Algoritmo Support Vector Machine

El algoritmo Support Vector Machine (SVM) es un método de aprendizaje supervisado utilizado en problemas clasificación y regresión. SVM busca encontrar el hiperplano que mejor separa los datos de diferentes clases en un espacio de alta dimensión. El funcionamiento de SVM se basa en la idea de encontrar el hiperplano que maximiza la distancia entre los puntos de datos de diferentes clases, también conocido como el margen. Los puntos de datos que se encuentran más cerca del hiperplano se denominan vectores de soporte, de ahí el nombre del algoritmo.

Las características principales del algoritmo SVM son: SVM es un algoritmo de aprendizaje supervisado que se utiliza para la clasificación y regresión de datos. SVM busca encontrar el hiperplano que mejor separa los datos en diferentes clases. El hiperplano es elegido de tal manera que maximiza la distancia entre los puntos de datos más cercanos de cada clase, lo que se conoce como margen. SVM es un algoritmo muy efectivo para clasificar datos en conjuntos de alta dimensionalidad. SVM es capaz de manejar datos no lineales utilizando diferentes funciones de kernel.

A continuación, se presenta una tabla que resume las fortalezas y debilidades de SVM:

Fortalezas:

1. Efectivo en conjuntos de datos de alta dimensionalidad
2. Capaz de manejar datos no lineales utilizando diferentes funciones de kernel
3. Puede manejar múltiples clases de datos
4. SVM tiene una buena capacidad de generalización
5. SVM es resistente a los valores atípicos

Debilidades:

1. No es adecuado para conjuntos de datos muy grandes
2. Sensible a la selección de parámetros
3. Requiere un preprocesamiento cuidadoso de los datos
4. Puede ser computacionalmente costoso entrenar el modelo
5. No proporciona una probabilidad de clasificación directa

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Leer el archivo CSV
df=pd.read_csv('D:/dataset_t3_1.csv')

print(df.columns)

# Eliminar variables no informativas
df = df.drop(['Image'], axis=1)

# Comprobar si hay valores faltantes
if df.isnull().values.any():
    # Calcular el porcentaje de valores faltantes en cada variable
```

```
    missing_percentages = df.isnull().sum() / len(df) * 100
    print("Porcentaje de valores faltantes en cada variable:")
    print(missing_percentages)

    # Eliminar variables con porcentaje de valores faltantes igual o mayor al 20
    df = df.dropna(thresh=len(df)*0.8, axis=1)
    print("Variables eliminadas:")
    print(list(set(missing_percentages[missing_percentages >= 20].index)))

    # Eliminar imágenes con valores faltantes
    df = df.dropna()
```

```
Index(['Image', 'Mean', 'Variance', 'Standard.Deviation', 'Entropy',
       'Skewness', 'Kurtosis', 'Contrast', 'Energy', 'ASM', 'Homogeneity',
       'Dissimilarity', 'Correlation', 'Coarseness', 'PSNR', 'SSIM', 'MSE',
       'DC', 'clase'],
      dtype='object')
Porcentaje de valores faltantes en cada variable:
Mean                  0.000000
Variance              0.000000
Standard.Deviation    0.000000
Entropy               0.000000
Skewness             22.445255
Kurtosis             22.445255
Contrast              0.000000
Energy                0.000000
ASM                   0.000000
Homogeneity           0.000000
Dissimilarity         0.000000
Correlation           0.000000
Coarseness            0.000000
PSNR                  0.000000
SSIM                 22.445255
MSE                   0.000000
DC                    5.961071
clase                 0.000000
dtype: float64
Variables eliminadas:
['Kurtosis', 'SSIM', 'Skewness']
```

In [ ]:

```
# Obtener estadísticas descriptivas de las variables numéricas
print(df.describe())
# Obtener información sobre el tipo de datos y la cantidad de valores no nulos e
print(df.info())
# Calcular la matriz de correlación entre las variables numéricas
print(df.corr())
# Crear histogramas de las variables numéricas
df.hist(bins=10, figsize=(20,15))
plt.show()
```

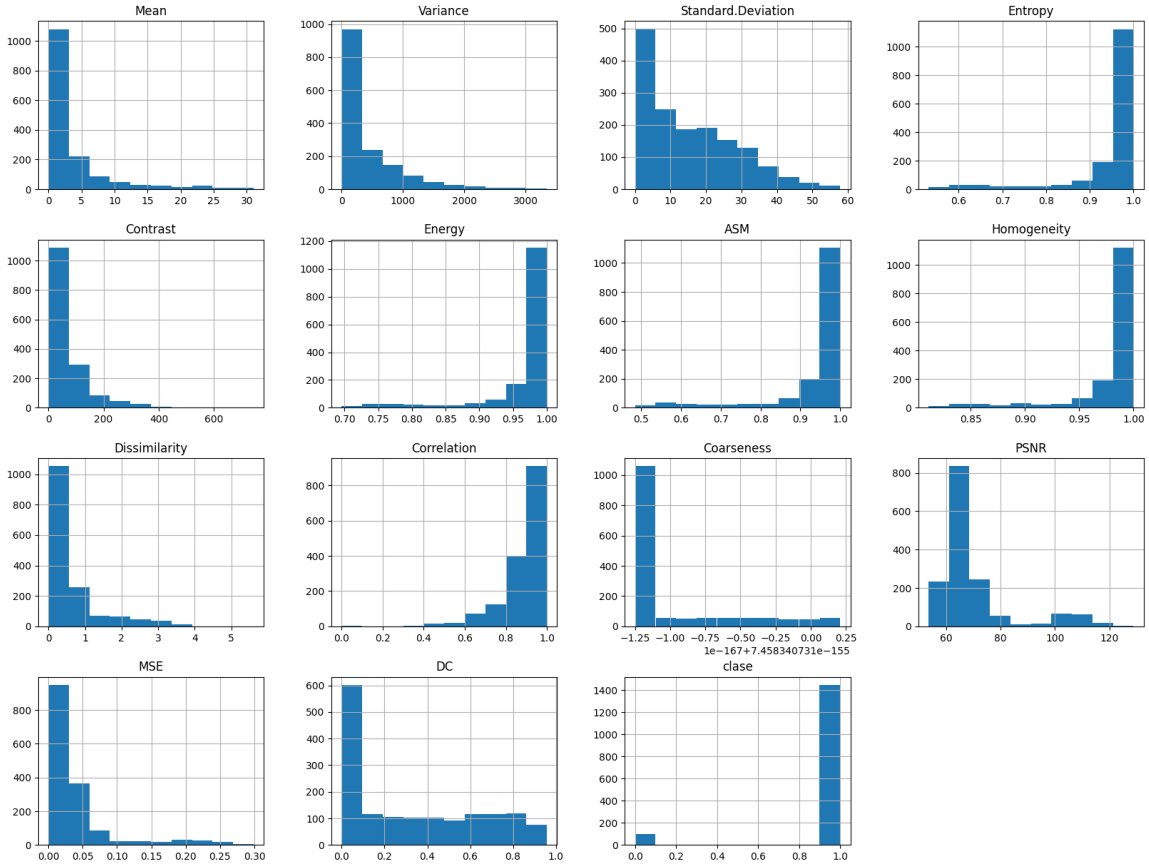
```
              Mean     Variance  Standard.Deviation      Entropy     Contrast
\
count  1546.000000  1546.000000         1546.000000  1546.000000  1546.000000
mean      3.323295   401.125634           15.106902     0.945917    61.895655
std       5.466482   556.738341           13.153671     0.096214    77.290750
min       0.000000     0.000000            0.000000     0.530684     0.000000
25%       0.082703    13.463382            3.669234     0.946582     7.890717
50%       1.027420   156.890324           12.525568     0.985381    36.877493
75%       3.944096   584.191571           24.170047     0.998680    83.837169
max      31.031021  3345.853590           57.843354     1.000000   743.368131

