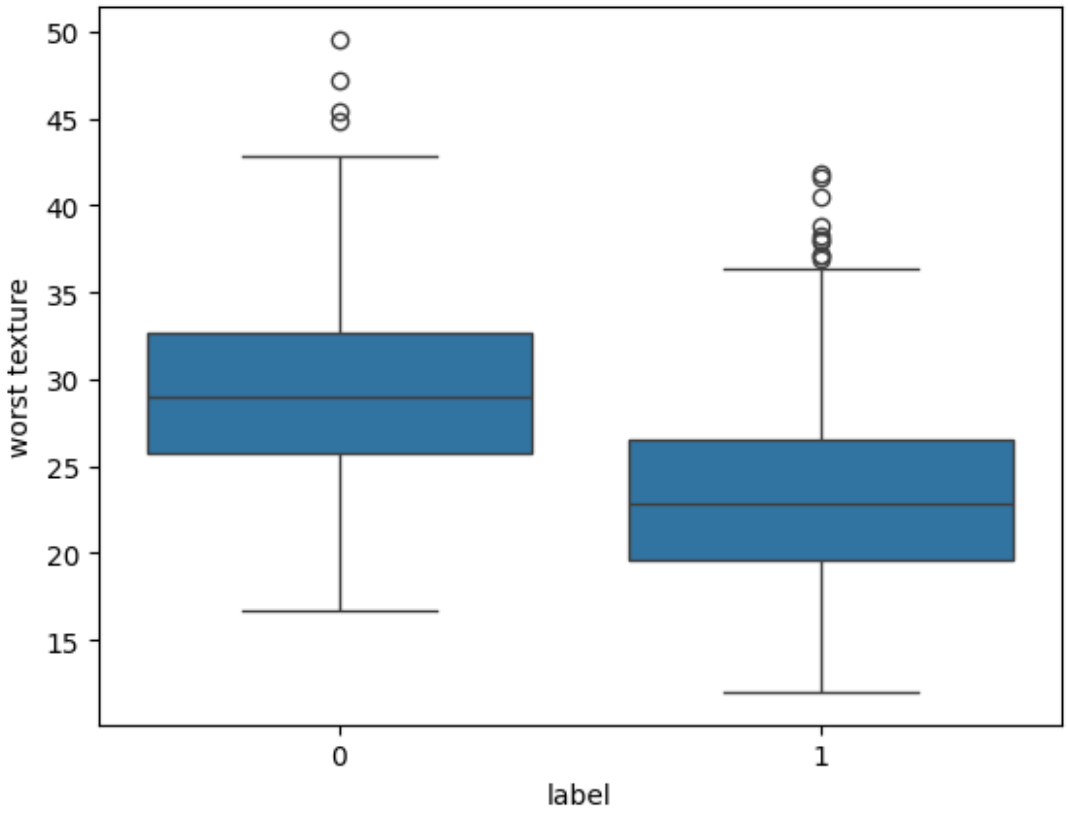

0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
concavity error
0
1
label

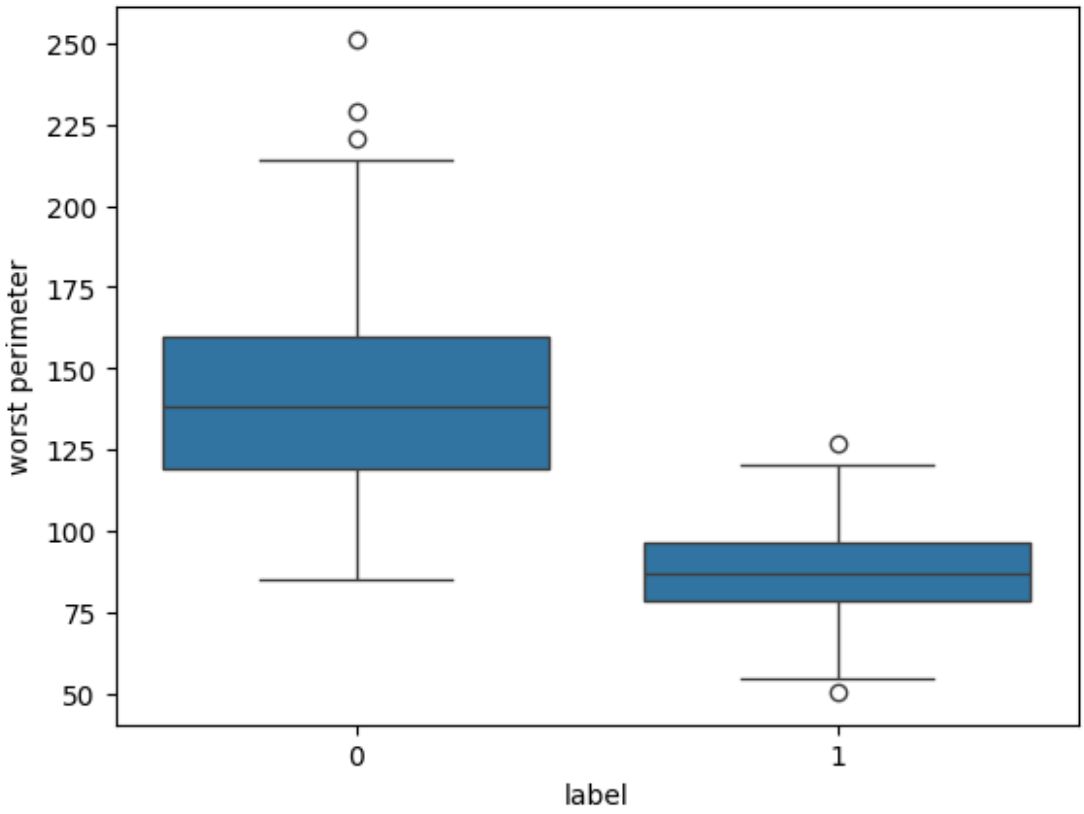

0.05
0.04
0.03
0.02
0.01
0.00
concave points error
0
1
label

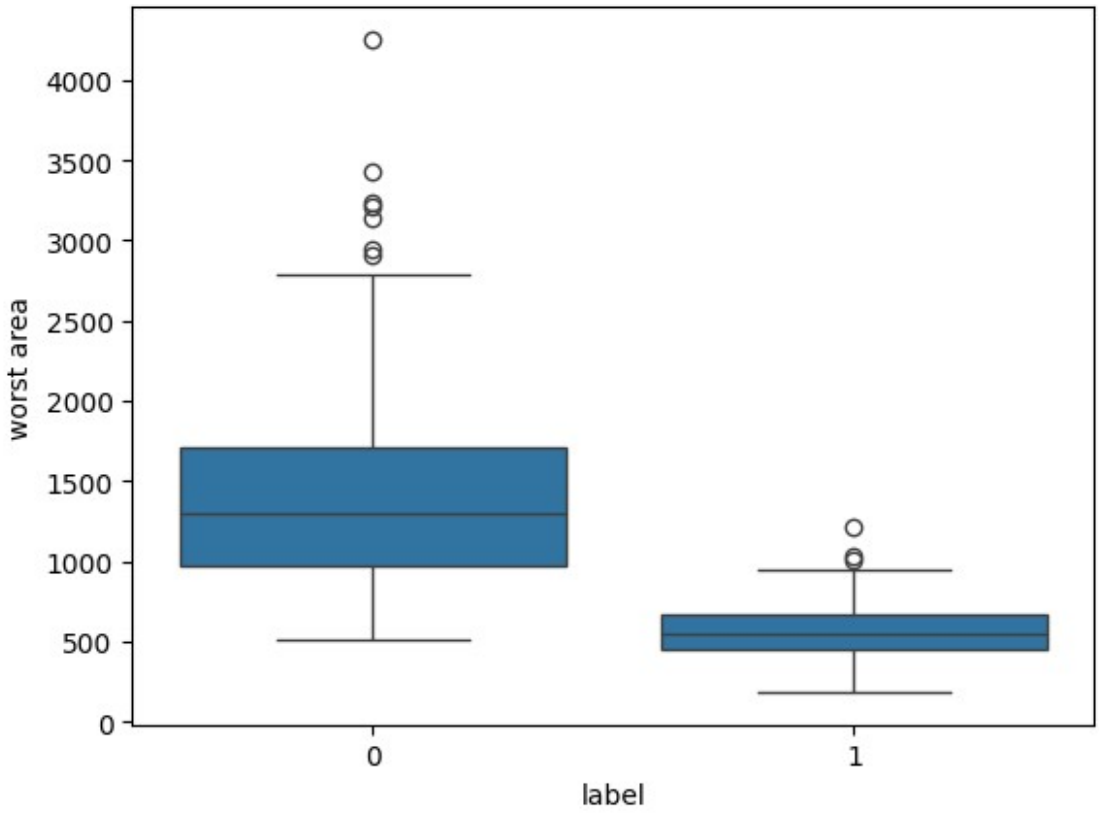

0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
symmetry error
0
1
label

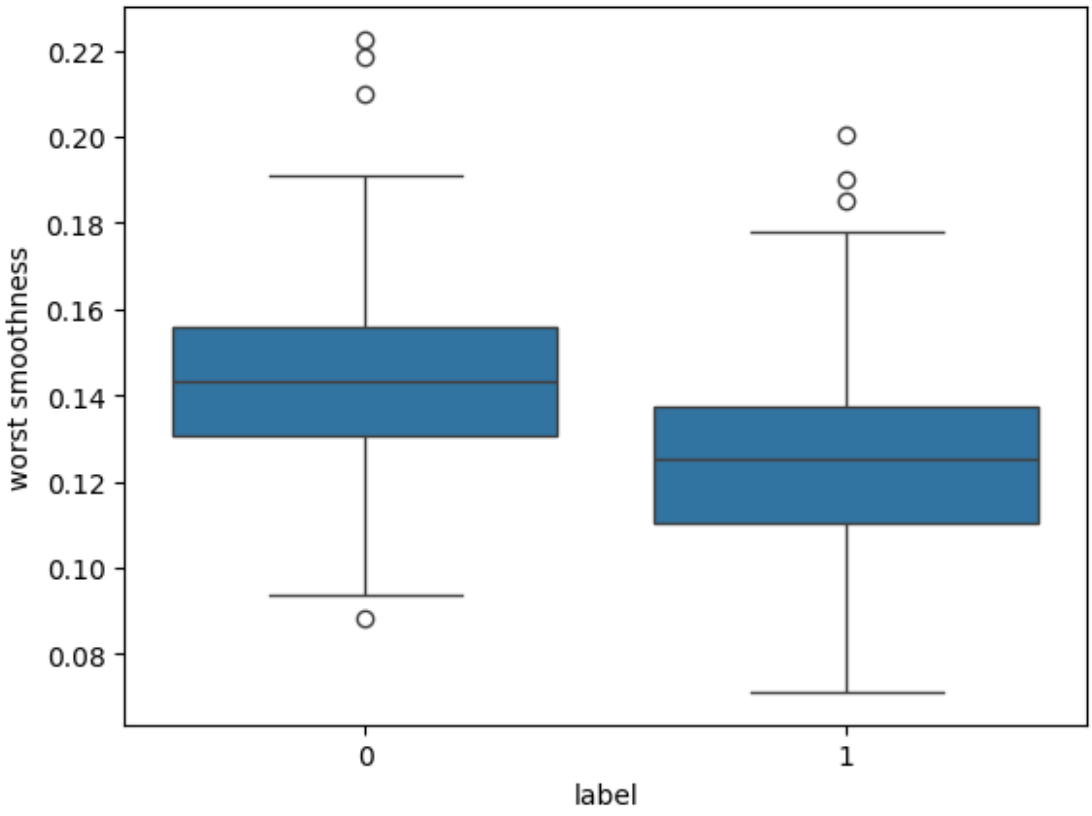

0.030
0.025
0.020
0.015
0.010
0.005
0.000
fractal dimension error
0
1
label

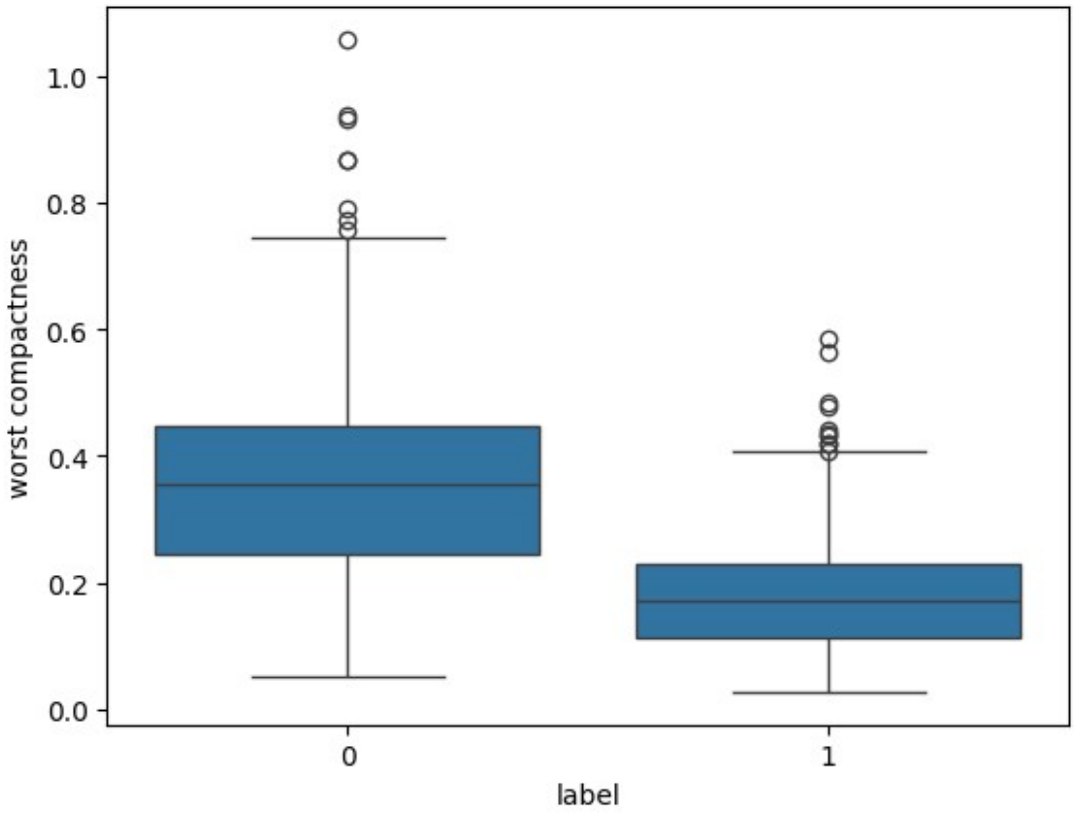

worst radius
35
30
25
20
15
10
0
1
label

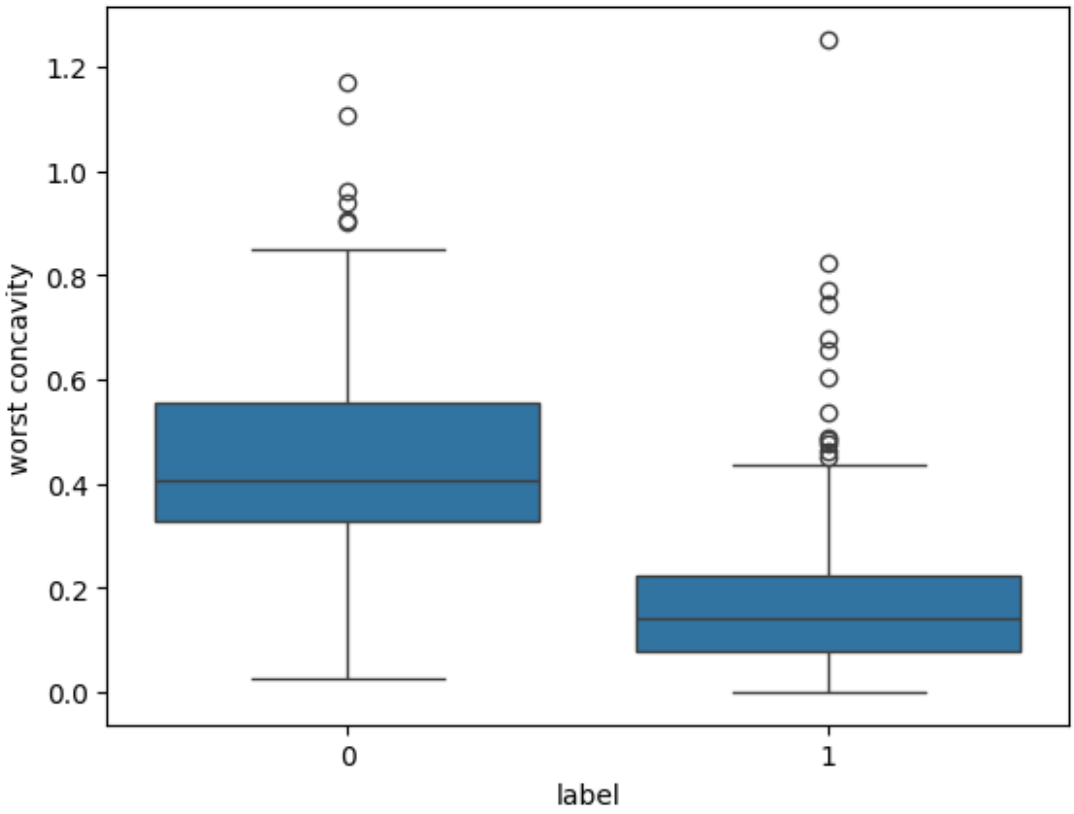

worst texture
50
45
40
35
30
25
20