            Energy          ASM  Homogeneity  Dissimilarity  Correlation  \
count  1546.000000  1546.000000  1546.000000    1546.000000  1546.000000
mean      0.967014     0.938762     0.978442       0.580709     0.888728
std       0.060400     0.107307     0.037638       0.791625     0.116067
min       0.695889     0.484262     0.811102       0.000000    -0.000035
25%       0.968655     0.938293     0.978690       0.053801     0.850080
50%       0.991484     0.983040     0.993697       0.289592     0.919154
75%       0.999233     0.998466     0.999423       0.713511     0.963400
max       1.000000     1.000000     1.000000       5.604428     1.000000

          Coarseness         PSNR          MSE           DC        clase
count   1.546000e+03  1546.000000  1546.000000  1546.000000  1546.000000
mean   7.458341e-155    69.788729     0.038760     0.320465     0.937257
std     0.000000e+00    14.180364     0.051930     0.312112     0.242578
min    7.458341e-155    53.378482     0.000017     0.000000     0.000000
25%    7.458341e-155    62.231101     0.009705     0.000000     1.000000
50%    7.458341e-155    65.402806     0.022387     0.247692     1.000000
75%    7.458341e-155    70.044243     0.043016     0.599091     1.000000
max    7.458341e-155   128.882059     0.298698     0.957969     1.000000
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1546 entries, 0 to 1643
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Mean                1546 non-null   float64
 1   Variance            1546 non-null   float64
 2   Standard.Deviation  1546 non-null   float64
 3   Entropy             1546 non-null   float64
 4   Contrast            1546 non-null   float64
 5   Energy              1546 non-null   float64
 6   ASM                 1546 non-null   float64
 7   Homogeneity         1546 non-null   float64
 8   Dissimilarity       1546 non-null   float64
 9   Correlation         1546 non-null   float64
 10  Coarseness          1546 non-null   float64
 11  PSNR                1546 non-null   float64
 12  MSE                 1546 non-null   float64
 13  DC                  1546 non-null   float64
 14  clase               1546 non-null   int64
dtypes: float64(14), int64(1)
memory usage: 193.2 KB
None
                     Mean  Variance  Standard.Deviation   Entropy  \
Mean             1.000000  0.942082            0.855266 -0.962085
Variance         0.942082  1.000000            0.938669 -0.834862
Standard.Deviation  0.855266  0.938669            1.000000 -0.770415
Entropy         -0.962085 -0.834862           -0.770415  1.000000
Contrast         0.716621  0.785865            0.810749 -0.652121
Energy          -0.960364 -0.828102           -0.759355  0.999661
```

| | | | | |
|---|---|---|---|---|
| ASM | -0.962712 | -0.838192 | -0.776129 | 0.999907 |
| Homogeneity | -0.971350 | -0.854092 | -0.785498 | 0.995328 |
| Dissimilarity | 0.858027 | 0.837398 | 0.828607 | -0.841593 |
| Correlation | 0.241558 | 0.271028 | 0.257279 | -0.219657 |
| Coarseness | NaN | NaN | NaN | NaN |
| PSNR | -0.210122 | -0.173540 | -0.179806 | 0.224454 |
| MSE | 0.826517 | 0.648245 | 0.546587 | -0.888544 |
| DC | 0.115751 | 0.303390 | 0.514301 | -0.028391 |
| clase | -0.171909 | -0.091415 | -0.092035 | 0.236299 |

| | Contrast | Energy | ASM | Homogeneity | Dissimilarity | \ |
|---|---|---|---|---|---|---|
| Mean | 0.716621 | -0.960364 | -0.962712 | -0.971350 | 0.858027 | |