15
0
1
label

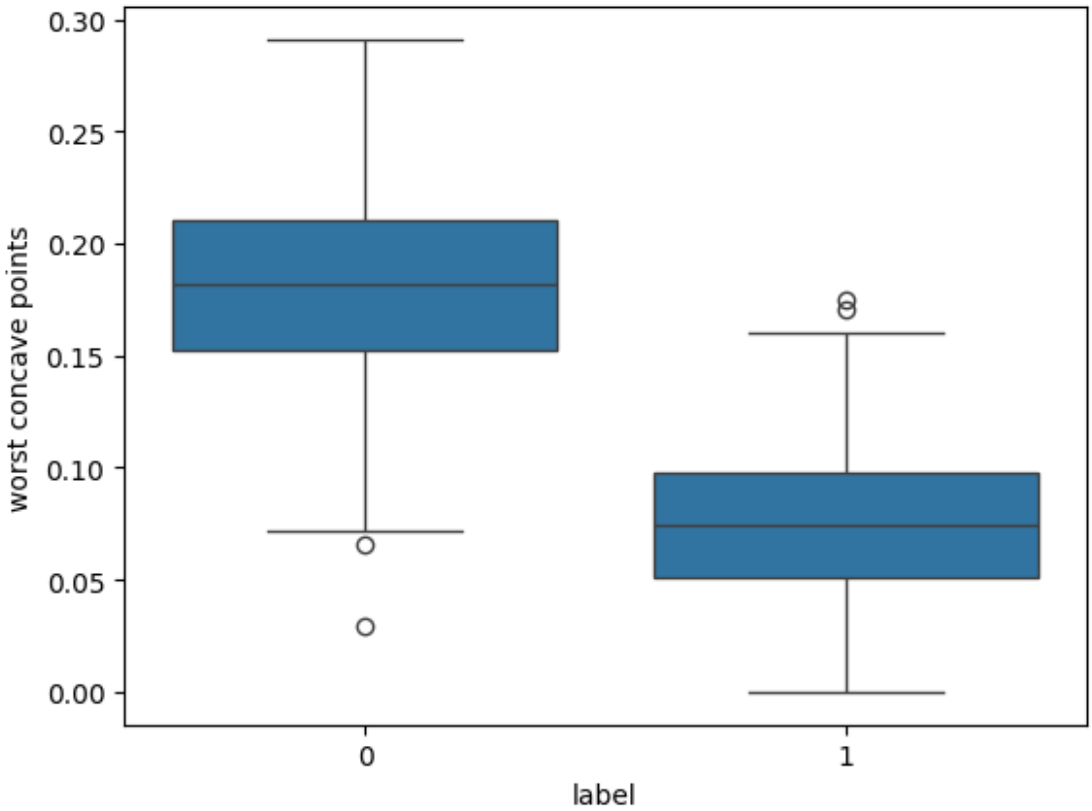

worst perimeter
250
225
200
175
150
125
100
75
50
0
1
label

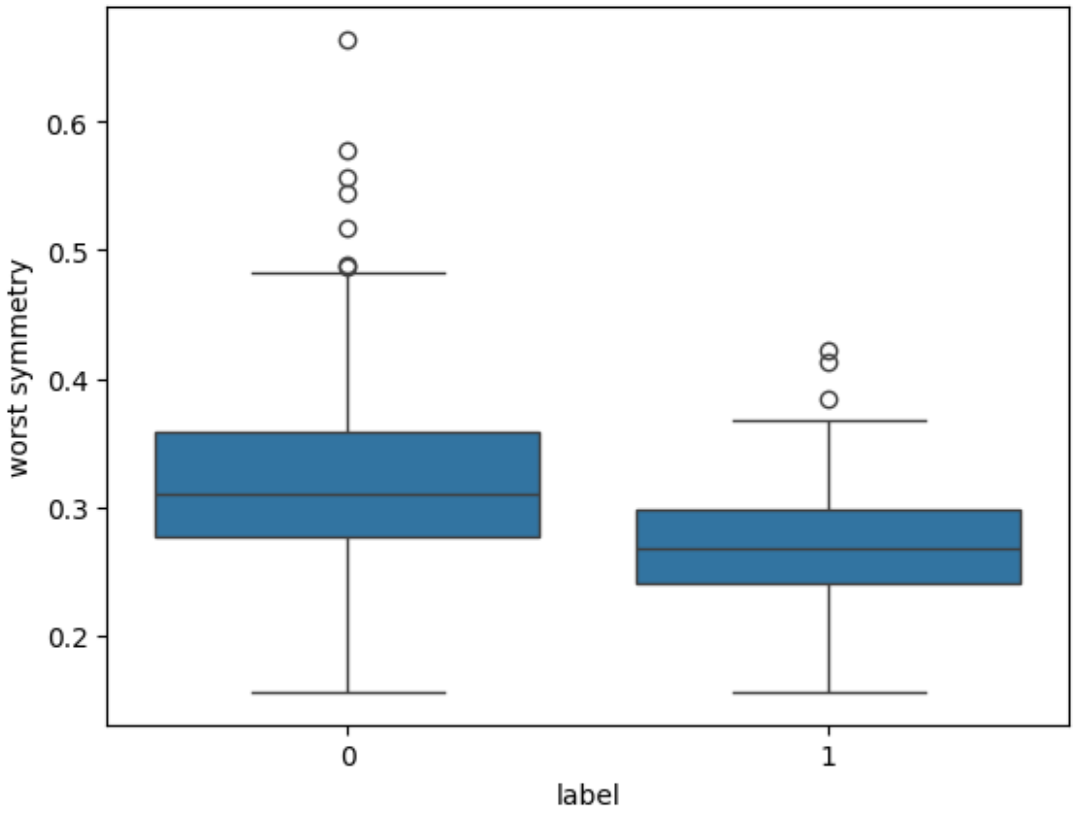

worst area
4000
3500
3000
2500
2000
1500
1000
500
0
0
1
label

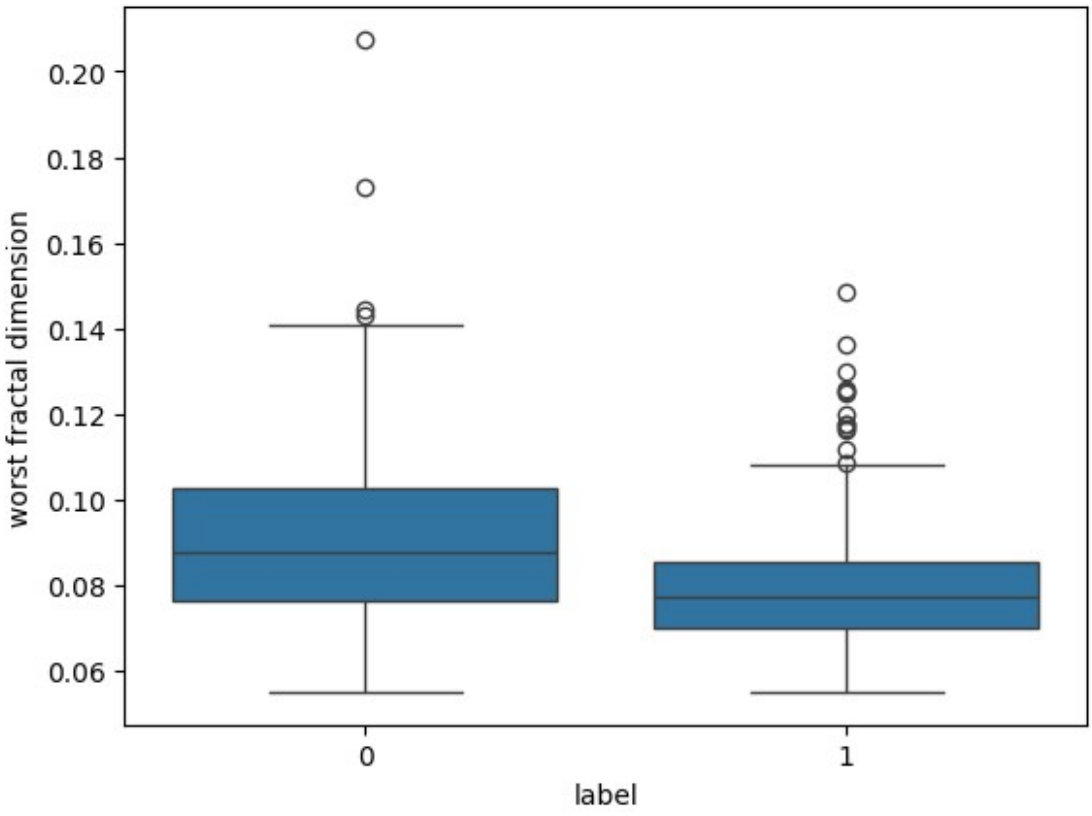

0.22
0.20
0.18
0.16
0.14
0.12
0.10
0.08
worst smoothness
0
1
label


1.0
0.8
0.6
0.4
0.2
0.0
worst compactness
0
1
label

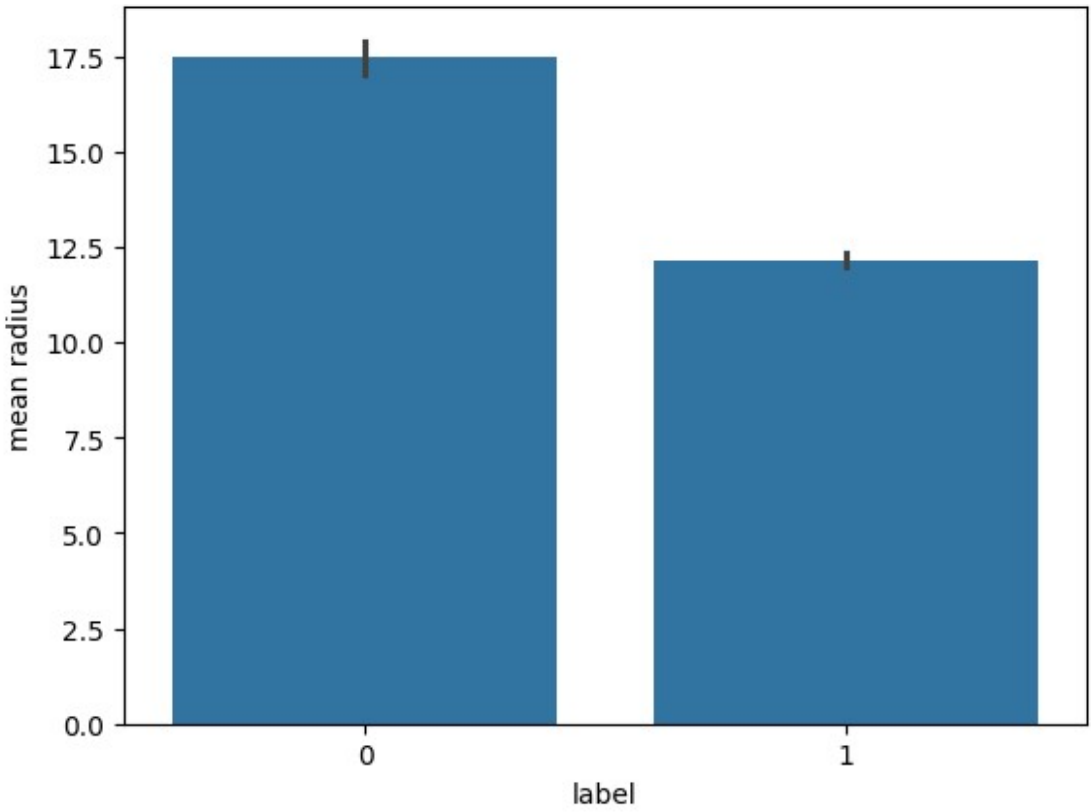

worst concavity
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
1
label

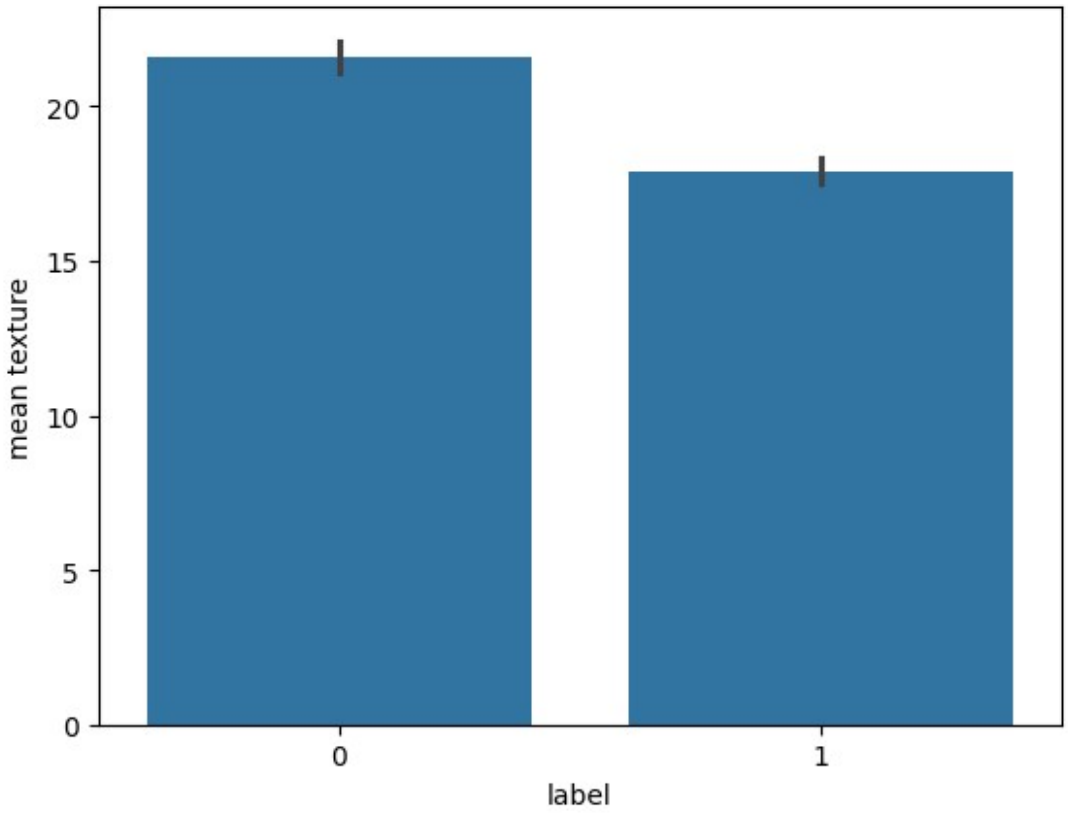

worst concave points
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
1
label