| Variance | 0.785865 | -0.828102 | -0.838192 | -0.854092 | 0.837398 | |
| Standard.Deviation | 0.810749 | -0.759355 | -0.776129 | -0.785498 | 0.828607 | |
| Entropy | -0.652121 | 0.999661 | 0.999907 | 0.995328 | -0.841593 | |
| Contrast | 1.000000 | -0.638289 | -0.659149 | -0.674474 | 0.940951 | |
| Energy | -0.638289 | 1.000000 | 0.999214 | 0.994783 | -0.831284 | |
| ASM | -0.659149 | 0.999214 | 1.000000 | 0.995332 | -0.846698 | |
| Homogeneity | -0.674474 | 0.994783 | 0.995332 | 1.000000 | -0.859884 | |
| Dissimilarity | 0.940951 | -0.831284 | -0.846698 | -0.859884 | 1.000000 | |
| Correlation | 0.111366 | -0.216776 | -0.221190 | -0.218906 | 0.151511 | |
| Coarseness | NaN | NaN | NaN | NaN | NaN | |
| PSNR | -0.184408 | 0.224676 | 0.224306 | 0.227021 | -0.214967 | |
| MSE | 0.478701 | -0.895083 | -0.884826 | -0.877213 | 0.669025 | |
| DC | 0.231578 | -0.014334 | -0.036061 | -0.043390 | 0.147468 | |
| clase | -0.139715 | 0.236770 | 0.235975 | 0.211503 | -0.184592 | |

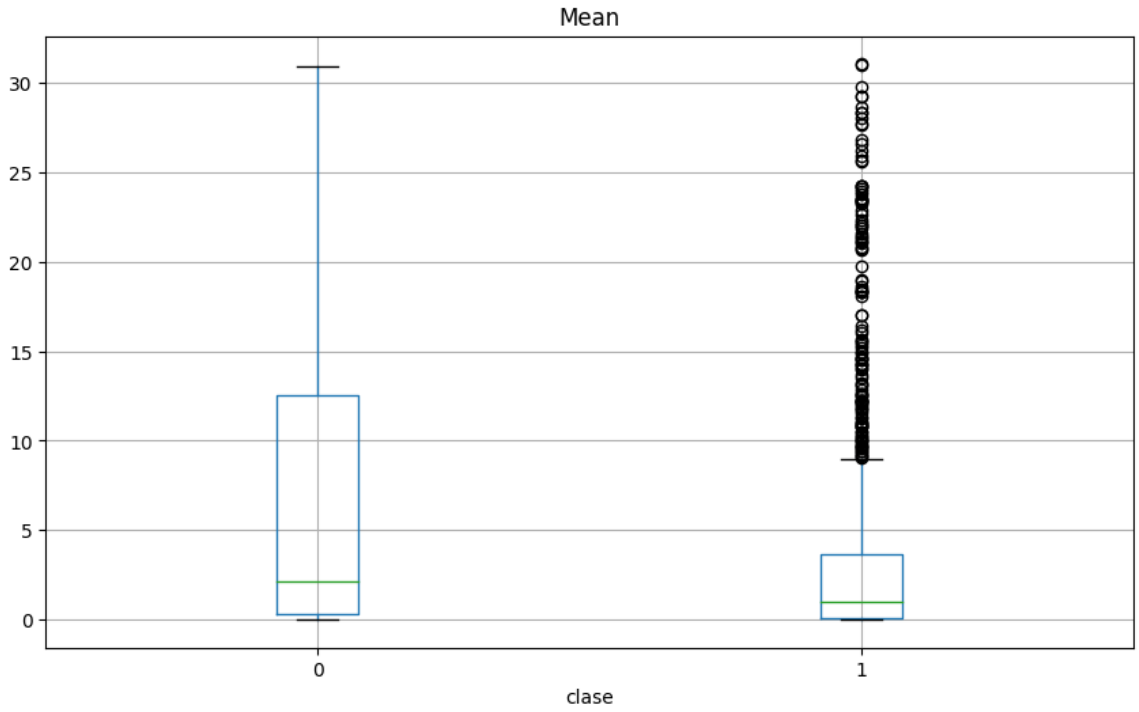
| | Correlation | Coarseness | PSNR | MSE | DC | \ |
|---|---|---|---|---|---|---|
| Mean | 0.241558 | NaN | -0.210122 | 0.826517 | 0.115751 | |
| Variance | 0.271028 | NaN | -0.173540 | 0.648245 | 0.303390 | |
| Standard.Deviation | 0.257279 | NaN | -0.179806 | 0.546587 | 0.514301 | |
| Entropy | -0.219657 | NaN | 0.224454 | -0.888544 | -0.028391 | |
| Contrast | 0.111366 | NaN | -0.184408 | 0.478701 | 0.231578 | |
| Energy | -0.216776 | NaN | 0.224676 | -0.895083 | -0.014334 | |
| ASM | -0.221190 | NaN | 0.224306 | -0.884826 | -0.036061 | |
| Homogeneity | -0.218906 | NaN | 0.227021 | -0.877213 | -0.043390 | |
| Dissimilarity | 0.151511 | NaN | -0.214967 | 0.669025 | 0.147468 | |
| Correlation | 1.000000 | NaN | 0.044581 | 0.105763 | 0.242314 | |
| Coarseness | NaN | NaN | NaN | NaN | NaN | |
| PSNR | 0.044581 | NaN | 1.000000 | -0.288462 | 0.054167 | |
| MSE | 0.105763 | NaN | -0.288462 | 1.000000 | -0.270253 | |
| DC | 0.242314 | NaN | 0.054167 | -0.270253 | 1.000000 | |
| clase | 0.117905 | NaN | -0.066223 | -0.221783 | 0.265744 | |

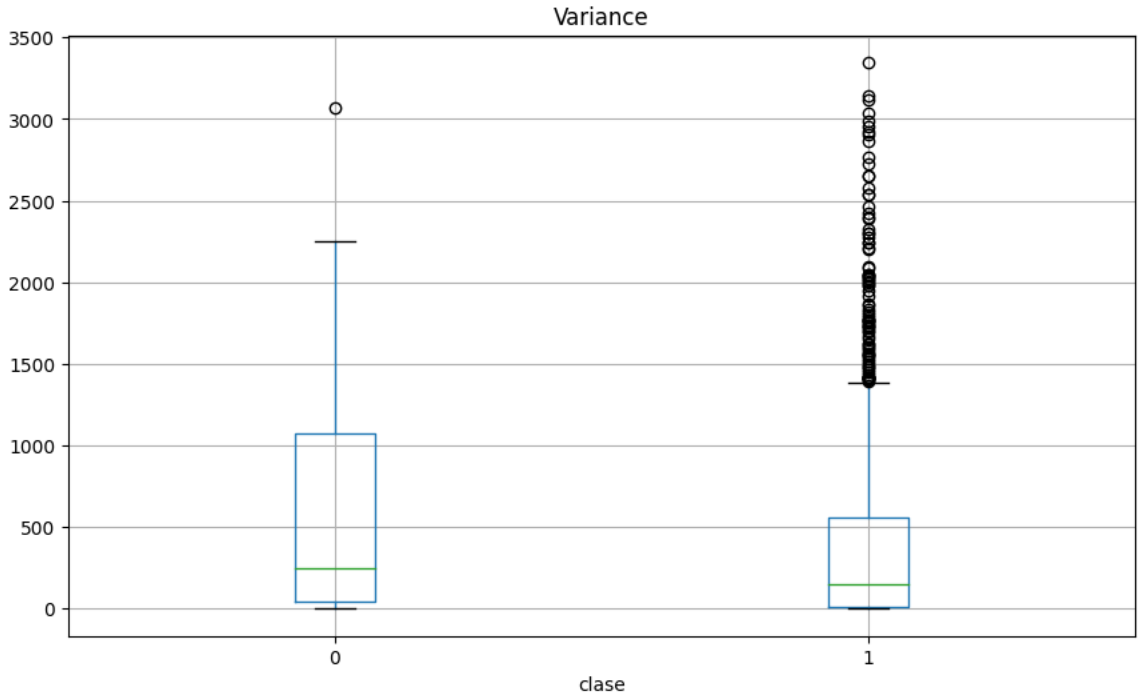
| | clase |
|---|---|
| Mean | -0.171909 |
| Variance | -0.091415 |
| Standard.Deviation | -0.092035 |
| Entropy | 0.236299 |
| Contrast | -0.139715 |
| Energy | 0.236770 |
| ASM | 0.235975 |
| Homogeneity | 0.211503 |
| Dissimilarity | -0.184592 |
| Correlation | 0.117905 |
| Coarseness | NaN |
| PSNR | -0.066223 |
| MSE | -0.221783 |
| DC | 0.265744 |
| clase | 1.000000 |

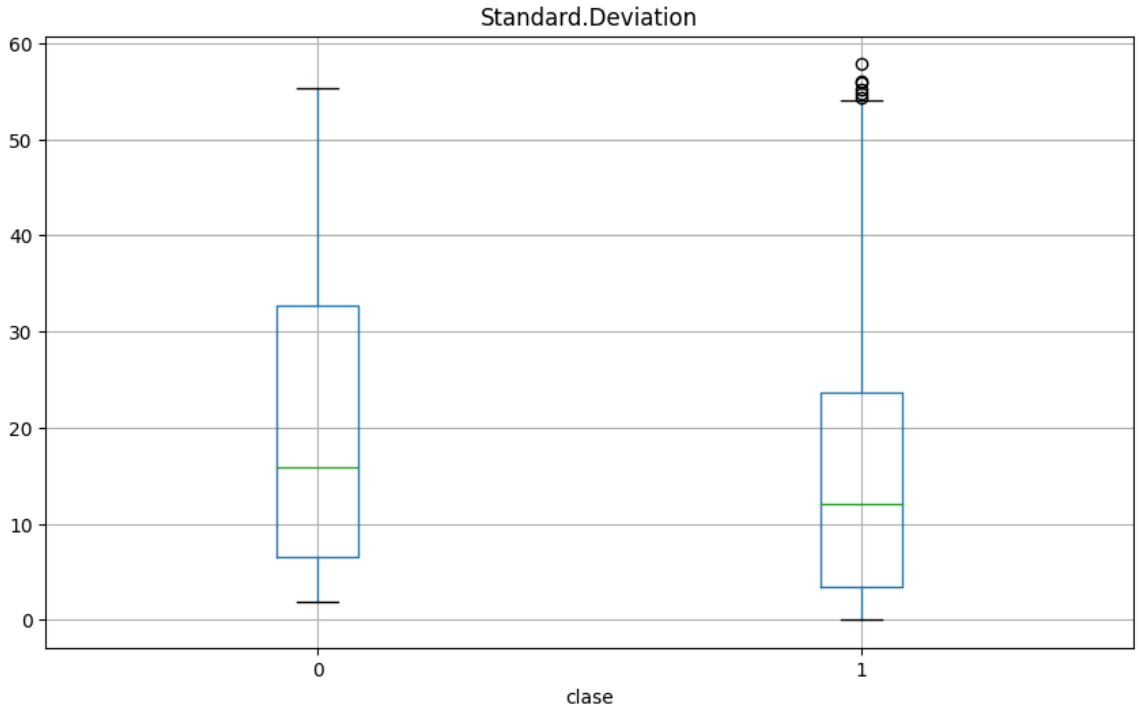


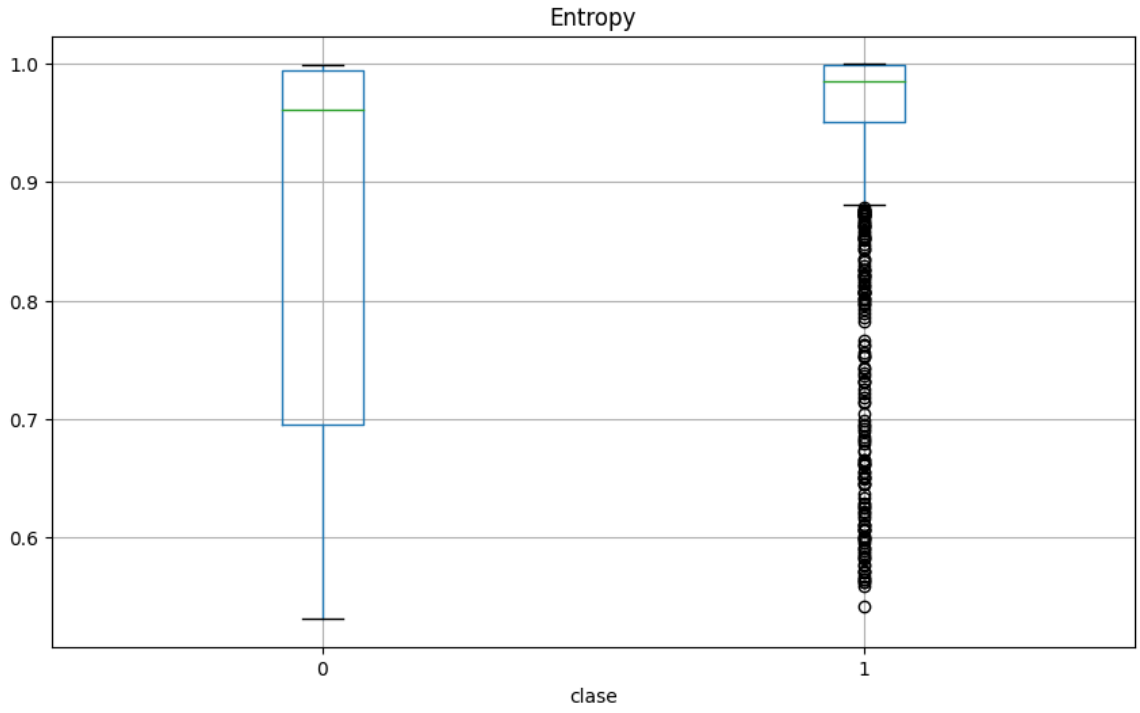

In [ ]:
```
# Agrupar los datos por clase y obtener estadísticas descriptivas de cada variab
grouped = df.groupby('clase')
print(grouped.describe())
# Crear diagramas de caja y bigotes para cada variable numérica, separados por c
for column in df.columns:
    if column != 'clase':
        df.boxplot(column=column, by='clase', figsize=(10,6))
        plt.title(column)
        plt.suptitle('')
        plt.show()
# Obtener el tamaño muestral de cada clase
print(df['clase'].value_counts())
```

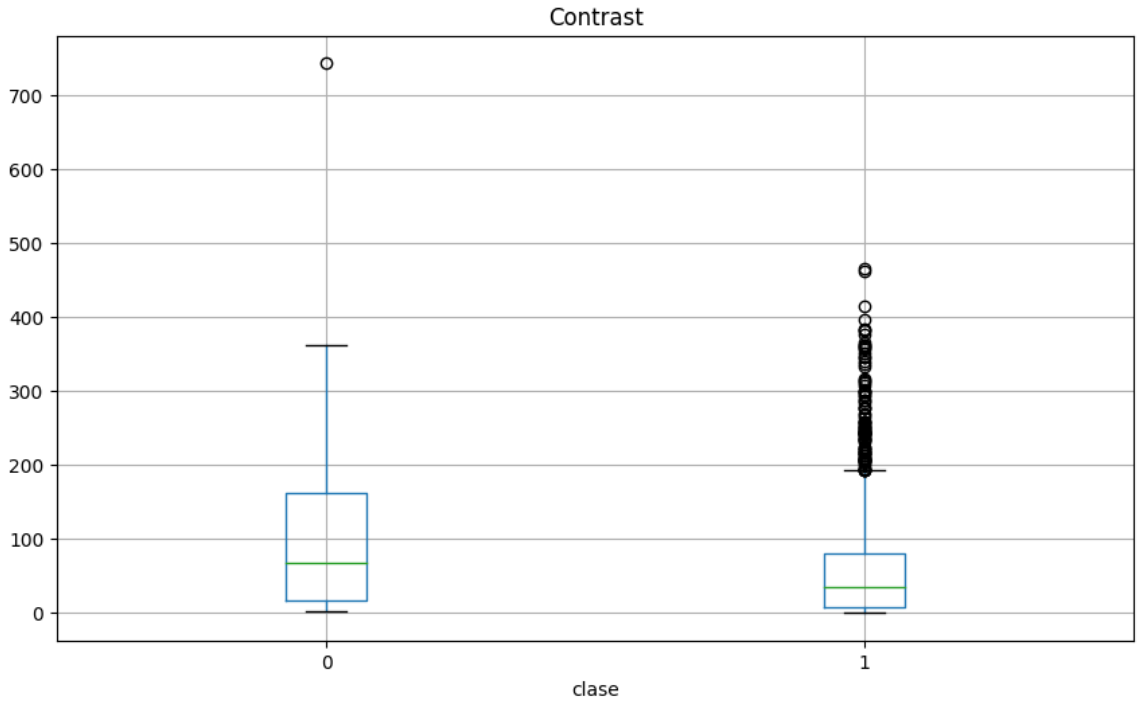
```
         Mean                                                          \
        count       mean       std      min       25%       50%       75%
clase
0        97.0   6.954205  7.793813  0.02652  0.345016  2.106567  12.572342
1      1449.0   3.080232  5.187943  0.00000  0.071823  0.993073   3.649307