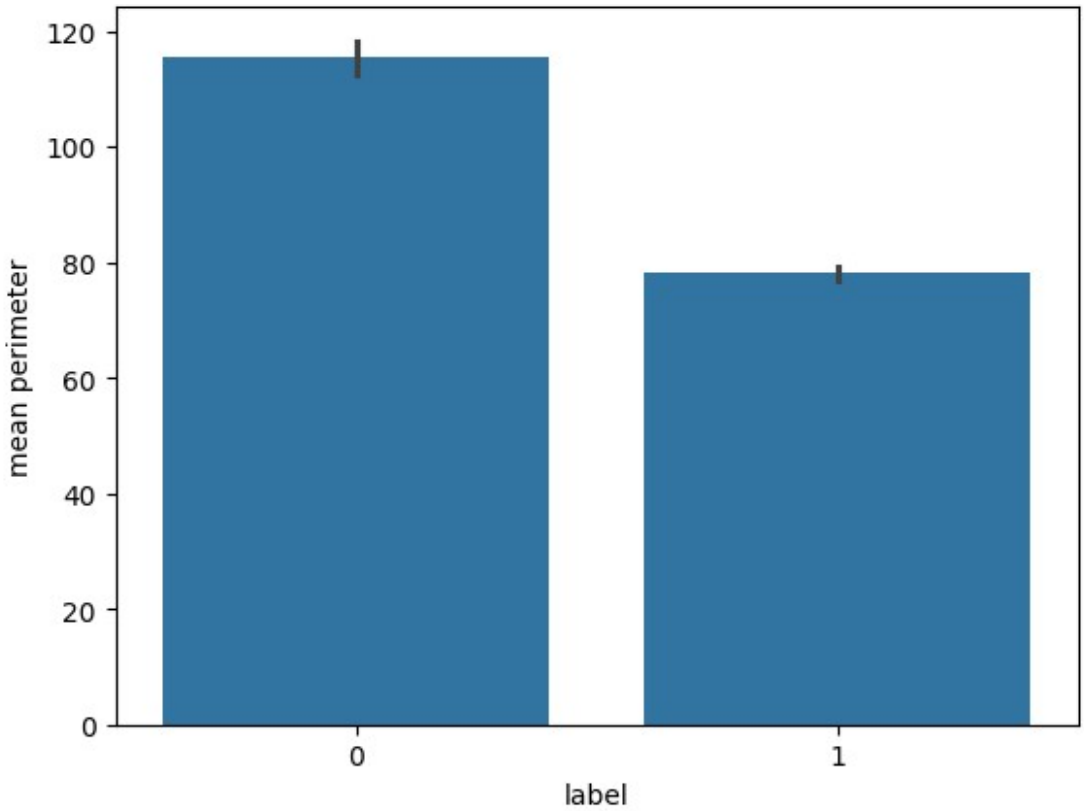

worst symmetry
0.6
0.5
0.4
0.3
0.2
0
1
label

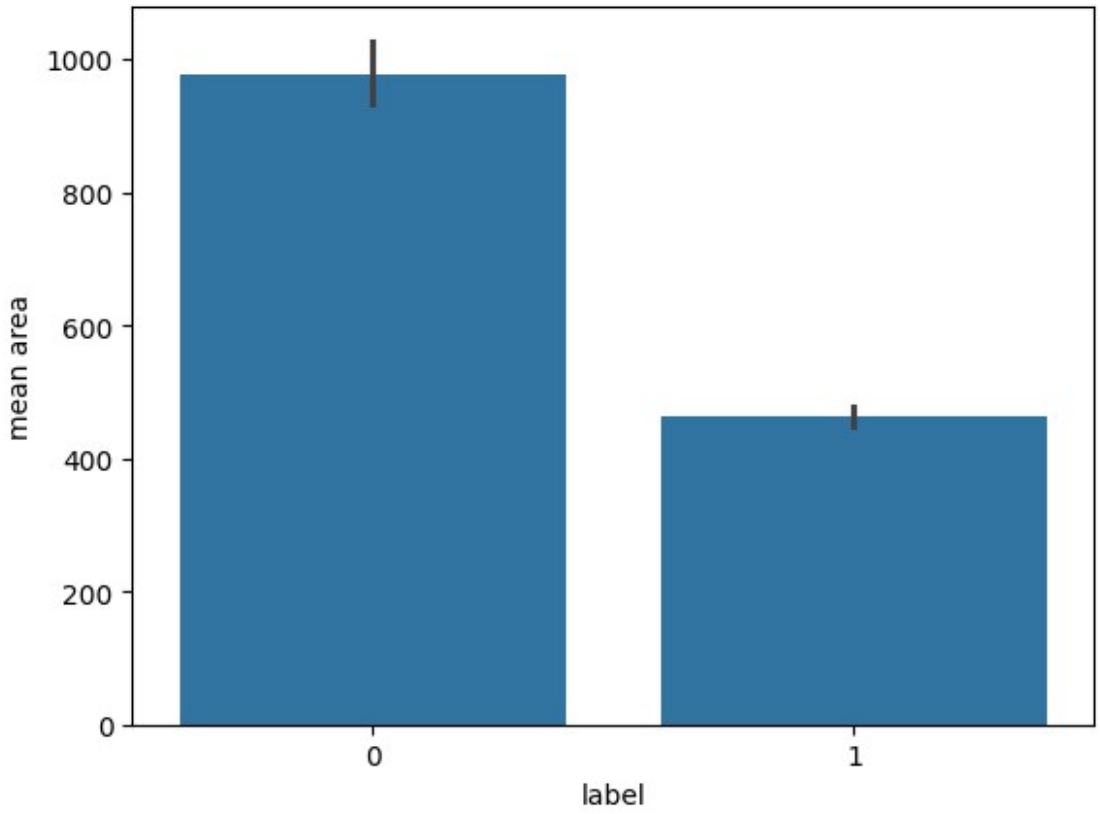

worst fractal dimension
0.20
0.18
0.16
0.14
0.12
0.10
0.08
0.06
0
1
label

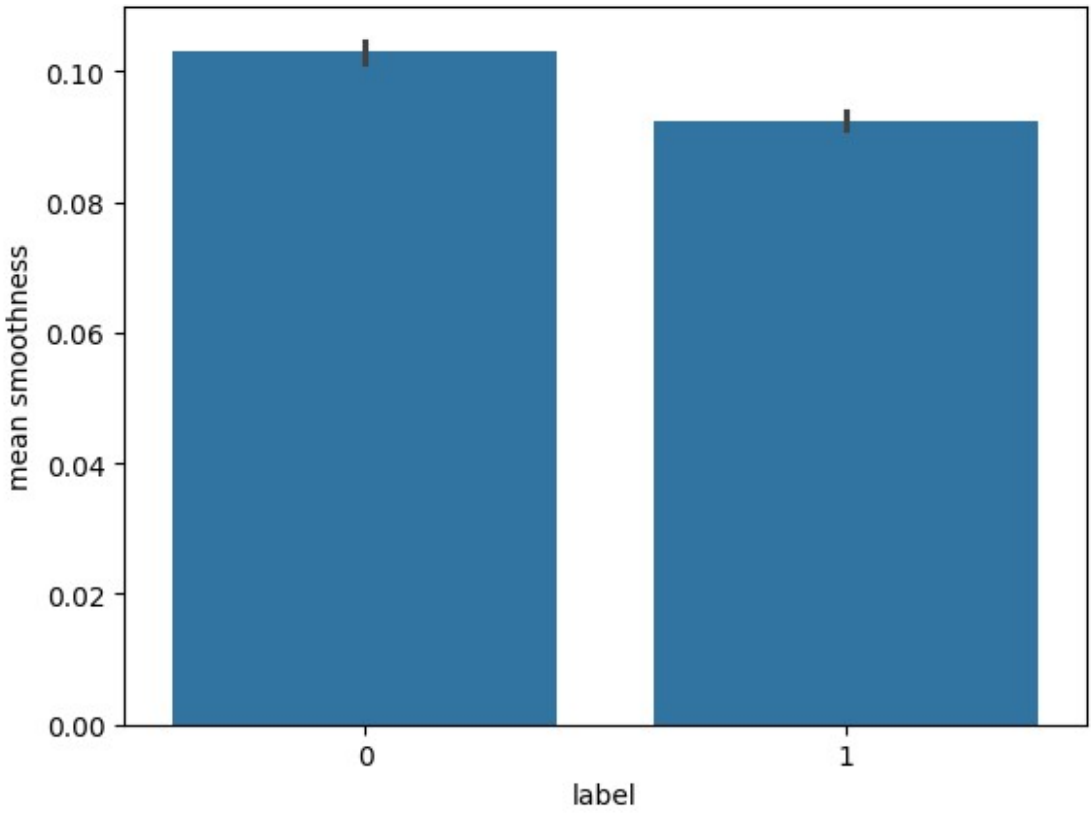
```
for i in data_frame.columns:
    if i != 'label':
        sns.barplot(x='label', y=i, data=data_frame)
        plt.show()
```

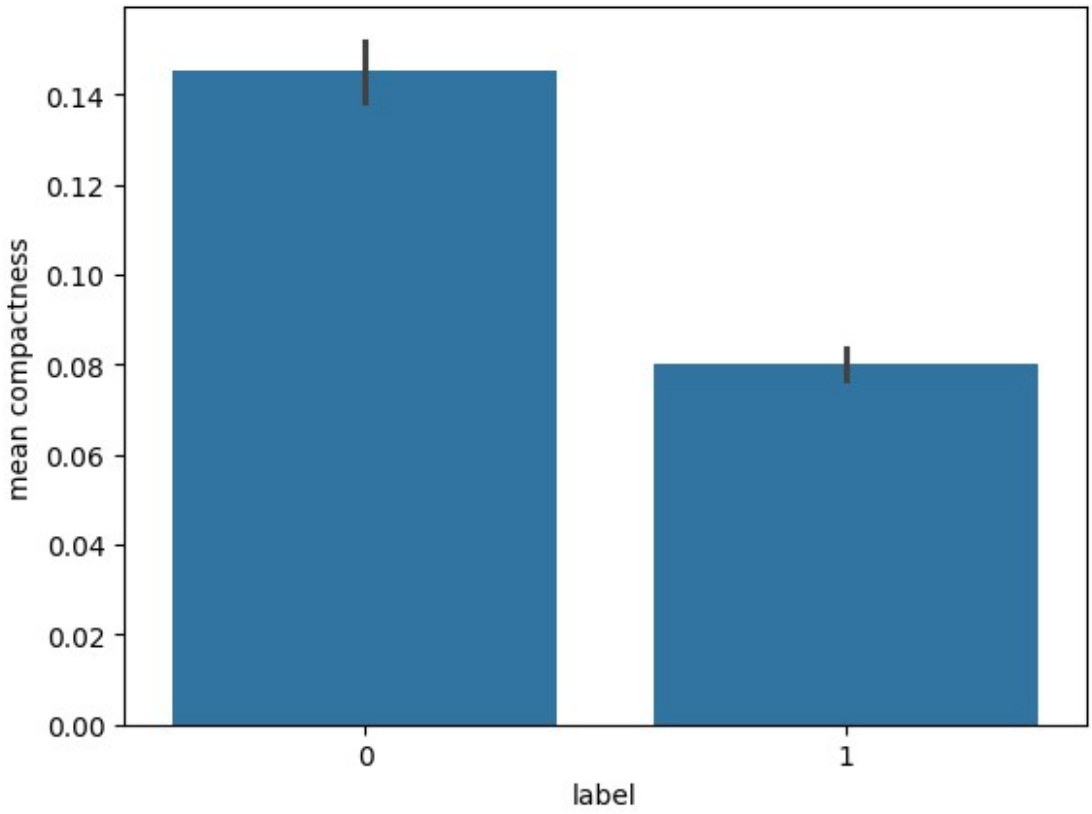


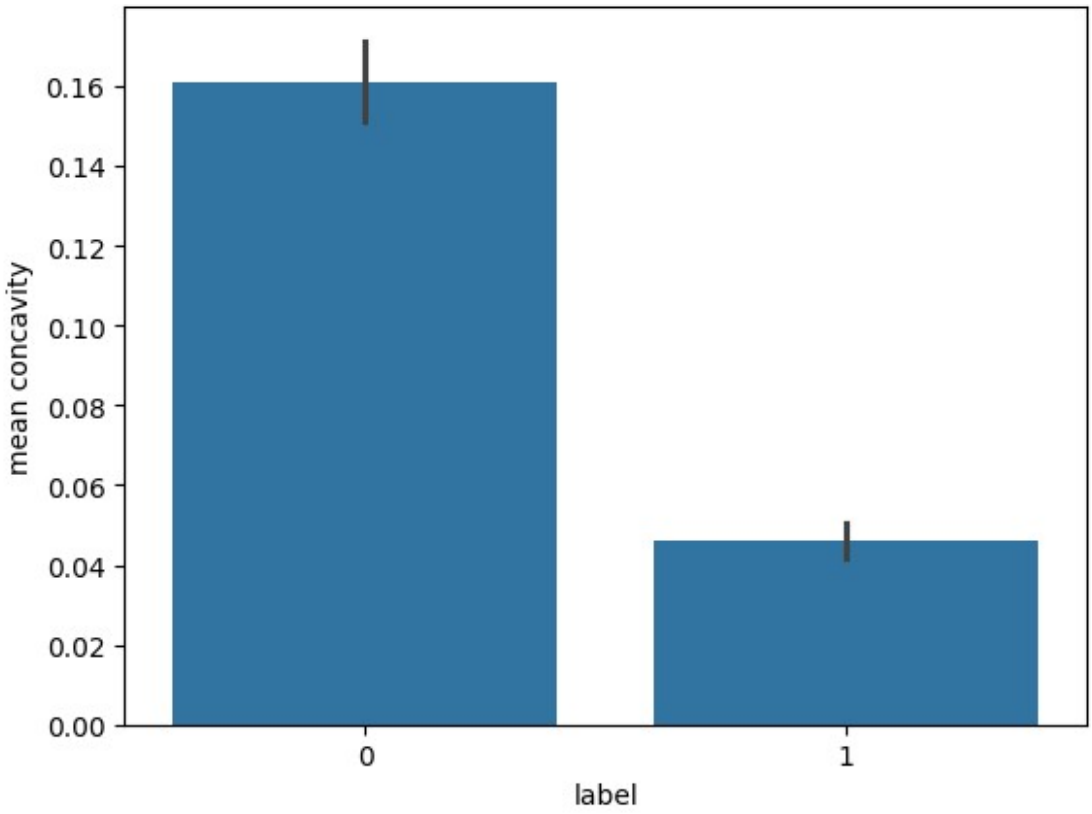

mean texture
20
15
10
5
0
0
1
label

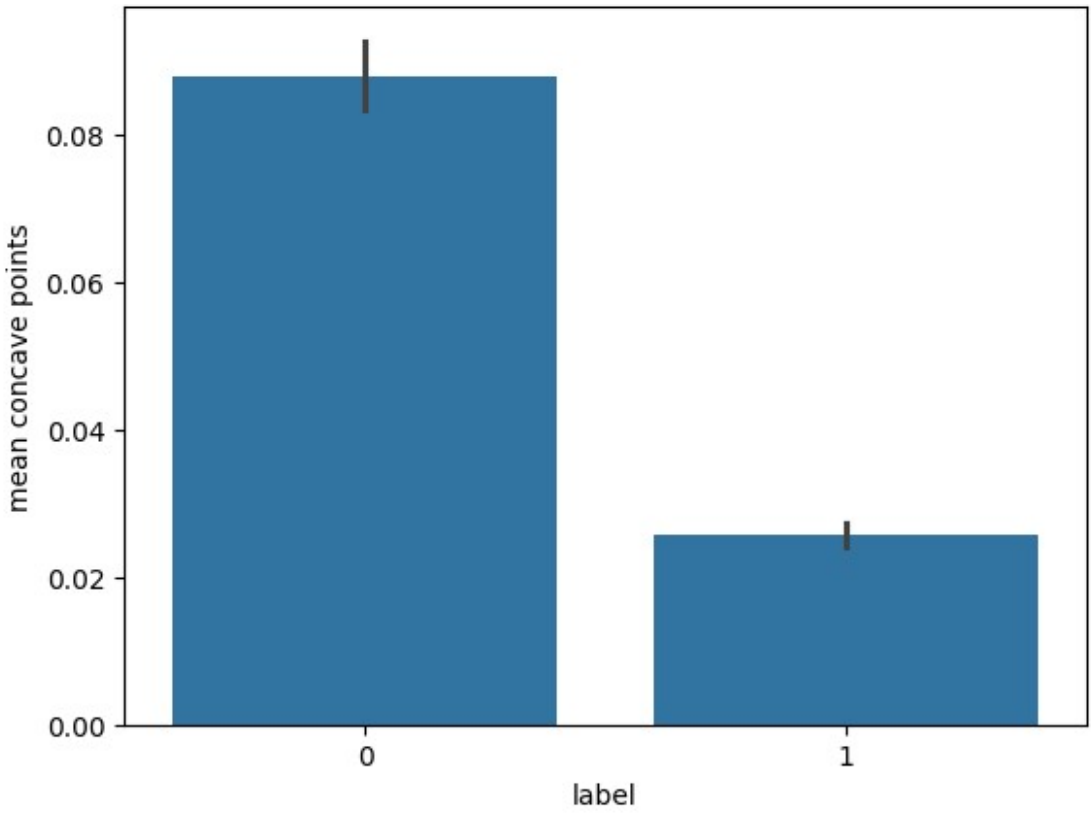

mean perimeter
120
100
80
60
40
20
0
0
1
label

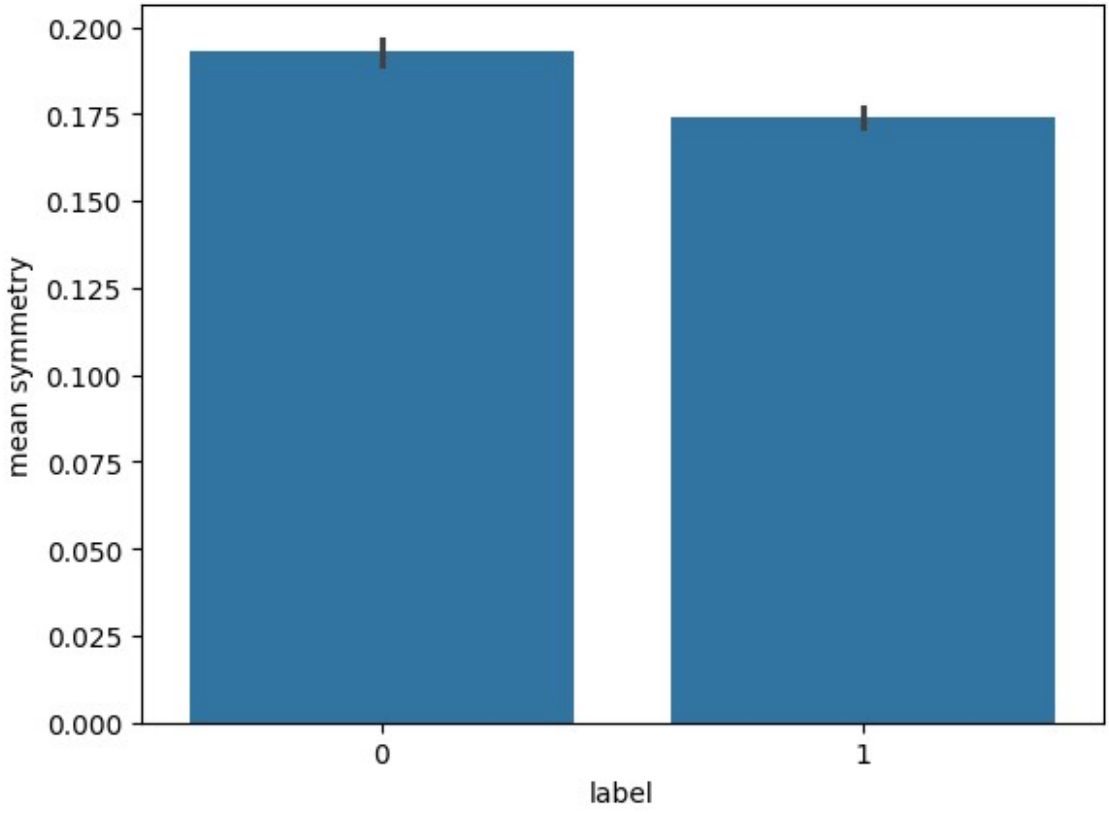

1000
800
600
400
200
0
mean area
0
1
label

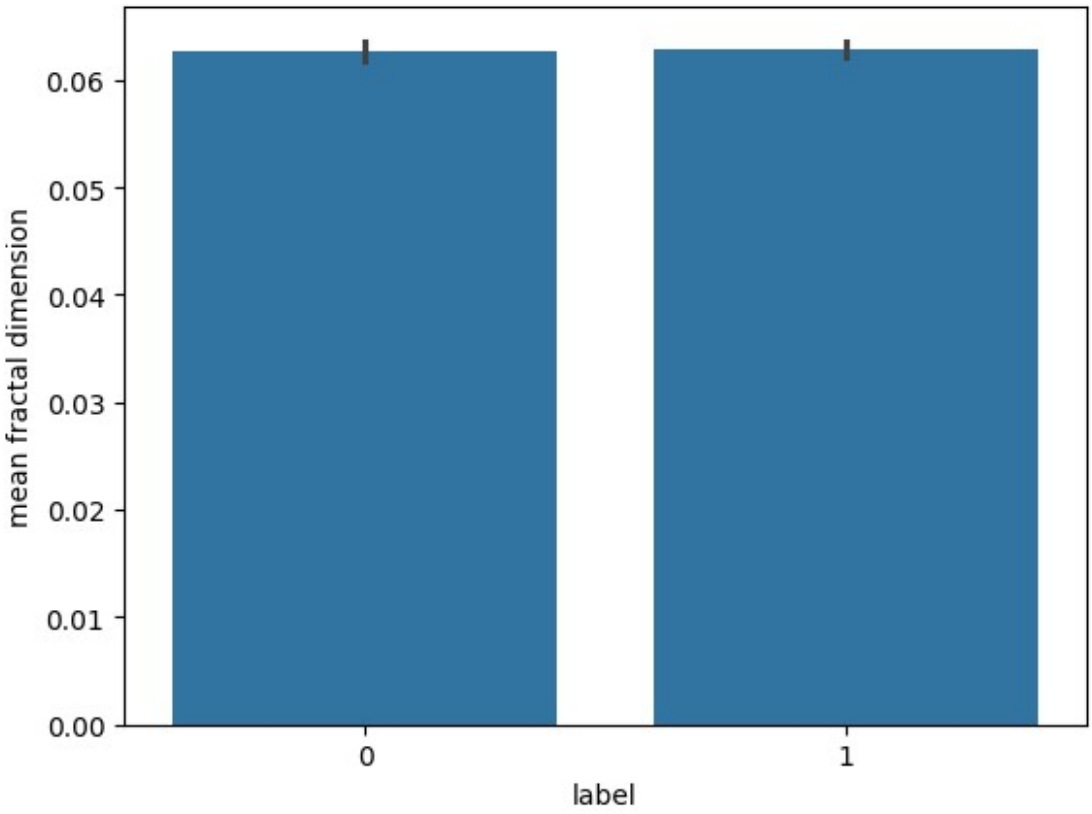

0.10
0.08
0.06
0.04
0.02
0.00
mean smoothness
0
1
label

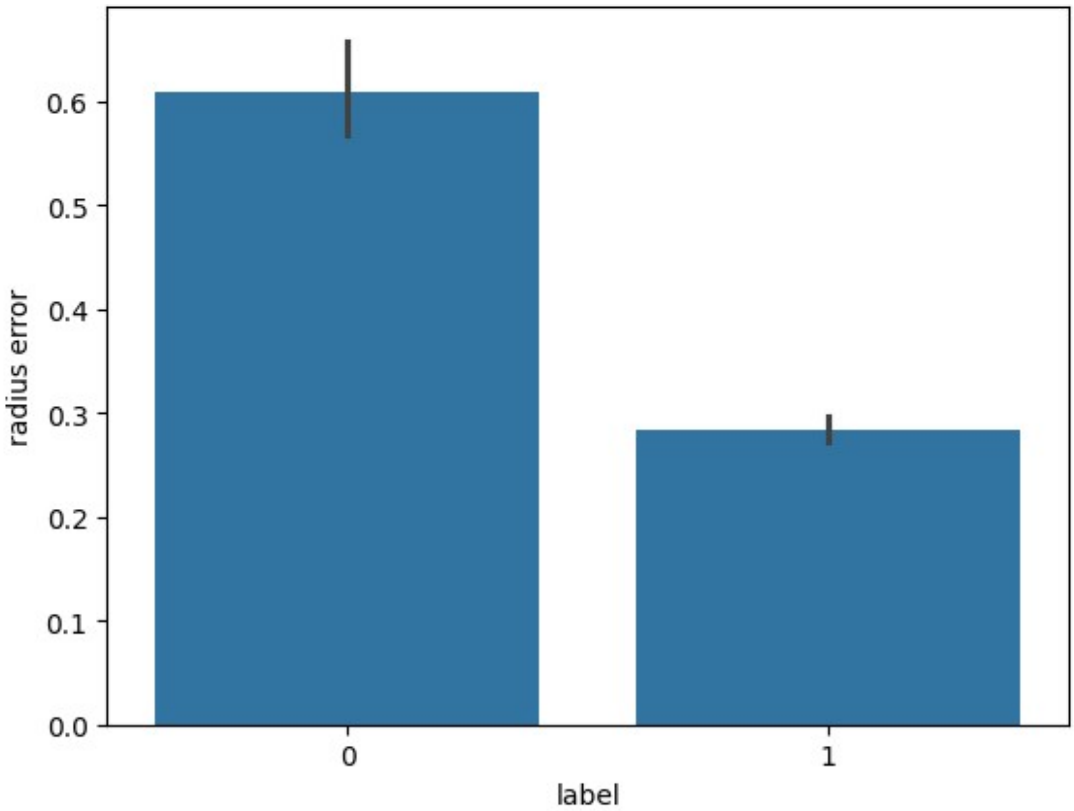

mean compactness
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
1
label

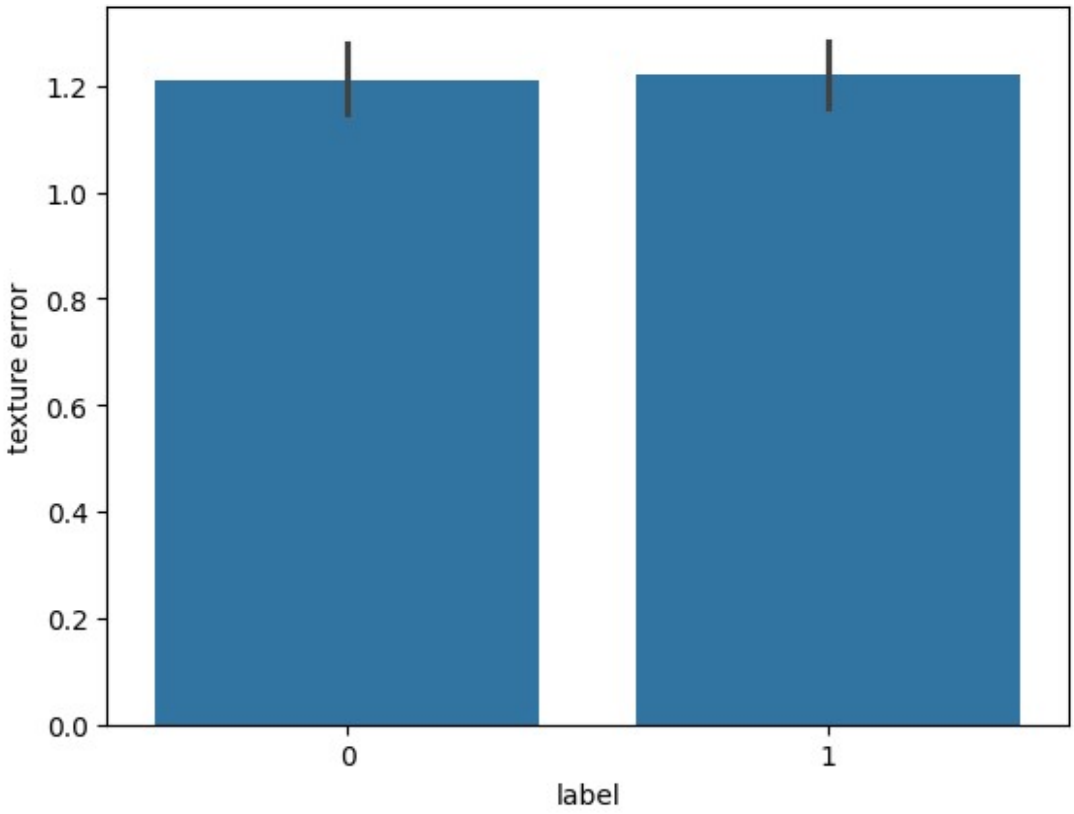

mean concavity
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
1
label

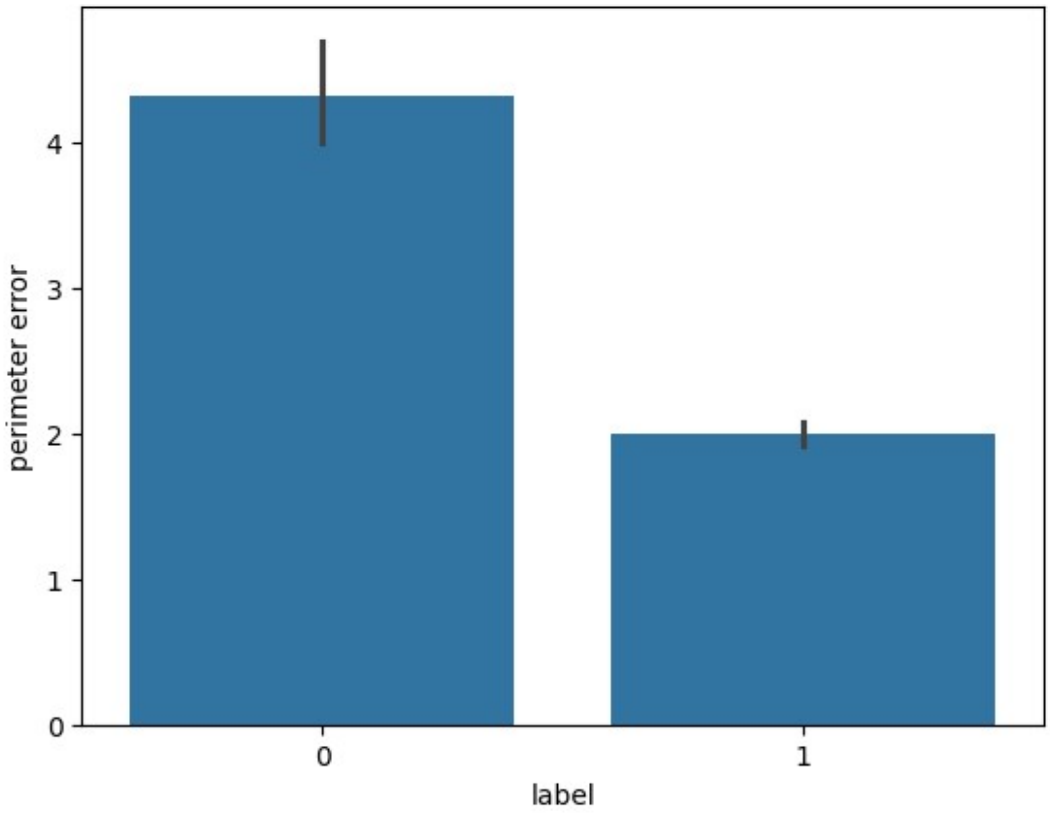

mean concave points
0.08
0.06
0.04
0.02
0.00
0
1
label

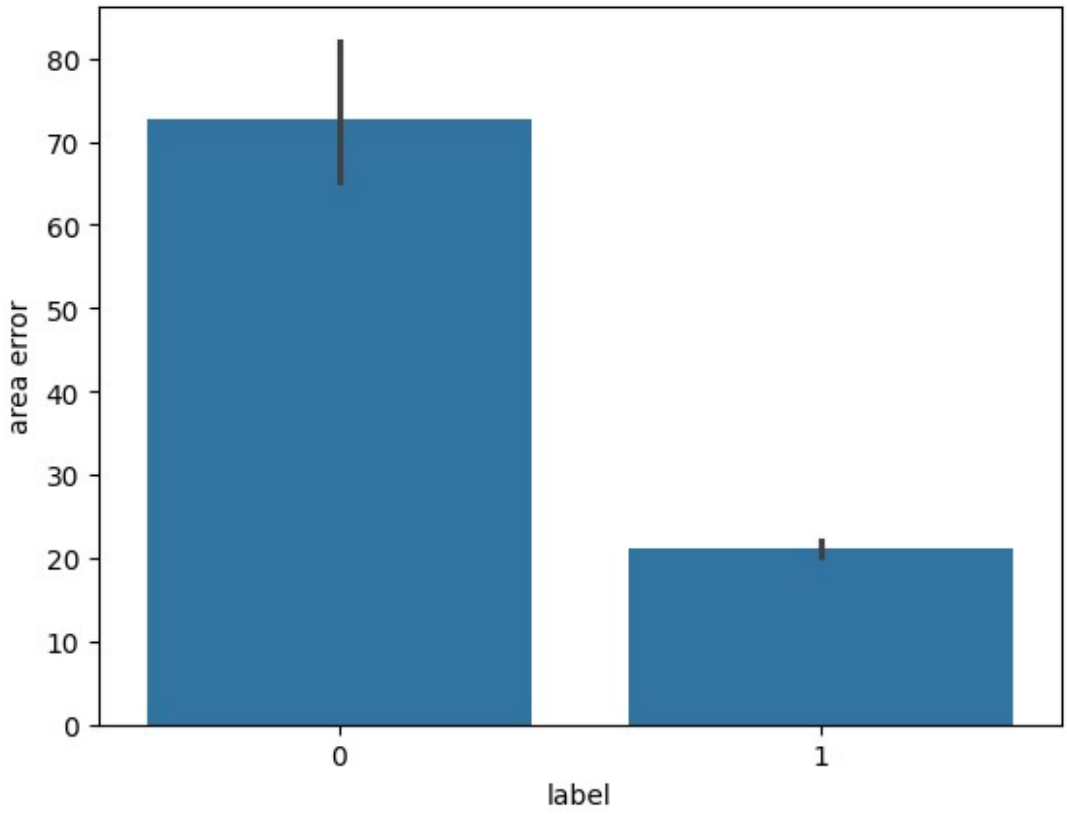

mean symmetry
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
1
label

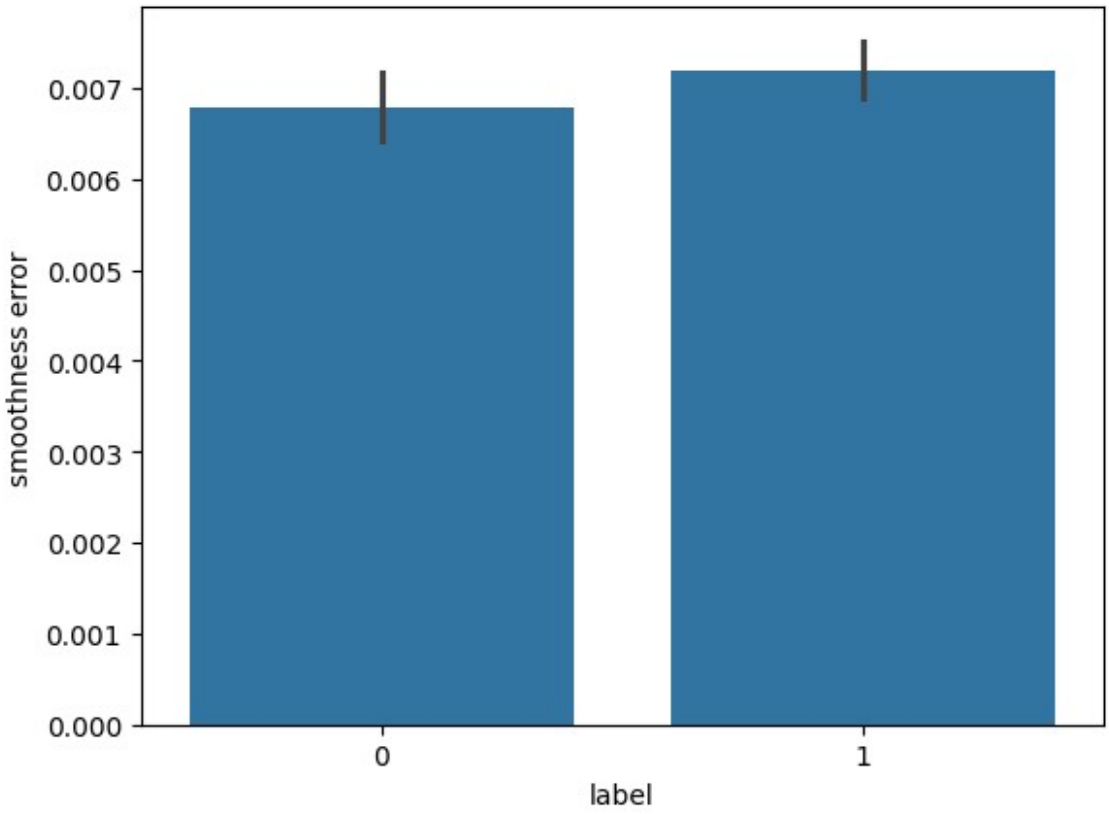

mean fractal dimension
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
1
label