                Variance                ...       MSE                DC  \
             max    count        mean   ...       75%       max   count
clase                                   ...
0      30.858765     97.0  597.767226   ...  0.183003  0.298698    97.0
1      31.031021   1449.0  387.961911   ...  0.041233  0.283906  1449.0


           mean       std  min       25%       50%      75%       max
clase
0      0.000000  0.000000  0.0  0.000000  0.000000  0.00000  0.000000
1      0.341918  0.310804  0.0  0.021053  0.285714  0.62089  0.957969

[2 rows x 112 columns]
```

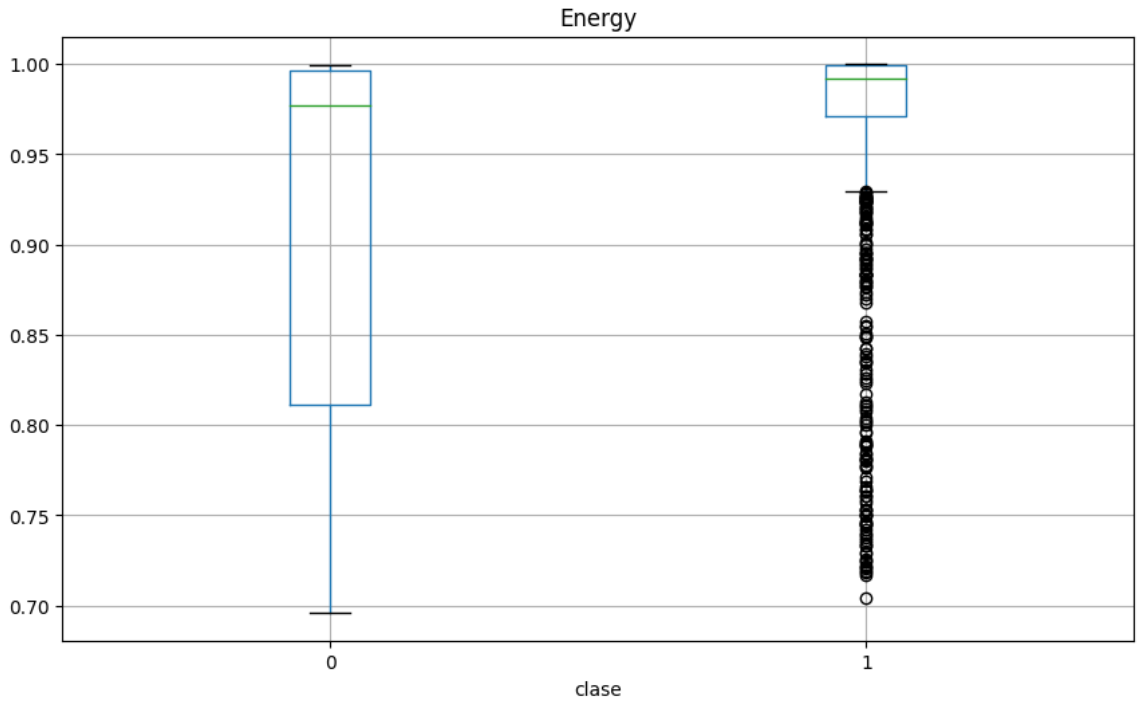

Mean
30
25
20
15
10
5
0
0
1
clase

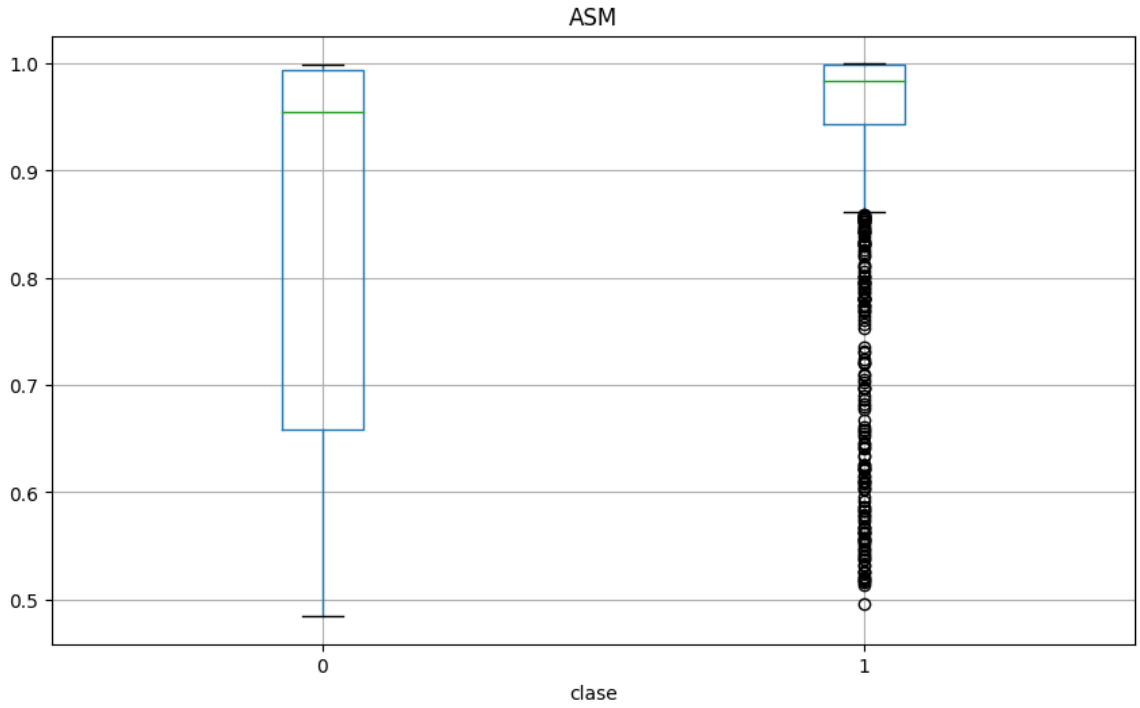

Variance
3500
3000
2500
2000
1500
1000
500
0
0
1
clase

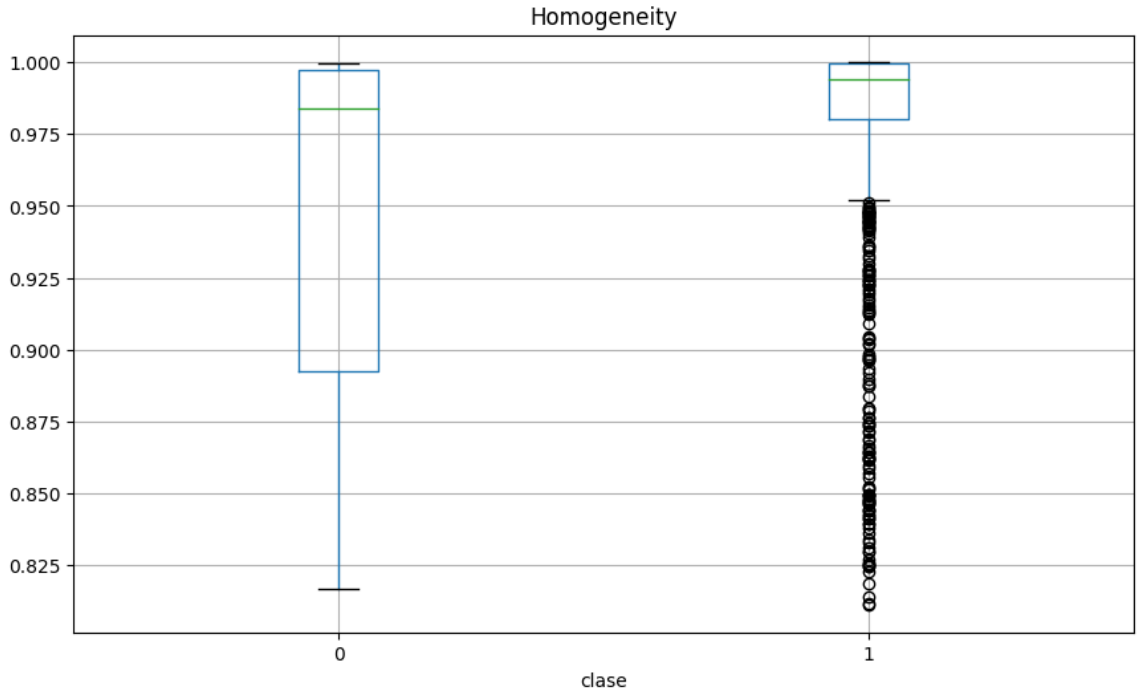

Standard.Deviation
60
50
40
30
20
10
0
0
1
clase

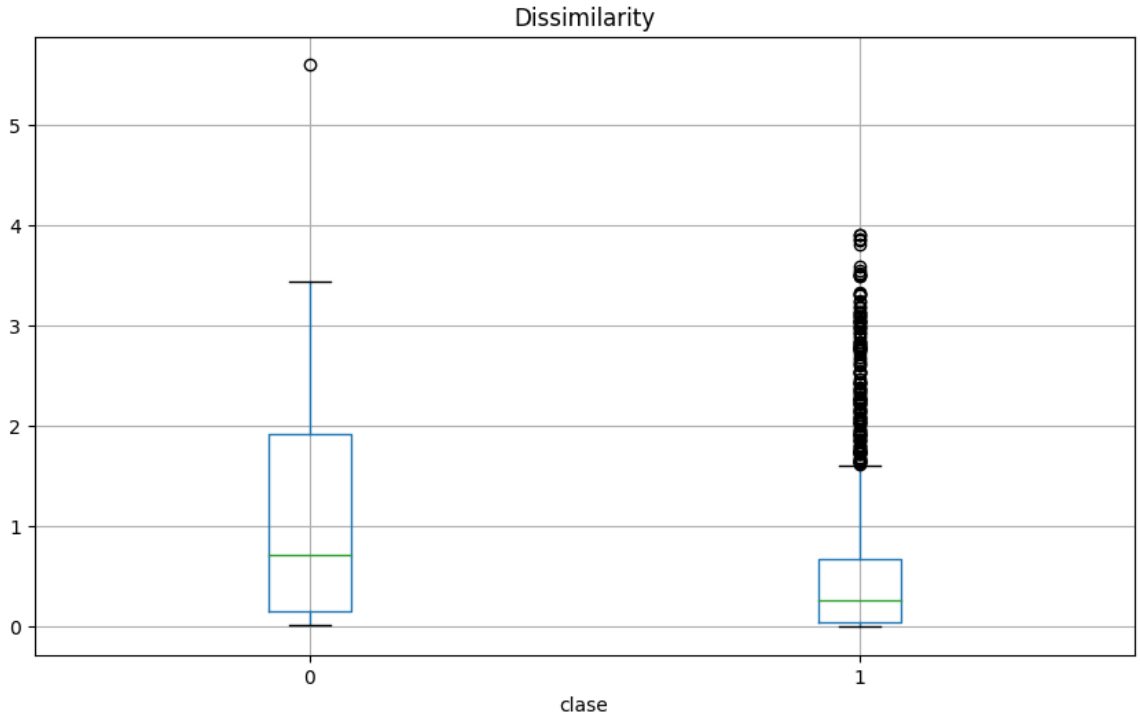

Entropy
1.0
0.9
0.8
0.7
0.6
0
1
clase

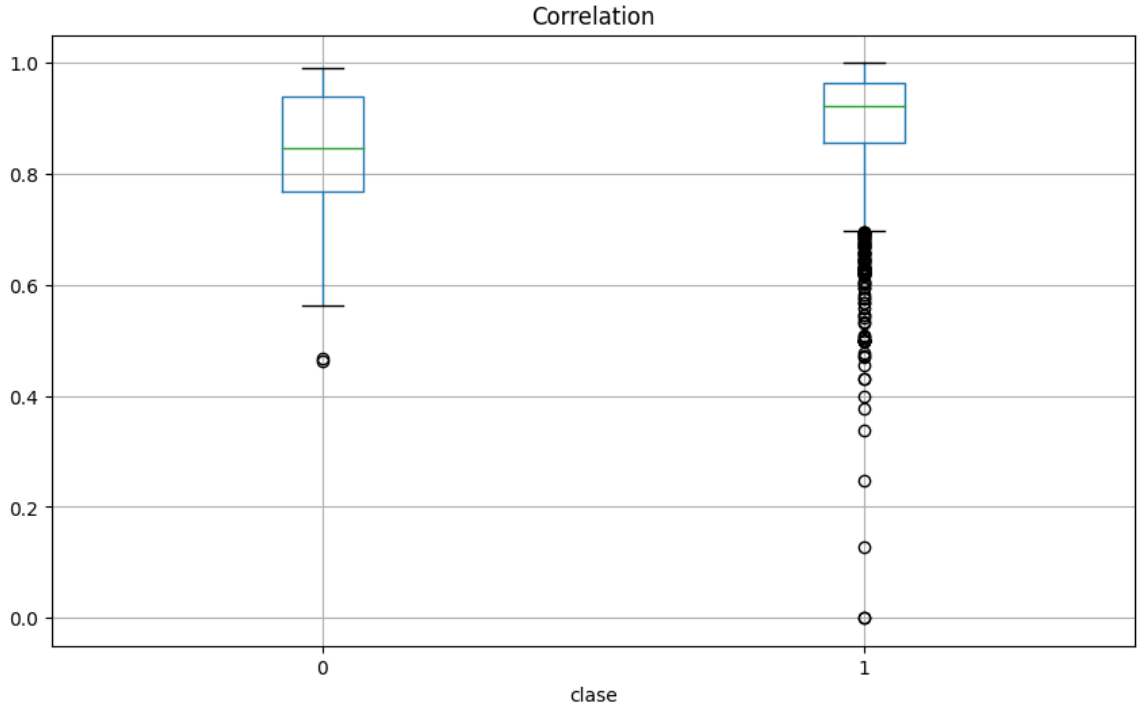

Contrast
700
600
500
400
300
200
100
0
0
1
clase

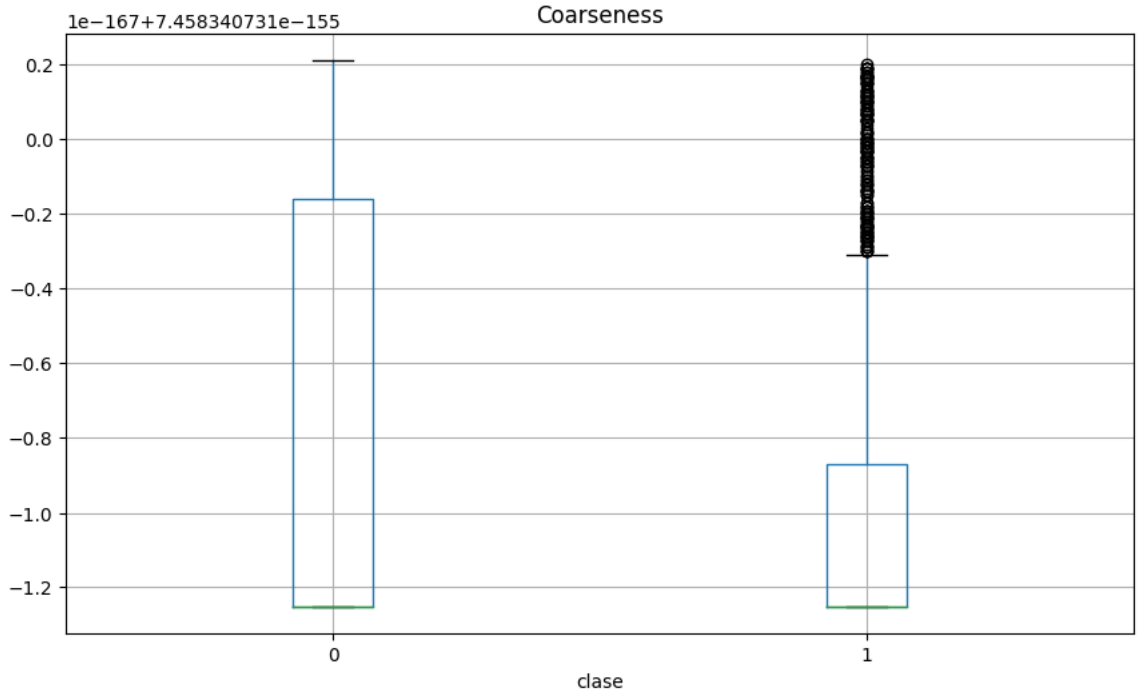

Energy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0
1
clase

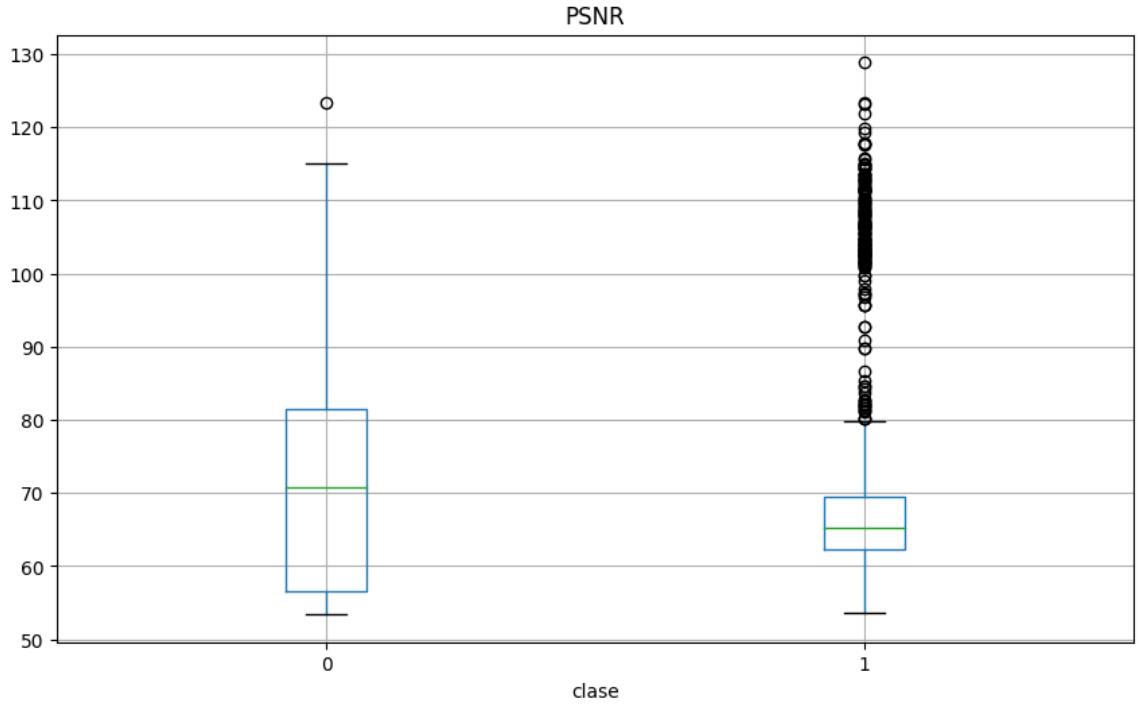

ASM
1.0
0.9
0.8
0.7
0.6
0.5
0
1
clase

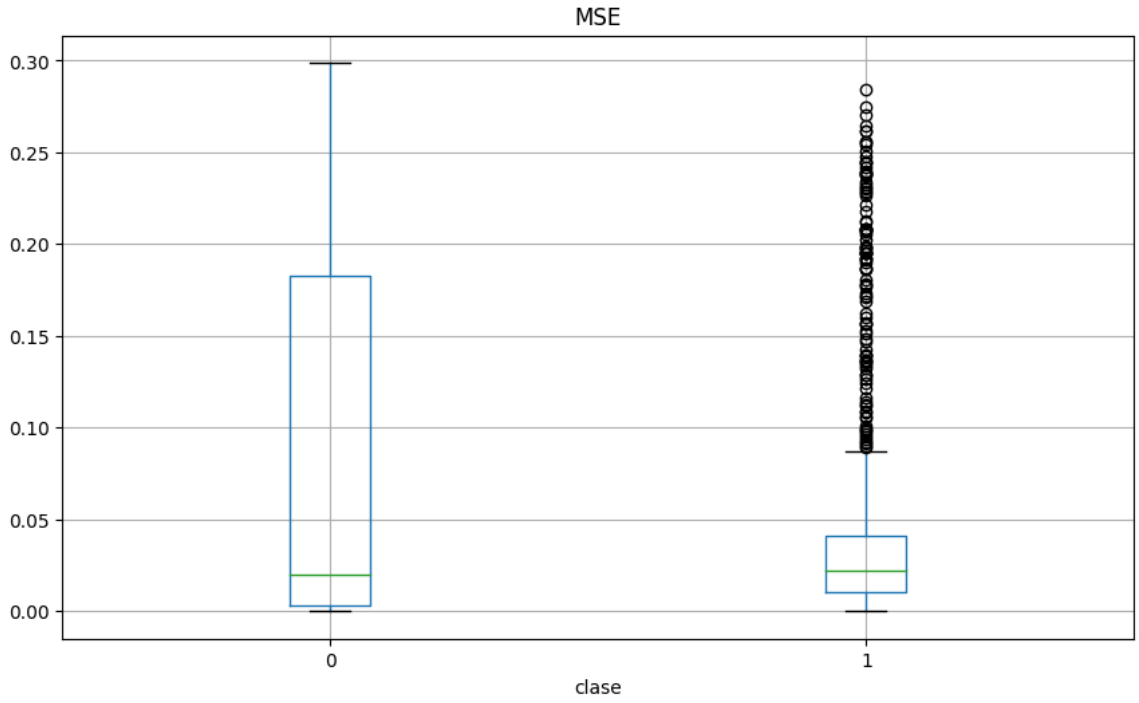

Homogeneity
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0
1
clase

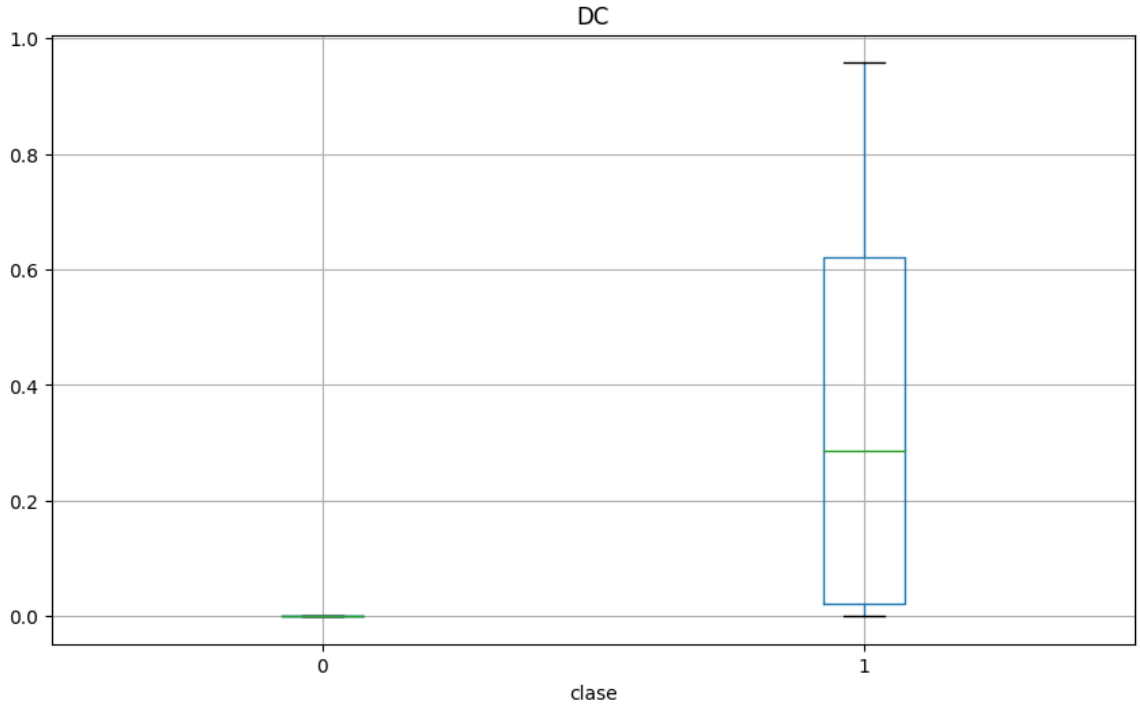

Dissimilarity
5
4
3
2
1
0
0
1
clase


Correlation
1.0
0.8
0.6
0.4
0.2
0.0
0
1
clase


Coarseness
1e−167+7.458340731e−155
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2
0
1
clase


PSNR
130
120
110
100
90
80
70
60
50
0
1
clase