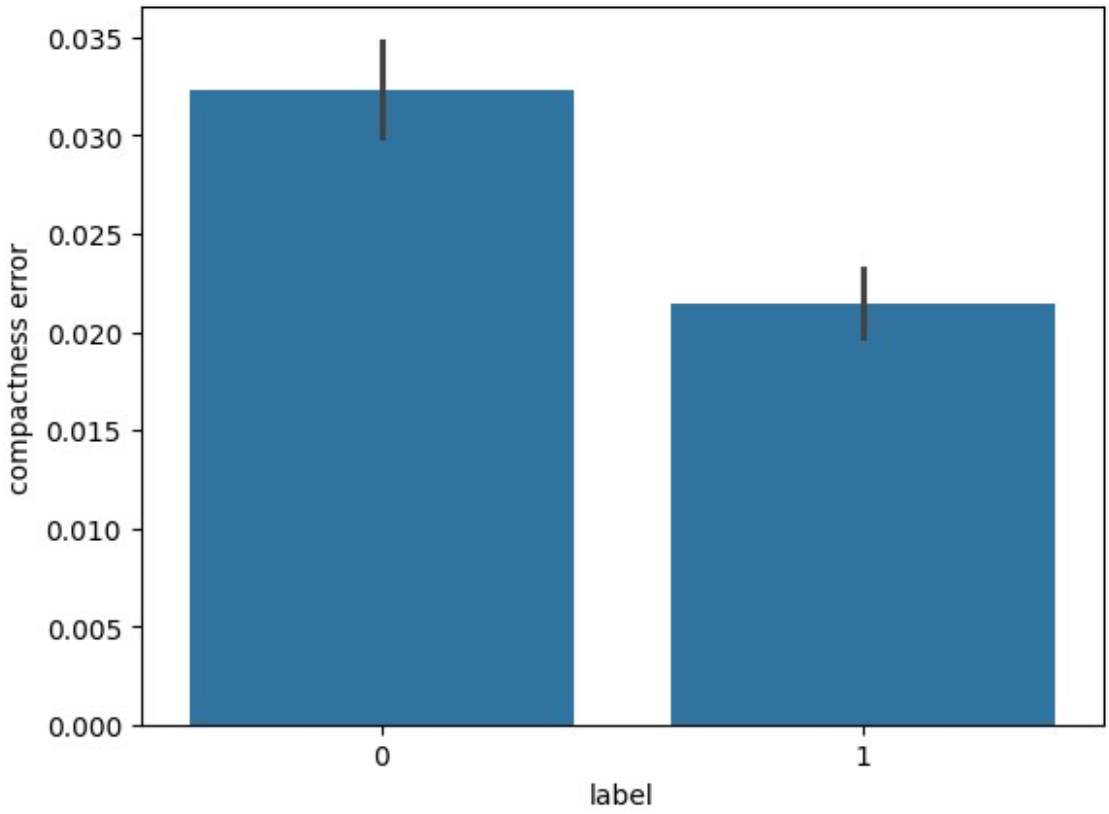

radius error
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
1
label

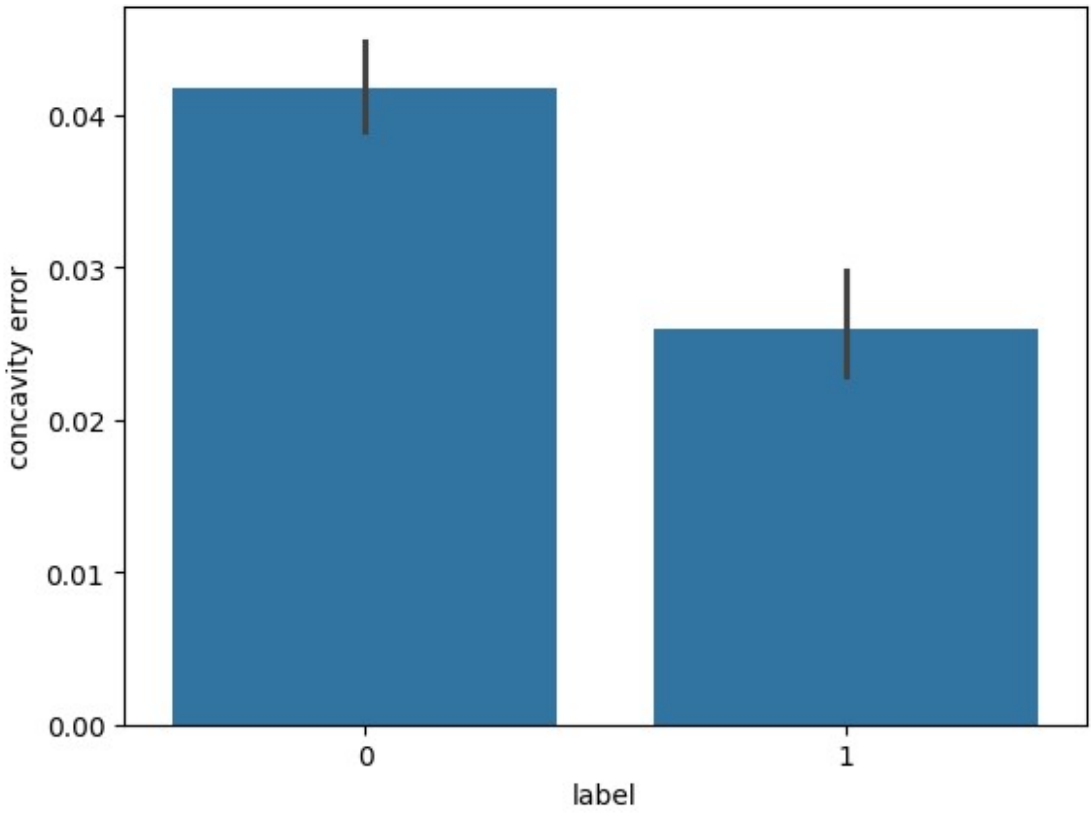

texture error
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
1
label

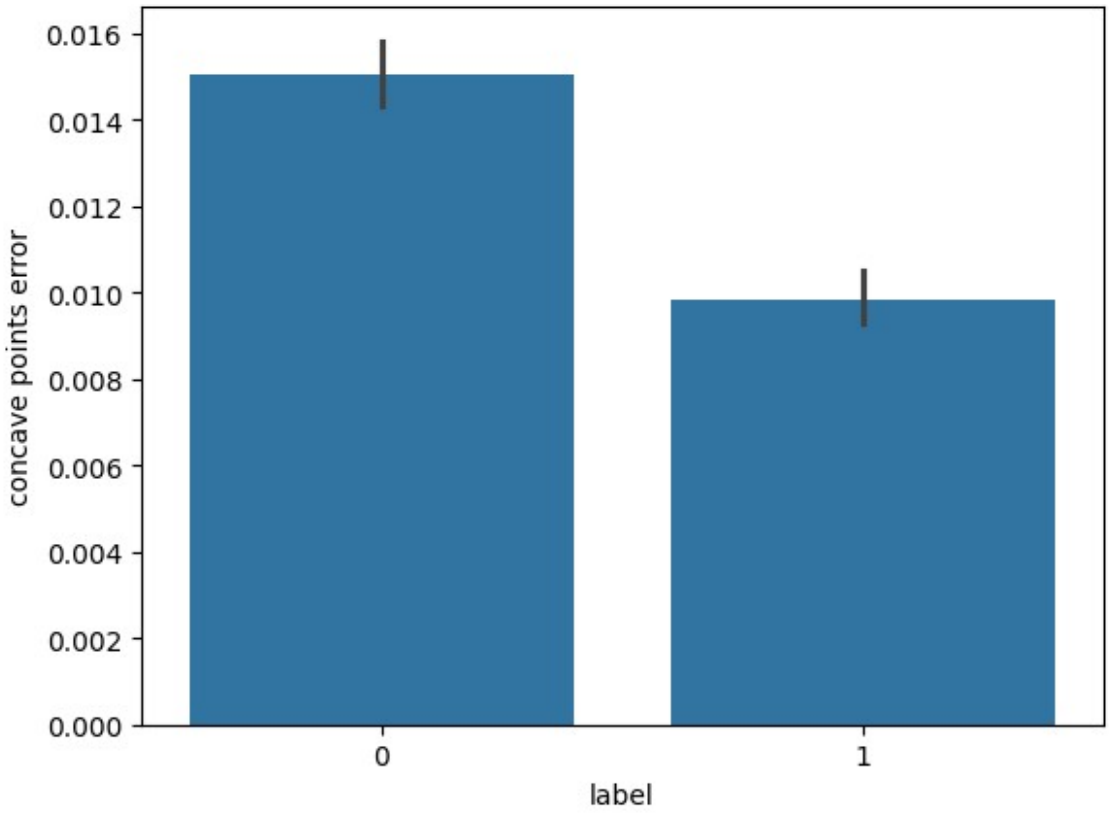

perimeter error
4
3
2
1
0
0
1
label

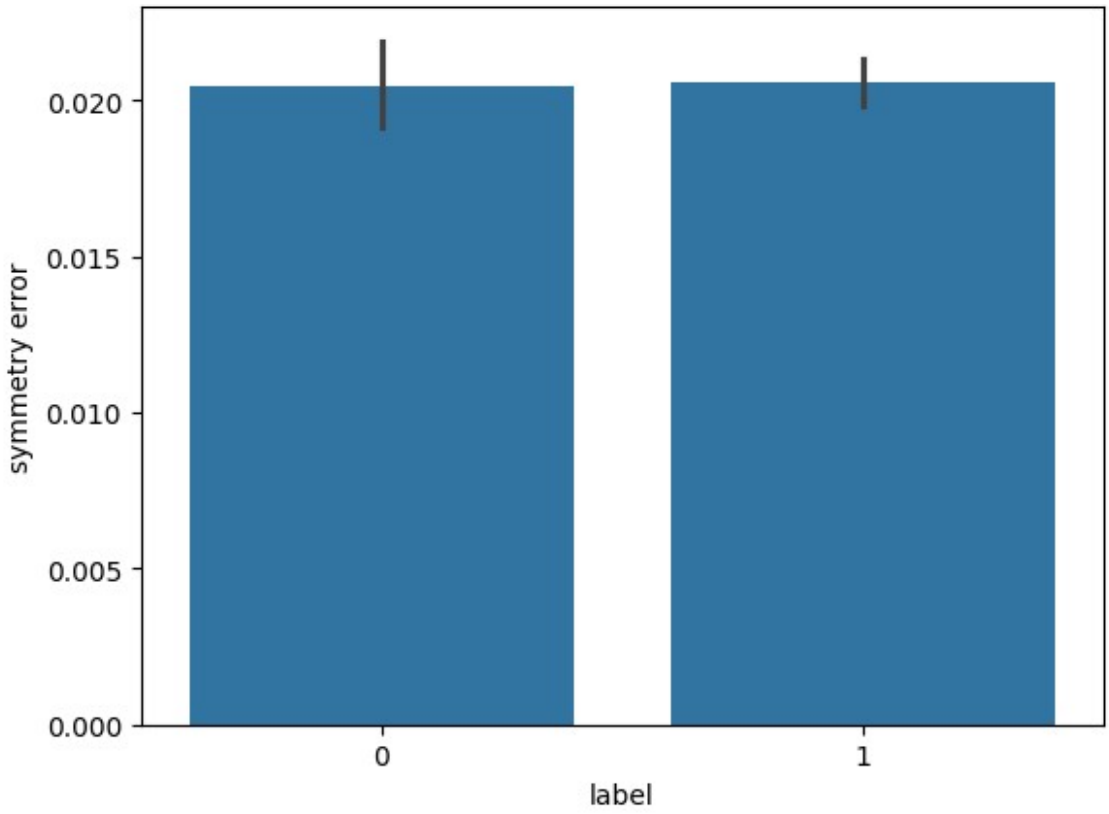

80
70
60
50
40
30
20
10
0
area error
0
1
label

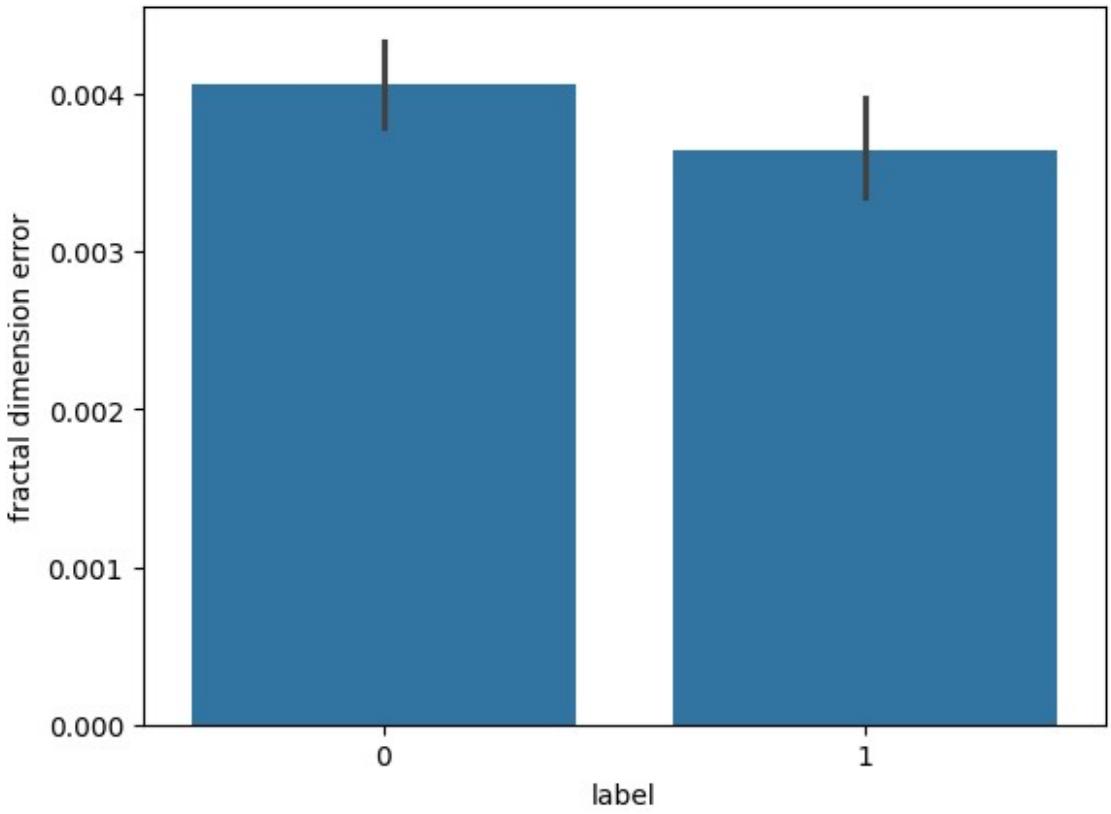

0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
smoothness error
0
1
label

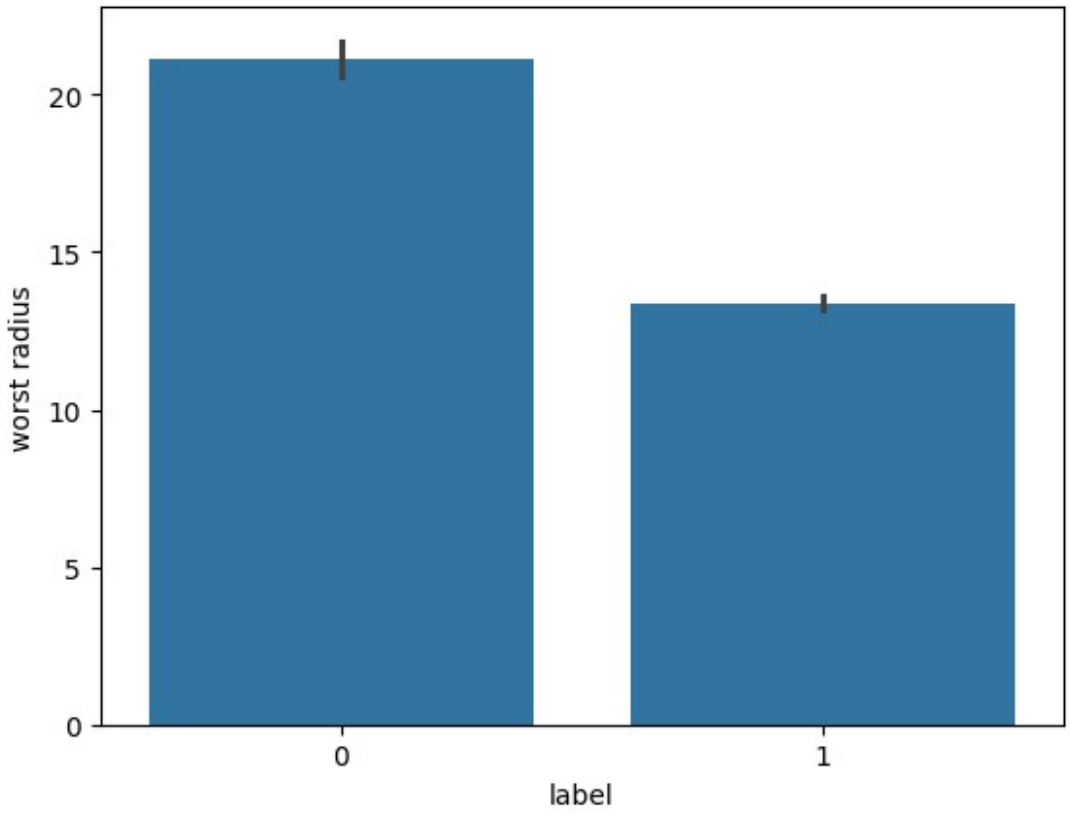

0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
compactness error
0
1
label

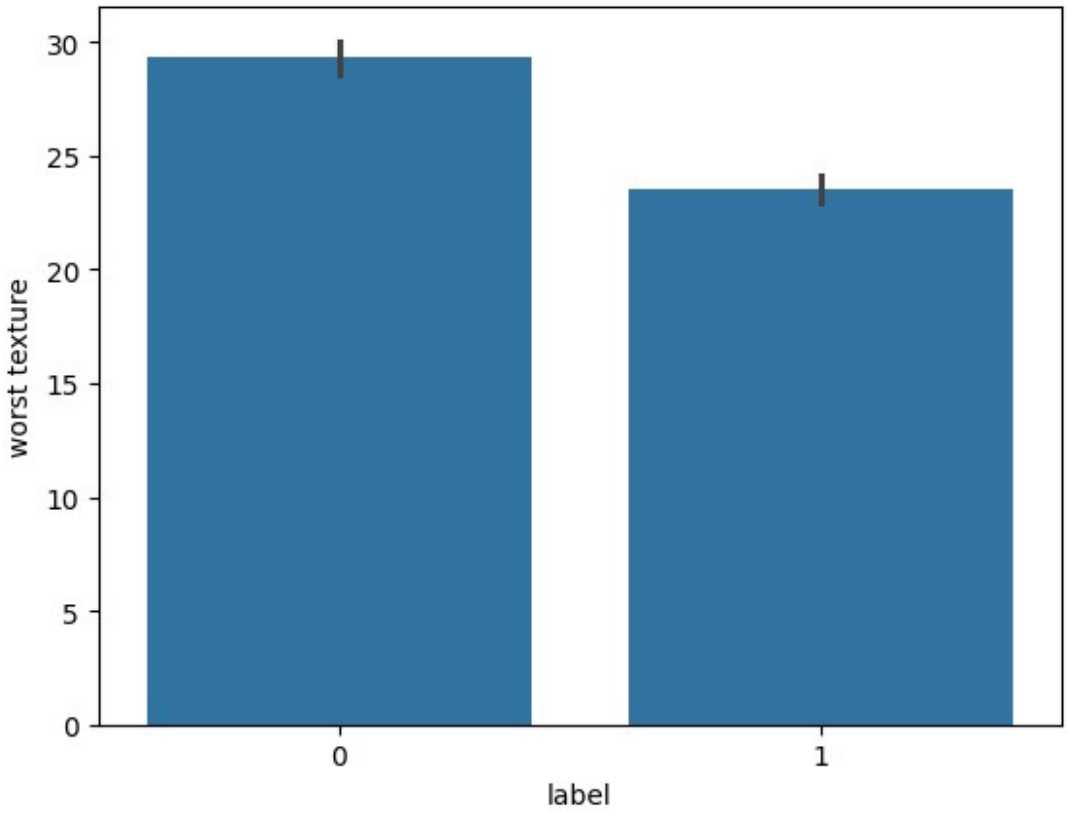

0.04
0.03
0.02
0.01
0.00
concavity error
0
1
label

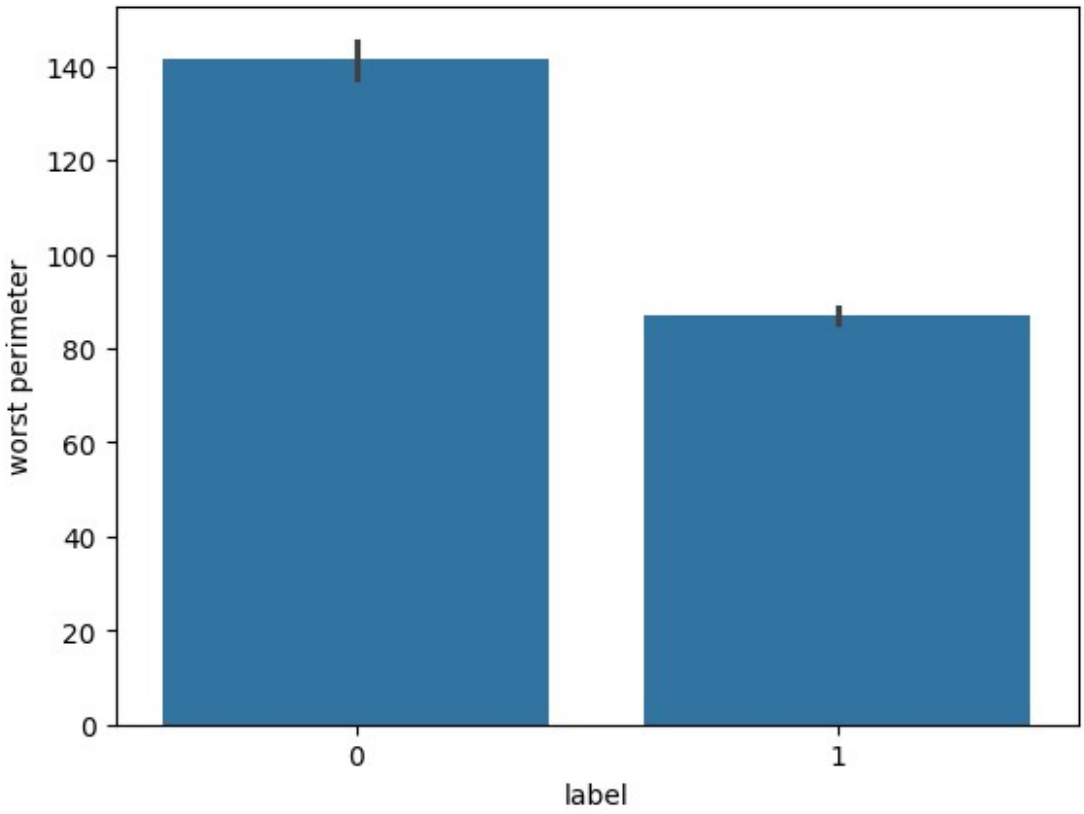

0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
concave points error
0
1
label