```
1    1449
0      97
Name: clase, dtype: int64
```

In [ ]:
```
from sklearn.model_selection import train_test_split
# Separar los datos en training y test
X = df.drop(['clase'], axis=1)
y = df['clase']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random
```

In [ ]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import random

# Crear el modelo de SVM con kernelal y semilla generadora 2468
random.seed(2468)
svm_linear = SVC(kernel='linear')
```

```
svm_linear.fit(X_train, y_train)
y_pred_linear = svm_linear.predict(X_test)
accuracy_linear = accuracy_score(y_test, y_pred_linear)
print("Accuracy del modelo SVM con kernel lineal:", accuracy_linear)

# Crear el modelo de SVM con kernel RBF y semilla generadora 2468
random.seed(2468)
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_train, y_train)
y_pred_rbf = svm_rbf.predict(X_test)
accuracy_rbf = accuracy_score(y_test, y_pred_rbf)
print("Accuracy del modelo SVM con kernel RBF:", accuracy_rbf)
```

```
Accuracy del modelo SVM con kernel lineal: 0.9471624266144814
Accuracy del modelo SVM con kernel RBF: 0.9373776908023483
```

In [ ]:
```
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
# Crear el modelo de SVM con kernel lineal
svm_linear = SVC(kernel='linear')

# Realizar la validación cruzada con 3 folds
scores = cross_val_score(svm_linear, X_train, y_train, cv=3)

# Imprimir los resultados
print("Accuracy del modelo SVM con kernel lineal y 3-fold cross-validation:", sc
```

```
Accuracy del modelo SVM con kernel lineal y 3-fold cross-validation: 0.93719806
76328503
```

Un puntaje de precisión de 0.937 en el modelo SVM con kernel lineal y 3-fold cross-validation indica que el modelo tiene un buen rendimiento en la tarea de clasificación. Esto indica que el modelo SVM es capaz de clasificar con precisión las imágenes de MRI y detectar la presencia de masa tumoral con una alta tasa de acierto.

In [ ]:
```
from sklearn.model_selection import GridSearchCV

# Definir los valores de C y sigma a explorar
param_grid = {'C': [0.1, 1, 10], 'gamma': [0.1, 1, 10]}

# Crear el modelo de SVM con kernel RBF
svm_rbf = SVC(kernel='rbf')

# Realizar la búsqueda de hiperparámetros validación cruzada de 3 folds
grid_search = GridSearchCV(svm_rbf, param_grid, cv=3)

# Entrenar el modelo con los datos de entrenamiento
grid_search.fit(X_train, y_train)

# Imprimir los resultados
print("Mejores hiperparámetros encontrados:", grid_search.best_params_)
print("Accuracy del modelo SVM con kernel RBF y mejores hiperparámetros:", grid_
```

```
Mejores hiperparámetros encontrados: {'C': 10, 'gamma': 0.1}
Accuracy del modelo SVM con kernel RBF y mejores hiperparámetros: 0.94202898550
72465
```

Los resultados obtenidos indican que el mejor modelo SVM con kernel RBF se obtiene con los hiperparámetros C = 10 y gamma = 0.1. La precisión del modelo con estos

hiperparámetros del 94.2%, lo que indica que el modelo es capaz de clasificar con alta precisión los datos de prueba. En este caso, los datos de prueba son los obtenidos por los escáner MRI y su masa tumoral.

In [ ]:
```
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Crear una lista de los modelos de clasificación
models = [LogisticRegression(max_iter=100000), DecisionTreeClassifier(), RandomF

# Crear una lista de los nombres de los modelos
model_names = ['Regresión Logística', 'Árbol de Decisión', 'Random Forest', 'SVM

# Crear una lista de los puntajes de precisión de cada modelo
# Crear una lista de las métricas
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']

# Crear una lista vacía para almacenar los resultados de cada modelo
results_list = []

# Iterar sobre cada modelo y calcular las métricas
for model in models:
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, output_dict=True)
    precision = report['weighted avg']['precision']
    recall = report['weighted avg']['recall']
    f1_score = report['weighted avg']['f1-score']
    results_list.append([accuracy, precision, recall, f1_score])

# Crear un DataFrame con los resultados
results = pd.DataFrame(results_list, columns=metrics, index=model_names)
print(results)
```

```
                   Accuracy  Precision    Recall  F1 Score
Regresión Logística  0.964775   0.961931  0.964775  0.962405
Árbol de Decisión    0.976517   0.977281  0.976517  0.976848
Random Forest        0.976517   0.976517  0.976517  0.976517
SVM con kernel RBF   0.962818   0.965847  0.962818  0.964068
```

Según los resultados de la tabla, el modelo de Árbol de Decisión y el modelo de Random Forest tienen los mismos puntajes de precisión, recuperación y puntuación F1, lo que que ambos modelos son igualmente buenos en términos de rendimiento. Sin embargo, el modelo de Árbol de Decisión tiene una precisión ligeramente más alta que el modelo de Random Forest, lo que lo convierte en el mejor modelo en términos de precisión. Además, el modelo de Árbol de Decisión tiene una puntuación de precisión recuperación y puntuación F1 más alta que los otros dos modelos, Regresión Logística y SVM con

kernel RBF. Por lo tanto, el modelo de Árbol de Decisión es mejor modelo en términos de rendimiento general.