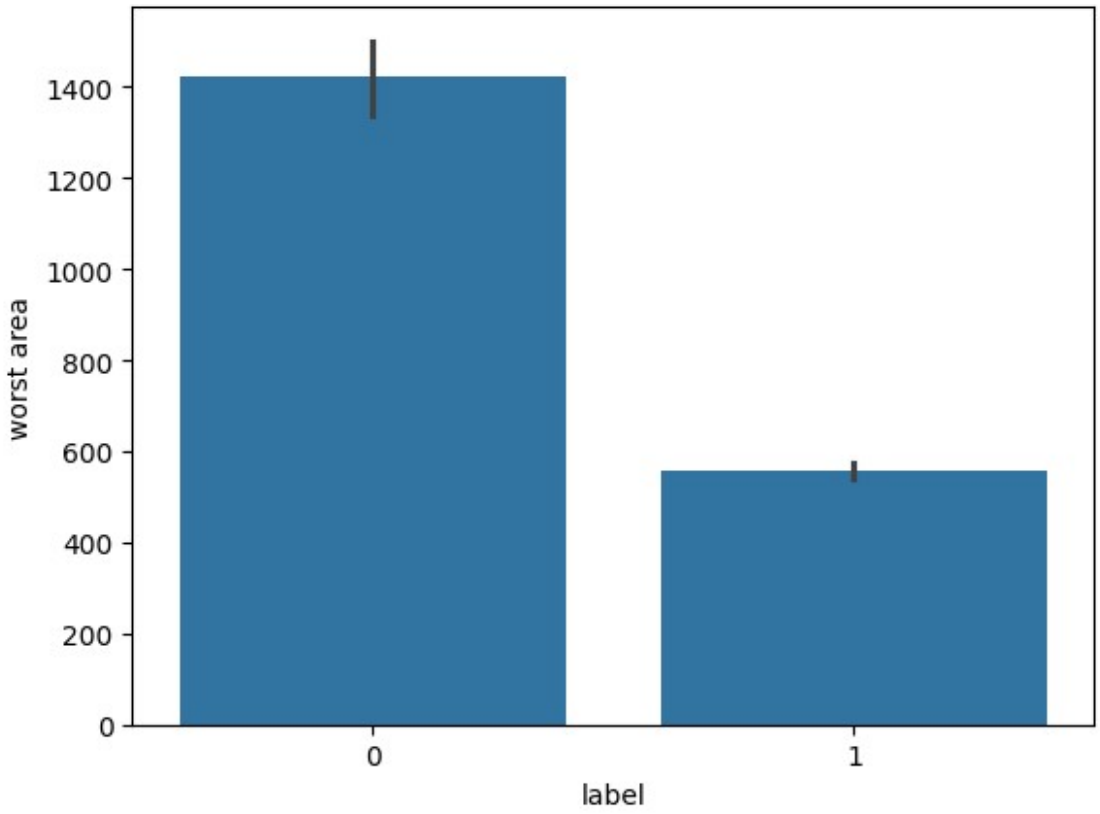

0.020
0.015
0.010
0.005
0.000
symmetry error
0
1
label

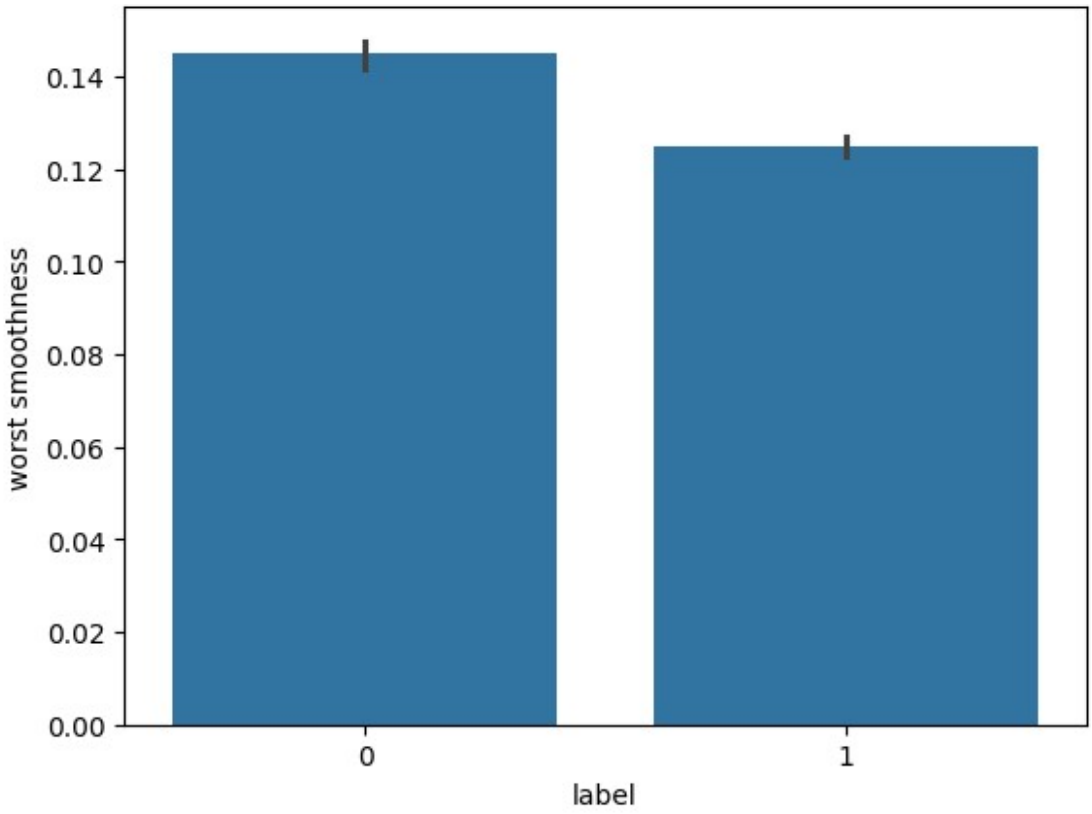

0.004
0.003
0.002
0.001
0.000
fractal dimension error
0
1
label

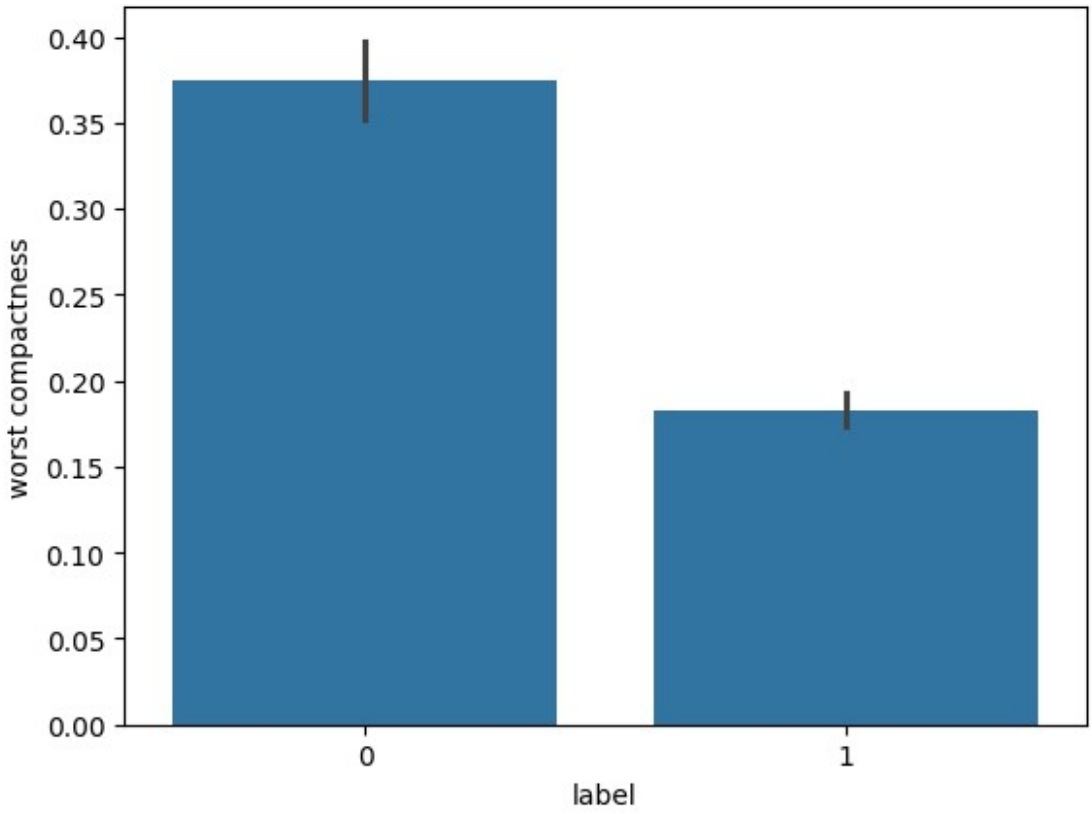

20
15
10
5
0
worst radius
0
1
label

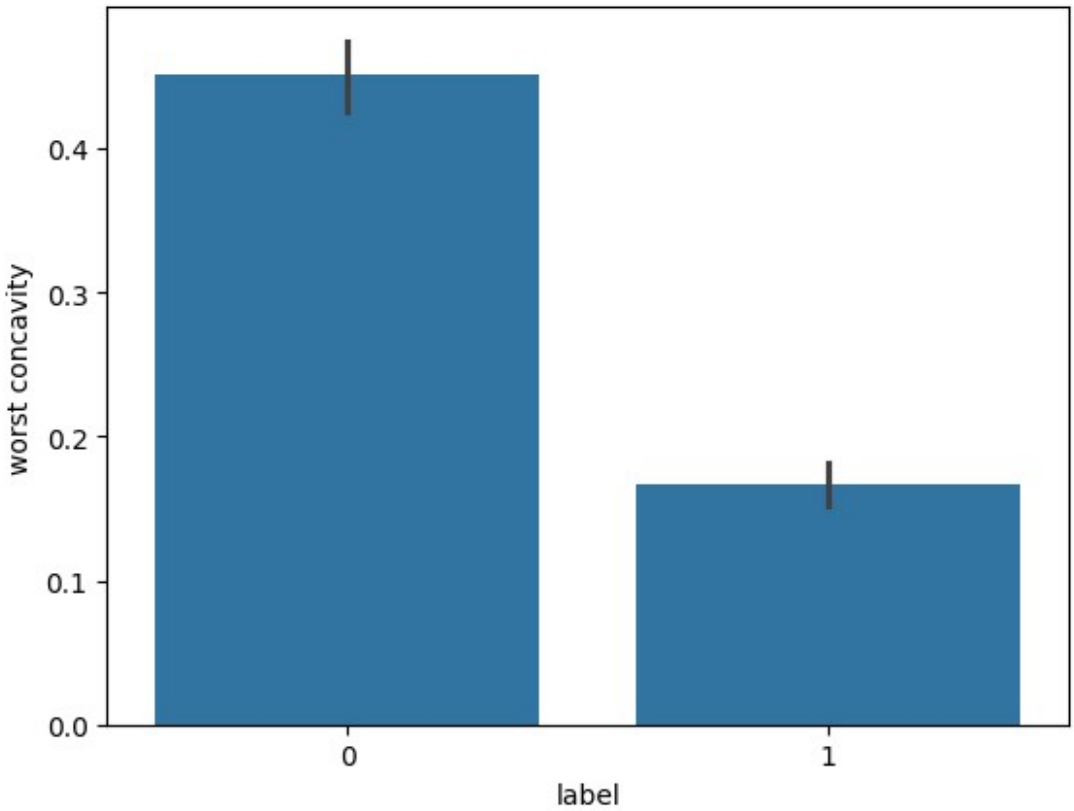

30
25
20
15
10
5
0
worst texture
0
1
label

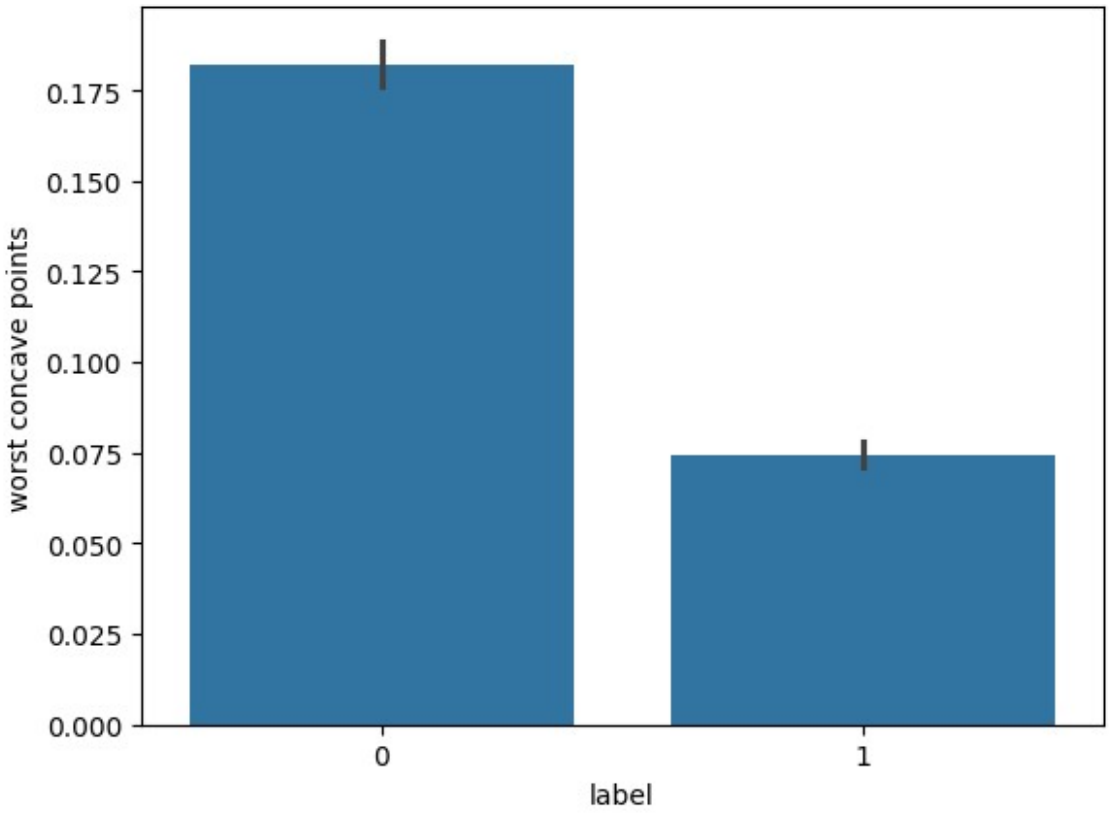

140
120
100
80
60
40
20
0
worst perimeter
0
1
label

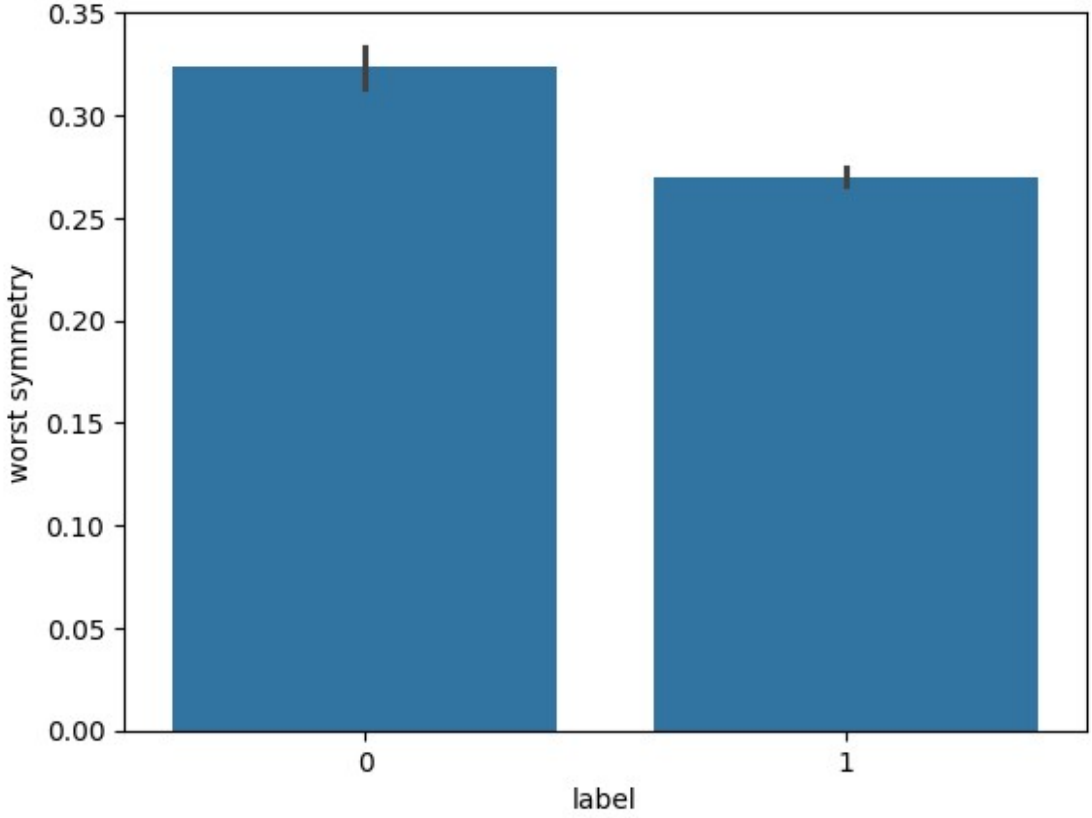

worst area
1400
1200
1000
800
600
400
200
0
0
1
label

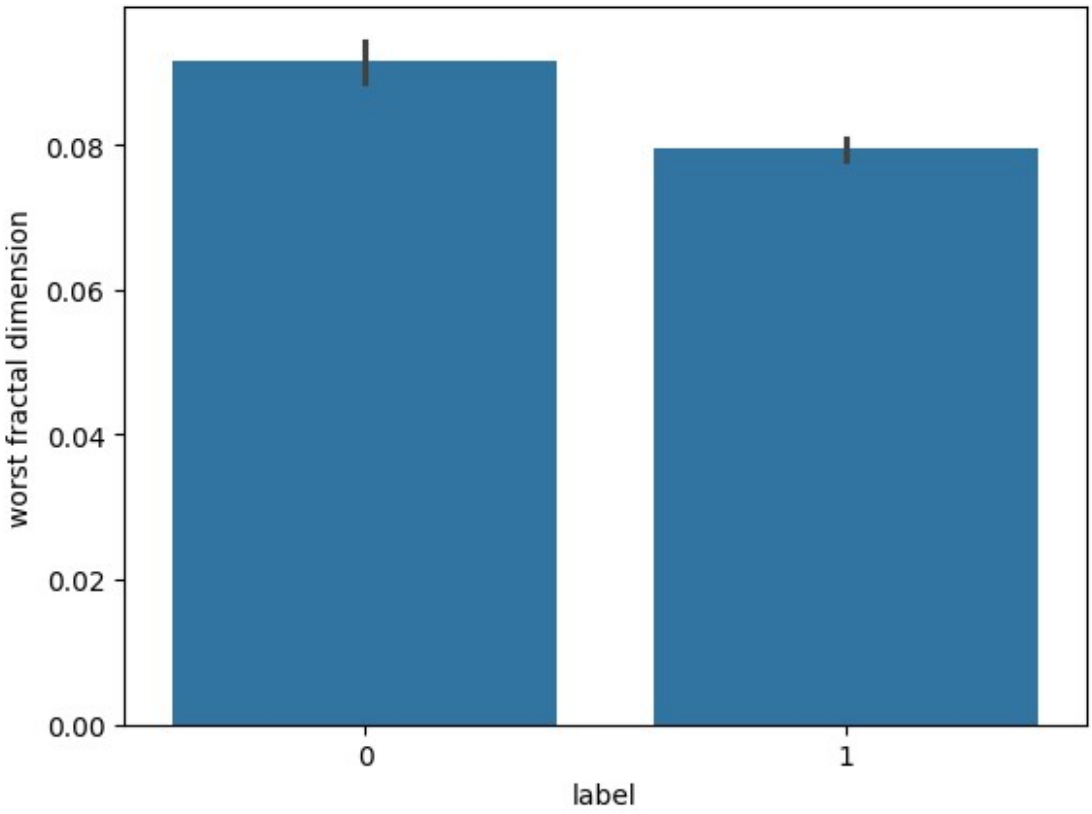

worst smoothness
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
1
label


0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
worst compactness
0
1
label


0.4
0.3
0.2
0.1
0.0
worst concavity
0
1
label

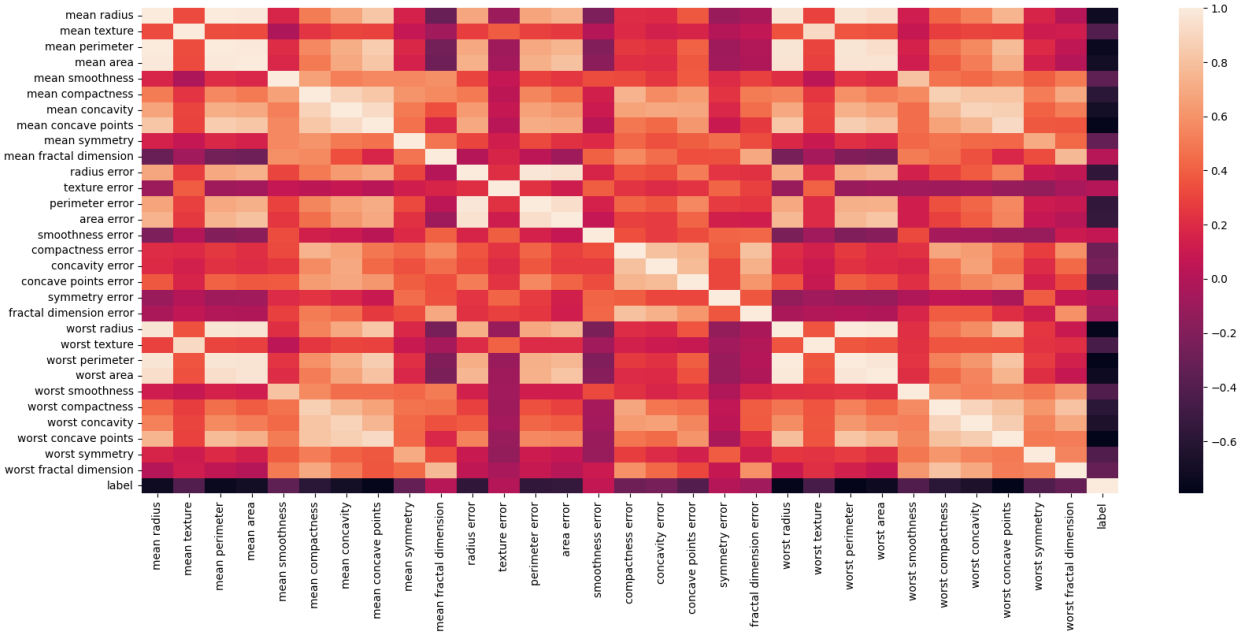

worst concave points
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
1
label


worst symmetry
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
1
label




```
fig, ax = plt.subplots(figsize=(20, 8))
sns.heatmap(data_frame.corr(), annot = False)
```

```
<Axes: >
```



Logistic Regression

- Descripción: Modelo estadístico utilizado para predecir probabilidades en problemas de clasificación binaria, transformando la salida lineal mediante una función logística.
- Ventajas: Fácil de interpretar, eficiente computacionalmente y proporciona probabilidades que ayudan en la toma de decisiones.
- Implementación: Se puede implementar fácilmente con Scikit-Learn utilizando LogisticRegression, ajustando el modelo a los datos y evaluando su precisión.

```
# comprobando valores faltantes
data_frame.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
label                       0
dtype: int64
```

```
# Se analiza la distribución del objetivo:
data_frame['label'].value_counts()
```

```
label
1    357
```

```
0    212
Name: count, dtype: int64
```

```
X = data_frame.drop(columns='label', axis=1)
Y = data_frame['label']
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2, random_state=2)
```

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

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

```
model = LogisticRegression()
```

```
model.fit(X_train, Y_train)
```

```
/usr/local/lib/python3.11/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()
```

```
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
```

```
print('Accuracy on training data = ', training_data_accuracy)
```

```
Accuracy on training data =  0.9472527472527472
```

```
# accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
```

```
print('Accuracy on test data = ', test_data_accuracy)
```

```
Accuracy on test data =  0.9298245614035088
```

Para realizar una predicción con nuevos datos, primero se convierten a un arreglo NumPy y se ajustan las dimensiones:

```
input_data =
(13.30,14.50,88.00,562.5,0.09750,0.08200,0.06750,0.04820,0.1890,0.0585
0,0.2720,0.7950,2.070,23.80,0.008550,0.01500,0.02420,0.01350,0.02010,0
.00260,15.25,19.50,100.20,708.5,0.1450,0.1780,0.2410,0.1305,0.2985,0.0
7350)

# change the input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the numpy array as we are predicting for one datapoint
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

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

if (prediction[0] == 0):
  print('The Breast cancer is Malignant')

else:
  print('The Breast Cancer is Benign')
```

```
[1]
The Breast Cancer is Benign

/usr/local/lib/python3.11/dist-packages/sklearn/utils/
validation.py:2739: UserWarning: X does not have valid feature names,
but LogisticRegression was fitted with feature names
  warnings.warn(
```

Usando Redes neuronales

- Descripción: Modelos computacionales inspirados en el cerebro humano, compuestos por capas de nodos que transforman entradas a través de funciones de activación, adecuados para tareas complejas.
- Ventajas: Capacidad para modelar relaciones no lineales complejas, flexibilidad para diferentes tareas y escalabilidad para grandes volúmenes de datos.
- Implementación: Se puede construir un modelo secuencial en Keras utilizando Sequential, añadiendo capas densas y compilando el modelo con optimizadores y funciones de pérdida específicas.

```
from sklearn.preprocessing import StandardScaler
```

La clase StandardScaler de Scikit-Learn se utiliza para estandarizar las características eliminando la media y escalando a la unidad de varianza.

```
scaler = StandardScaler()

X_train_std = scaler.fit_transform(X_train)
```

```
X_test_std = scaler.transform(X_test)
```

Para trabajar con redes neuronales, se utiliza TensorFlow, una biblioteca poderosa para el aprendizaje automático que incluye herramientas para construir y entrenar modelos complejos. Establecimiento de Semilla Aleatoria Para asegurar que los resultados sean reproducibles, se establece una semilla aleatoria:

```
import tensorflow as tf
tf.random.set_seed(3)
from tensorflow import keras
```

Se utiliza Keras para definir un modelo secuencial, que es una forma sencilla y lineal de construir redes neuronales:

```
model = keras.Sequential([
                          keras.layers.Flatten(input_shape=(30,)),
                          keras.layers.Dense(20, activation='relu'),
                          keras.layers.Dense(2, activation='sigmoid')
])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/
flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

Descripción de las Capas

- Capa Flatten: Función: Convierte la entrada (que puede ser un arreglo multidimensional) en un vector unidimensional.
- Entrada: En este caso, se espera una entrada con 30 características (por ejemplo, las características del conjunto de cáncer de mama).
- Capa Dense (Capa Oculta):
- Neuronas: 20 neuronas.
- Función de Activación: ReLU (Rectified Linear Unit), que introduce no linealidades en el modelo. Es comúnmente utilizada en capas ocultas debido a su capacidad para manejar problemas de gradiente.
- Capa Dense (Capa de Salida):
- Neuronas: 2 neuronas, que corresponden a las dos clases posibles (benigno o maligno).
- Función de Activación: Sigmoid, que produce una salida entre 0 y 1, adecuada para problemas de clasificación binaria cuando se tiene más de una clase.

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

Descripción de los Parámetros

- Optimizador (optimizer): adam: Este es un optimizador popular que combina las ventajas de dos métodos: AdaGrad y RMSProp. Adam ajusta automáticamente la tasa de aprendizaje durante el entrenamiento, lo que a menudo resulta en una convergencia más rápida y eficiente.
- Función de Pérdida (loss): sparse_categorical_crossentropy: Esta función se utiliza para problemas de clasificación múltiple donde las etiquetas son enteros. En este caso, dado que el conjunto tiene dos clases (benigno o maligno), esta función calcula la diferencia entre las predicciones del modelo y las etiquetas verdaderas. Es adecuada cuando las etiquetas están en forma de enteros en lugar de vectores one-hot.
- Métricas (metrics): accuracy: Esta métrica evalúa la precisión del modelo durante el entrenamiento y la validación. Mide la proporción de predicciones correctas respecto al total de predicciones realizadas.

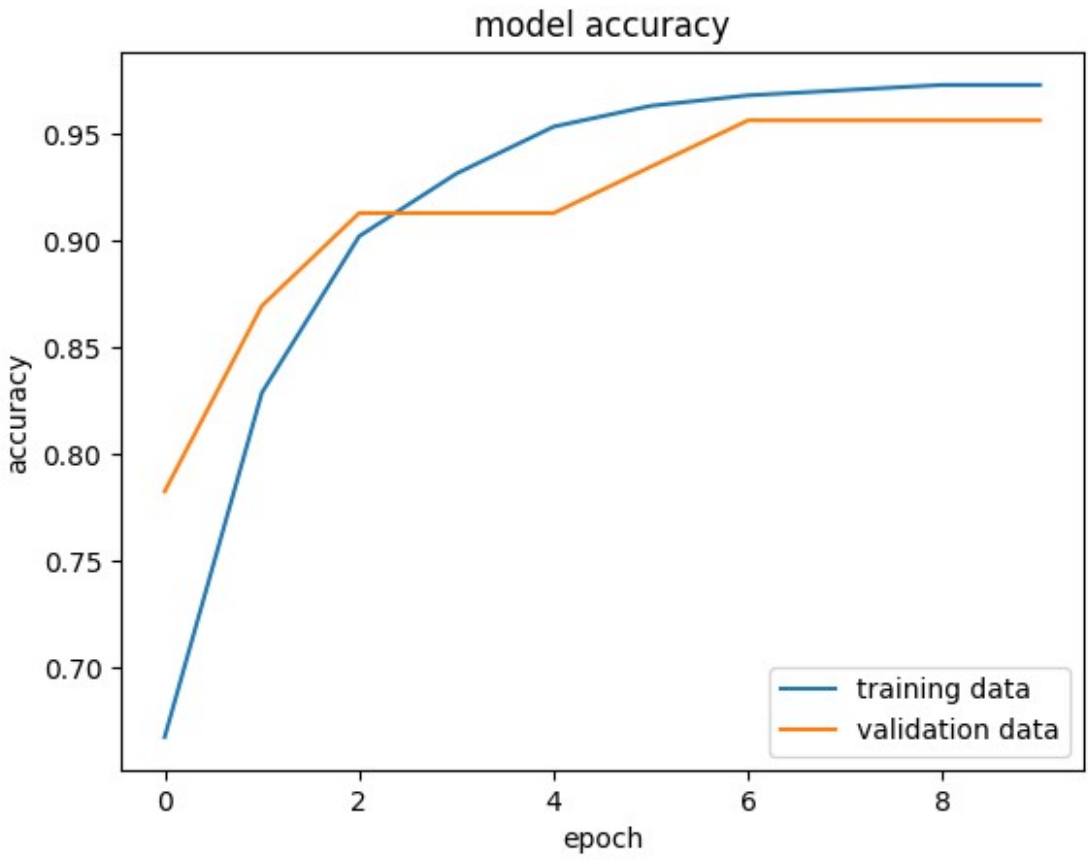
```
history = model.fit(X_train_std, Y_train, validation_split=0.1,
epochs=10)

Epoch 1/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.6437 - loss:
0.6799 - val_accuracy: 0.7826 - val_loss: 0.4845
Epoch 2/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8209 - loss:
0.4703 - val_accuracy: 0.8696 - val_loss: 0.3614
Epoch 3/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8989 - loss:
0.3481 - val_accuracy: 0.9130 - val_loss: 0.2881
Epoch 4/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9270 - loss:
0.2732 - val_accuracy: 0.9130 - val_loss: 0.2405
Epoch 5/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9516 - loss:
0.2232 - val_accuracy: 0.9130 - val_loss: 0.2073
Epoch 6/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9607 - loss:
0.1881 - val_accuracy: 0.9348 - val_loss: 0.1822
Epoch 7/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9662 - loss:
0.1628 - val_accuracy: 0.9565 - val_loss: 0.1620
Epoch 8/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9713 - loss:
0.1443 - val_accuracy: 0.9565 - val_loss: 0.1457
Epoch 9/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9728 - loss:
0.1302 - val_accuracy: 0.9565 - val_loss: 0.1325
Epoch 10/10
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9728 - loss:
0.1191 - val_accuracy: 0.9565 - val_loss: 0.1219
```

Descripción de los Parámetros

- Datos de Entrenamiento (X_train_std, Y_train):
  - X_train_std: Conjunto de características estandarizadas.
  - Y_train: Etiquetas correspondientes a las características.
- validation_split: 0.1: Este parámetro indica que el 10% de los datos de entrenamiento se separará para validación durante el entrenamiento. Esto permite evaluar cómo se está desempeñando el modelo en datos no vistos en cada época.
- epochs: 10: Este parámetro especifica cuántas veces se pasará por todo el conjunto de datos durante el entrenamiento. Un mayor número de épocas puede llevar a un mejor ajuste, pero también puede aumentar el riesgo de sobreajuste si no se controla adecuadamente.

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

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')

plt.legend(['training data', 'validation data'], loc = 'lower right')
```

```
<matplotlib.legend.Legend at 0x7b65dd2f1650>
```

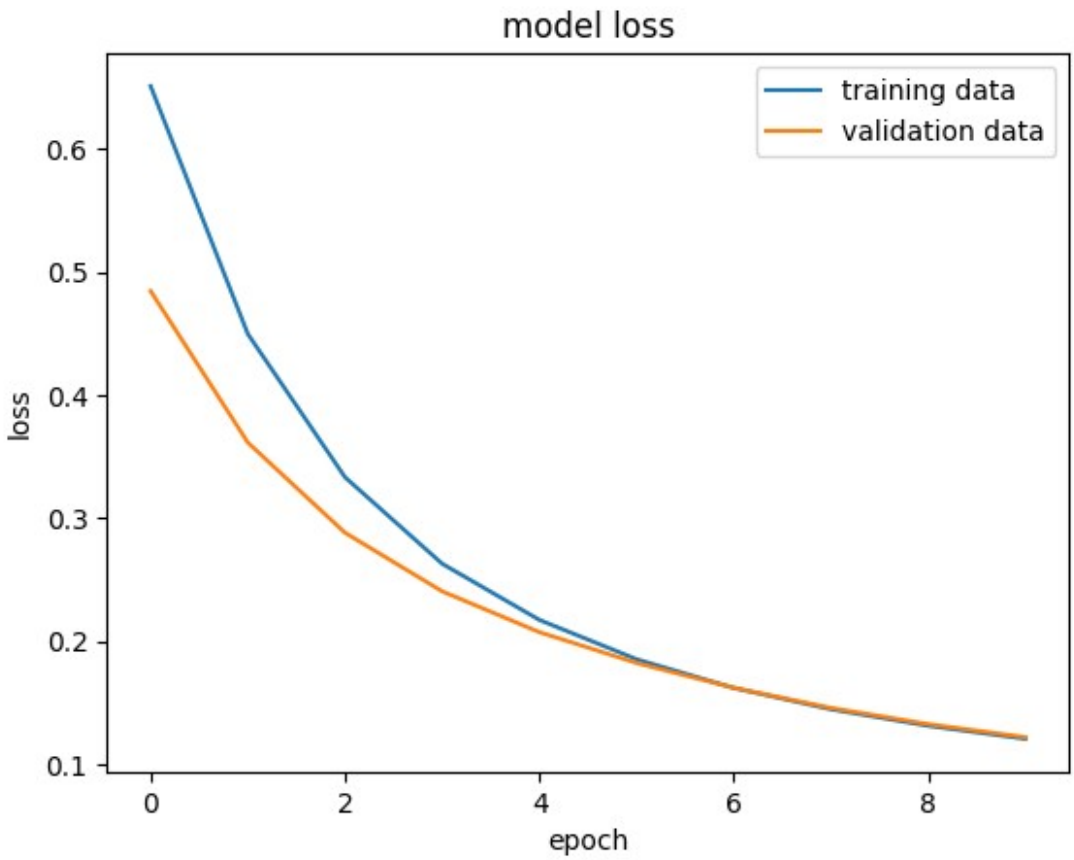


Descripción del Código

- plt.plot(history.history['accuracy']): Esta línea grafica la precisión del conjunto de entrenamiento a lo largo de las épocas. history.history['accuracy'] contiene los valores de precisión registrados durante el entrenamiento.
- plt.plot(history.history['val_accuracy']): Esta línea grafica la precisión del conjunto de validación. history.history['val_accuracy'] contiene los valores de precisión para los datos que no fueron utilizados durante el entrenamiento, permitiendo evaluar cómo se desempeña el modelo en datos no vistos.
- Título y Etiquetas: plt.title('Model Accuracy'): Establece el título del gráfico. plt.ylabel('Accuracy'): Etiqueta el eje Y, que representa la precisión. plt.xlabel('Epoch'): Etiqueta el eje X, que representa el número de épocas.
- Leyenda: plt.legend(['Training Data', 'Validation Data'], loc='lower right'): Agrega una leyenda al gráfico para identificar qué línea corresponde a los datos de entrenamiento y cuál a los datos de validación. La ubicación se establece en la esquina inferior derecha.

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

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')

plt.legend(['training data', 'validation data'], loc = 'upper right')
```

```
<matplotlib.legend.Legend at 0x7b65d6737490>
```




```
loss, accuracy = model.evaluate(X_test_std, Y_test)
print(accuracy)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9641 - loss: 0.1206

0.9649122953414917
```

Descripción del Código

- model.evaluate(X_test_std, Y_test): Este método calcula la pérdida y la precisión del modelo utilizando los datos estandarizados de prueba (X_test_std) y las etiquetas correspondientes (Y_test).
- Salida: La salida muestra tanto la precisión como la pérdida. Por ejemplo, la salida 0.9323 indica que el modelo tiene una precisión del 93.23% en el conjunto de prueba.

```
print(X_test_std.shape)
print(X_test_std[0])
```

```
(114, 30)
[-0.04462793 -1.41612656 -0.05903514 -0.16234067  2.0202457  -
0.11323672
  0.18500609  0.47102419  0.63336386  0.26335737  0.53209124
2.62763999
  0.62351167  0.11405261  1.01246781  0.41126289  0.63848593
```

```
2.88971815
 -0.41675911  0.74270853 -0.32983699 -1.67435595 -0.36854552 -
0.38767294
  0.32655007 -0.74858917 -0.54689089 -0.18278004 -1.23064515 -
0.6268286 ]
```

Salida Esperada Dimensiones: (114, 30) indica que hay 114 muestras y 30 características. Primer Ejemplo: Muestra las características estandarizadas del primer dato en el conjunto.

```
Y_pred = model.predict(X_test_std)

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
```

Descripción del Código model.predict(X_test_std): Este método genera predicciones para cada uno de los ejemplos en X_test_std. La salida es una matriz donde cada fila corresponde a las probabilidades predichas para cada clase.

```
print(Y_pred.shape)
print(Y_pred[0])

(114, 2)
[0.08728971 0.6191301 ]

print(X_test_std)

[[-0.04462793 -1.41612656 -0.05903514 ... -0.18278004 -1.23064515
  -0.6268286 ]
 [ 0.24583601 -0.06219797  0.21802678 ...  0.54129749  0.11047691
   0.0483572 ]
 [-1.26115925 -0.29051645 -1.26499659 ... -1.35138617  0.269338
  -0.28231213]
 ...
 [ 0.72709489  0.45836817  0.75277276 ...  1.46701686  1.19909344
   0.65319961]
 [ 0.25437907  1.33054477  0.15659489 ... -1.29043534 -2.22561725
  -1.59557344]
 [ 0.84100232 -0.06676434  0.8929529  ...  2.15137705  0.35629355
   0.37459546]]

print(X_test_std)

[[-0.04462793 -1.41612656 -0.05903514 ... -0.18278004 -1.23064515
  -0.6268286 ]
 [ 0.24583601 -0.06219797  0.21802678 ...  0.54129749  0.11047691
   0.0483572 ]
 [-1.26115925 -0.29051645 -1.26499659 ... -1.35138617  0.269338
  -0.28231213]
 ...
 [ 0.72709489  0.45836817  0.75277276 ...  1.46701686  1.19909344
   0.65319961]
```

```
 [ 0.25437907  1.33054477  0.15659489 ... -1.29043534 -2.22561725
  -1.59557344]
 [ 0.84100232 -0.06676434  0.8929529  ...  2.15137705  0.35629355
   0.37459546]]

my_list = [0.25, 0.56]

index_of_max_value = np.argmax(my_list)
print(my_list)
print(index_of_max_value)

[0.25, 0.56]
1
```

Se puede determinar la clase predicha a partir de las probabilidades utilizando np.argmax, que devuelve el índice del valor máximo:

```
Y_pred_labels = [np.argmax(i) for i in Y_pred]
print(Y_pred_labels)

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

Para predecir si un nuevo tumor es benigno o maligno, primero se debe estandarizar la entrada:

```
input_data =
(11.76,21.6,74.72,427.9,0.08637,0.04966,0.01657,0.01115,0.1495,0.05888
,0.4062,1.21,2.635,28.47,0.005857,0.009758,0.01168,0.007445,0.02406,0.
001769,12.98,25.72,82.98,516.5,0.1085,0.08615,0.05523,0.03715,0.2433,0
.06563)

# change the input_data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the numpy array as we are predicting for one data point
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardizing the input data
input_data_std = scaler.transform(input_data_reshaped)

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

prediction_label = [np.argmax(prediction)]
print(prediction_label)

if(prediction_label[0] == 0):
```

```
  print('The tumor is Malignant')

else:
  print('The tumor is Benign')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
[[0.14747918 0.9072381 ]]
[1]
The tumor is Benign

/usr/local/lib/python3.11/dist-packages/sklearn/utils/
validation.py:2739: UserWarning: X does not have valid feature names,
but StandardScaler was fitted with feature names
  warnings.warn(
```

La evaluación y predicción con un modelo de red neuronal son procesos esenciales para determinar su rendimiento y aplicabilidad en problemas reales como la clasificación médica. A través de métricas como la precisión y funciones como predict, se puede obtener una visión clara sobre cómo el modelo está funcionando y cómo se pueden interpretar sus resultados para tomar decisiones informadas sobre diagnósticos o tratamientos en contextos clínicos.

Bosques Aleatorios (Random Forests)

- Descripción: Los árboles de decisión son modelos simples que dividen los datos en función de características específicas. Los bosques aleatorios combinan múltiples árboles de decisión para mejorar la precisión y reducir el riesgo de sobreajuste.
- Ventajas: Son fáciles de interpretar y pueden manejar tanto variables categóricas como continuas. Además, los bosques aleatorios son robustos ante el sobreajuste.
- Implementación: Puedes usar RandomForestClassifier de Scikit-Learn.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2, random_state=42)

# Crear el modelo de Bosques Aleatorios
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Entrenar el modelo
model.fit(X_train, Y_train)

RandomForestClassifier(random_state=42)

# Realizar predicciones sobre el conjunto de prueba
Y_pred = model.predict(X_test)

# Calcular la precisión del modelo
```

```python
accuracy = accuracy_score(Y_test, Y_pred)
print(f'Accuracy: {accuracy:.4f}')

# Mostrar un informe de clasificación
print(classification_report(Y_test, Y_pred))

# Mostrar la matriz de confusión
conf_matrix = confusion_matrix(Y_test, Y_pred)
print('Confusion Matrix:')
print(conf_matrix)
```

```
Accuracy: 0.9649
              precision    recall  f1-score   support

           0       0.98      0.93      0.95        43
           1       0.96      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114

Confusion Matrix:
[[40  3]
 [ 1 70]]
```

- Accuracy (Precisión Global): Valor: 0.9649 (o 96.49%) Interpretación: Este valor indica que el modelo clasificó correctamente aproximadamente el 96.49% de las instancias en el conjunto de prueba. Esto es un excelente indicador del rendimiento general del modelo.
- Precision (Precisión): Clase 0: 0.98 Clase 1: 0.96 Interpretación: La precisión mide la proporción de verdaderos positivos entre todos los ejemplos que fueron clasificados como positivos por el modelo. Para la clase 0 (por ejemplo, benigno), el modelo tiene una precisión del 98%, lo que significa que el 98% de las instancias clasificadas como benignas son realmente benignas. Para la clase 1 (por ejemplo, maligno), la precisión es del 96%, indicando que el 96% de las instancias clasificadas como malignas son realmente malignas.
- Recall (Sensibilidad o Tasa de Verdaderos Positivos): Clase 0: 0.93 Clase 1: 0.99 Interpretación: El recall mide la proporción de verdaderos positivos entre todos los ejemplos reales positivos. Para la clase 0, el recall es del 93%, lo que significa que el modelo detecta correctamente el 93% de los casos reales de esta clase. Para la clase 1, el recall es del 99%, lo que indica que casi todos los casos reales de esta clase son detectados correctamente.
- F1-Score: Clase 0: 0.95 Clase 1: 0.97 Interpretación: El F1-score es la media armónica entre precisión y recall, proporcionando un balance entre ambas métricas. Un F1-score alto para ambas clases indica un buen equilibrio entre la capacidad del modelo para identificar correctamente las instancias positivas y minimizar los falsos positivos.
- Support (Soporte): Clase 0: 43 Clase 1: 71 Interpretación: El soporte indica el número de instancias reales en cada clase en el conjunto de prueba. Hay más instancias de la clase 1 en este caso.

- Promedios (Macro y Weighted): Macro Avg: Precision: 0.97 Recall: 0.96 F1-Score: 0.96 Interpretación: Promedia las métricas sin tener en cuenta el soporte, lo que proporciona una visión equitativa del rendimiento del modelo en ambas clases. Weighted Avg: Precision: 0.97 Recall: 0.96 F1-Score: 0.96 Interpretación: Promedia las métricas teniendo en cuenta el soporte, proporcionando una visión general del rendimiento considerando la distribución real de clases.
- TP = 70: El modelo predijo correctamente que había 70 instancias de la clase positiva (por ejemplo, maligno).
- TN = 40: El modelo predijo correctamente que había 40 instancias de la clase negativa (por ejemplo, benigno).
- FP = 3: El modelo predijo incorrectamente que había 3 instancias de la clase positiva cuando en realidad eran negativas.
- FN = 1: El modelo no detectó correctamente 1 instancia de la clase positiva, clasificándola como negativa.

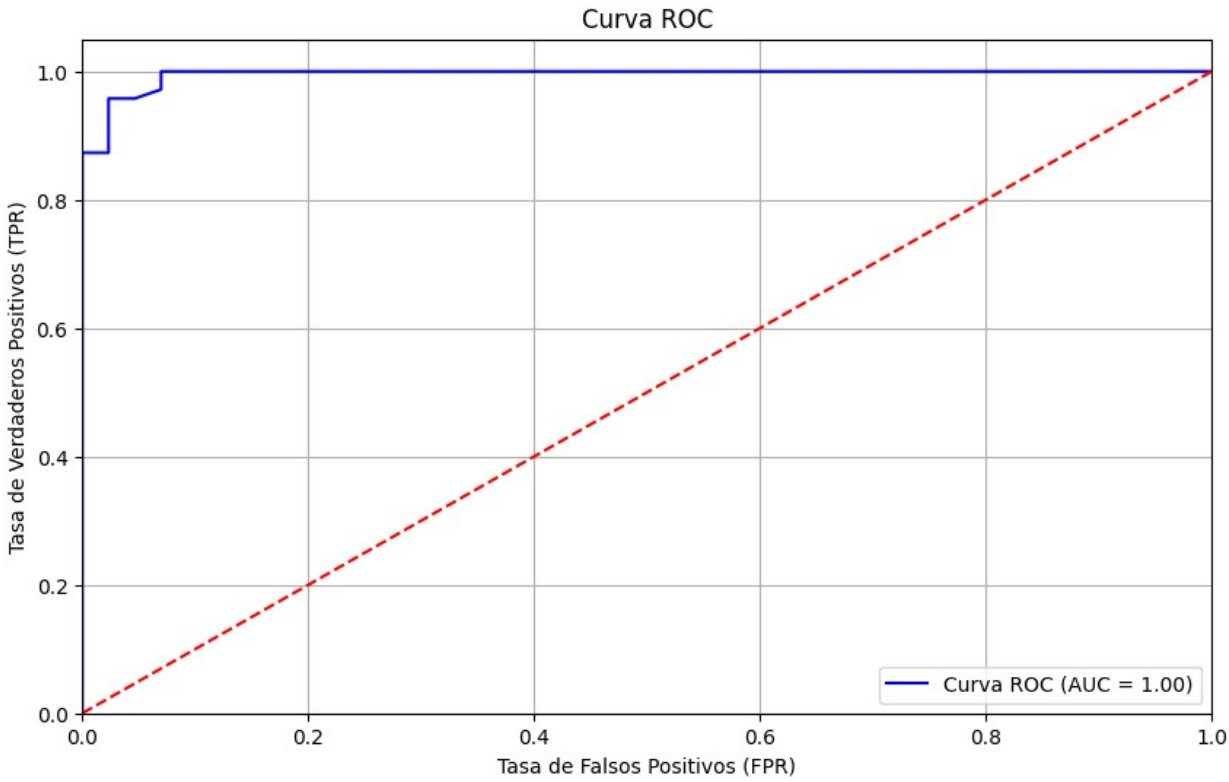
```
 #               Predicción
##              |  Positivo  |  Negativo
#---------------------------------------
#Real Positivo  |     TP     |     FN
#---------------------------------------
#Real Negativo  |     FP     |     TN


input_data =
(13.30,14.50,88.00,562.5,0.09750,0.08200,0.06750,0.04820,0.1890,0.0585
0,0.2720,0.7950,2.070,23.80,0.008550,0.01500,0.02420,0.01350,0.02010,0
.00260,15.25,19.50,100.20,708.5,0.1450,0.1780,0.2410,0.1305,0.2985,0.0
7350)

# change the input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the numpy array as we are predicting for one datapoint
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

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

if (prediction[0] == 0):
  print('The Breast cancer is Malignant')

else:
  print('The Breast Cancer is Benign')
```

```
[1]
The Breast Cancer is Benign

/usr/local/lib/python3.11/dist-packages/sklearn/utils/
validation.py:2739: UserWarning: X does not have valid feature names,
```

```
but RandomForestClassifier was fitted with feature names
  warnings.warn(
```

ROC

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Supongamos que 'model' es tu clasificador ya entrenado y 'X_test'
son tus datos de prueba
Y_pred_prob = model.predict_proba(X_test)[:, 1]  # Obtener las
probabilidades para la clase positiva

# Obtener los valores FPR y TPR
fpr, tpr, thresholds = roc_curve(Y_test, Y_pred_prob)

roc_auc = auc(fpr, tpr)

plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='blue', label='Curva ROC (AUC =
{:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='red', linestyle='--')  # Línea
diagonal aleatoria
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Tasa de Falsos Positivos (FPR)')
plt.ylabel('Tasa de Verdaderos Positivos (TPR)')
plt.title('Curva ROC')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```



Máquinas de Soporte Vectorial (SVM)

- Descripción: Las SVM son modelos que buscan encontrar el hiperplano óptimo que separa las diferentes clases en el espacio de características.
- Ventajas: Son efectivas en espacios de alta dimensión y son robustas frente a la sobreajuste, especialmente con un buen kernel.
- Implementación: Utiliza SVC de Scikit-Learn y experimenta con diferentes kernels (lineal, polinómico, RBF).

```
from sklearn.svm import SVC

# Dividir los datos en conjuntos de entrenamiento y prueba (80% entrenamiento, 20% prueba)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

# Crear el modelo SVM con un kernel lineal
model = SVC(kernel='linear', random_state=42)

# Entrenar el modelo
model.fit(X_train, Y_train)
```

```
SVC(kernel='linear', random_state=42)
```

```python
# Realizar predicciones sobre el conjunto de prueba
Y_pred = model.predict(X_test)

# Calcular la precisión del modelo
accuracy = accuracy_score(Y_test, Y_pred)
print(f'Accuracy: {accuracy:.4f}')

# Mostrar un informe de clasificación
print(classification_report(Y_test, Y_pred))

# Mostrar la matriz de confusión
conf_matrix = confusion_matrix(Y_test, Y_pred)
print('Confusion Matrix:')
print(conf_matrix)
```

```
Accuracy: 0.9561
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        43
           1       0.95      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114

Confusion Matrix:
[[39  4]
 [ 1 70]]
```

- Accuracy (Precisión Global): Valor: 0.9561 (o 95.61%) Interpretación: Este valor indica que el modelo clasificó correctamente aproximadamente el 95.61% de las instancias en el conjunto de prueba. Es un buen indicador general del rendimiento del modelo.
- Precision (Precisión): Clase 0: 0.97 Clase 1: 0.95 Interpretación: La precisión mide la proporción de verdaderos positivos entre todos los ejemplos que fueron clasificados como positivos por el modelo. Para la clase 0 (por ejemplo, benigno), el modelo tiene una precisión del 97%, lo que significa que el 97% de las instancias clasificadas como benignas son realmente benignas. Para la clase 1 (por ejemplo, maligno), la precisión es del 95%, indicando que el 95% de las instancias clasificadas como malignas son realmente malignas.
- Recall (Sensibilidad o Tasa de Verdaderos Positivos): Clase 0: 0.91 Clase 1: 0.99 Interpretación: El recall mide la proporción de verdaderos positivos entre todos los ejemplos reales positivos. Para la clase 0, el recall es del 91%, lo que significa que el modelo detecta correctamente el 91% de los casos reales de esta clase. Para la clase 1, el recall es del 99%, lo que indica que casi todos los casos reales de esta clase son detectados correctamente.
- F1-Score: Clase 0: 0.94 Clase 1: 0.97 Interpretación: El F1-score es la media armónica entre precisión y recall, proporcionando un balance entre ambas métricas. Un F1-score alto para ambas clases indica un buen equilibrio entre la capacidad del modelo para identificar correctamente las instancias positivas y minimizar los falsos positivos.

- Support (Soporte): Clase 0: 43 Clase 1: 71 Interpretación: El soporte indica el número de instancias reales en cada clase en el conjunto de prueba. Hay más instancias de la clase 1 en este caso.
- Promedios (Macro y Weighted): Macro Avg: Precision: 0.96 Recall: 0.95 F1-Score: 0.95 Interpretación: Promedia las métricas sin tener en cuenta el soporte, lo que proporciona una visión equitativa del rendimiento del modelo en ambas clases. Weighted Avg: Precision: 0.96 Recall: 0.96 F1-Score: 0.96 Interpretación: Promedia las métricas teniendo en cuenta el soporte, proporcionando una visión general del rendimiento considerando la distribución real de clases.

```
input_data =
(13.30,14.50,88.00,562.5,0.09750,0.08200,0.06750,0.04820,0.1890,0.0585
0,0.2720,0.7950,2.070,23.80,0.008550,0.01500,0.02420,0.01350,0.02010,0
.00260,15.25,19.50,100.20,708.5,0.1450,0.1780,0.2410,0.1305,0.2985,0.0
7350)

# change the input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the numpy array as we are predicting for one datapoint
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

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

if (prediction[0] == 0):
  print('The Breast cancer is Malignant')

else:
  print('The Breast Cancer is Benign')
```

```
[1]
The Breast Cancer is Benign

/usr/local/lib/python3.11/dist-packages/sklearn/utils/
validation.py:2739: UserWarning: X does not have valid feature names,
but SVC was fitted with feature names
  warnings.warn(
```