aiml-exp-6

October 25, 2024

G.KRUTHIK ROSHAN

BATCH >> 41

2303A52339

1 Implement the K Nearest Neighbor Classification using Classified Manufacturing Dataset

Part 1 – Import the required Python, Pandas, Matplotlib, Seaborn packages. [CO1] 1. Load the classified dataset into a dataframe using pandas 2. Check the data types of each feature(column) in the dataset. 3. Generate a summary of the dataset for min, max, stddev, quartile vales for 25%,50%,75%,90%, 4. List the names of columns/features in the dataset 5. Scale the features using StandardScaler and transform the data

Part 2 – Model training and Fit the data to Model. [CO2]

1. Split the data generated from list created as X, Y is distributed using train test split function as X train, Y train, X test, Y test
2. Apply the KNN Classifier model of sklearn.neighbors import KNeighborsClassifier package
3. Fit the data to the Classier Model using fit. Part 3 – Evaluate the Classification Quality. [CO3]
4. Generate the confusion matrix to estimate the correction among features
5. Generate the classification report using classification report

Part 3 – Evaluate the Classification Quality. [CO3]

1. Generate the confusion matrix to estimate the correction among features
2. Generate the classification report using classification report

[1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from google.colab import files
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

[2]:
```
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>

Saving Classified Data.unknown to Classified Data (1).unknown
```

[4]:
```python
a = pd.read_csv('Classified Data.unknown', index_col=0)
```

[5]:
```python
print("Data Types of Each Feature:")
print(a.dtypes)
```

```
Data Types of Each Feature:
WTT             float64
PTI             float64
EQW             float64
SBI             float64
LQE             float64
QWG             float64
FDJ             float64
PJF             float64
HQE             float64
NXJ             float64
TARGET CLASS      int64
dtype: object
```

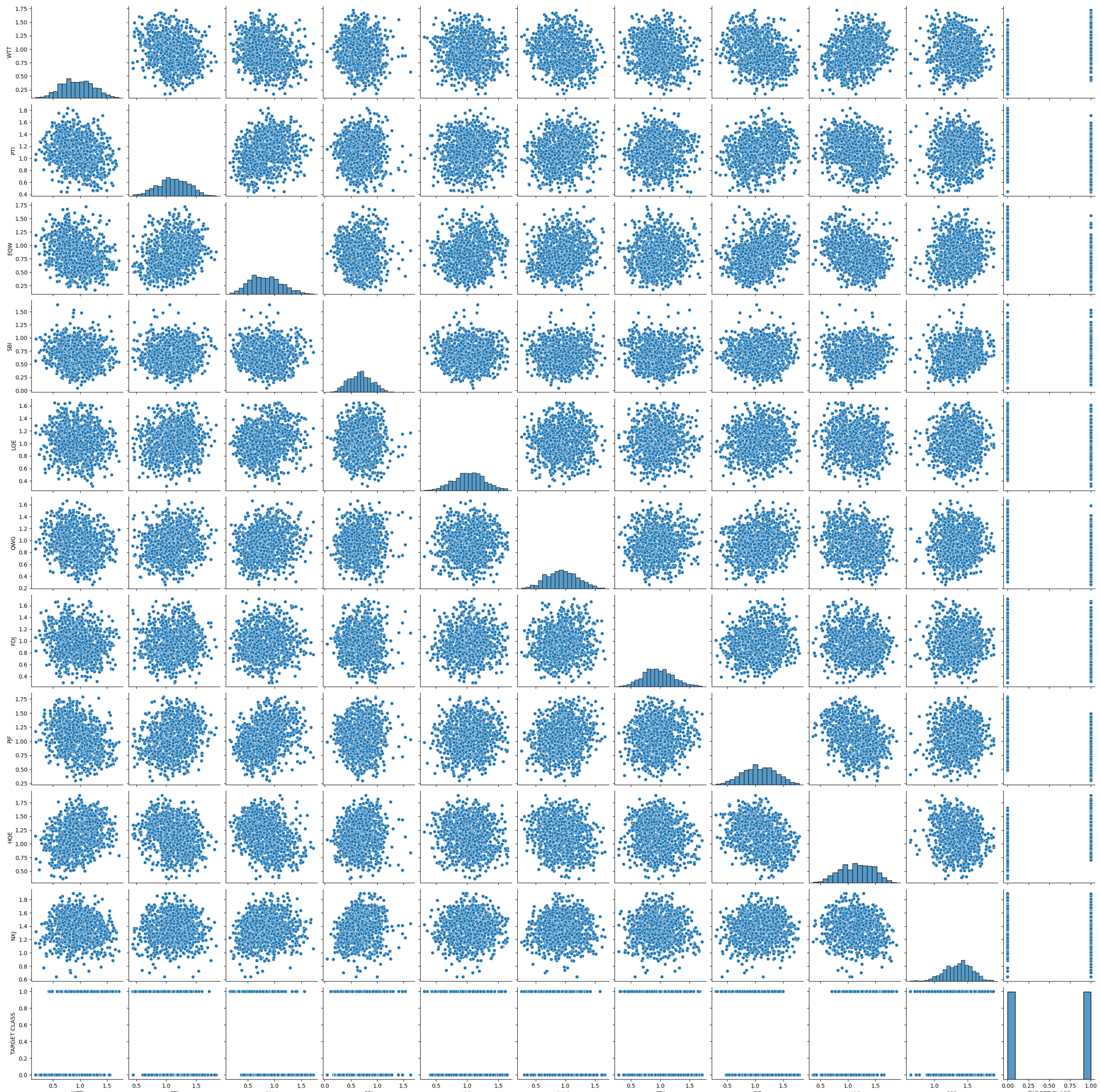
[6]:
```python
print("\nSummary Statistics of the Dataset (including quartiles):")
print(a.describe(percentiles=[.25, .5, .75, .90]))
```

```
Summary Statistics of the Dataset (including quartiles):
               WTT          PTI          EQW          SBI          LQE  \
count  1000.000000  1000.000000  1000.000000  1000.000000  1000.000000
mean      0.949682     1.114303     0.834127     0.682099     1.032336
std       0.289635     0.257085     0.291554     0.229645     0.243413
min       0.174412     0.441398     0.170924     0.045027     0.315307
25%       0.742358     0.942071     0.615451     0.515010     0.870855
50%       0.940475     1.118486     0.813264     0.676835     1.035824
75%       1.163295     1.307904     1.028340     0.834317     1.198270
90%       1.336612     1.441901     1.223127     0.983470     1.341138
max       1.721779     1.833757     1.722725     1.634884     1.650050

               QWG          FDJ          PJF          HQE          NXJ  \
count  1000.000000  1000.000000  1000.000000  1000.000000  1000.000000
mean      0.943534     0.963422     1.071960     1.158251     1.362725
std       0.256121     0.255118     0.288982     0.293738     0.204225
min       0.262389     0.295228     0.299476     0.365157     0.639693
25%       0.761064     0.784407     0.866306     0.934340     1.222623
50%       0.941502     0.945333     1.065500     1.165556     1.375368
75%       1.123060     1.134852     1.283156     1.383173     1.504832
90%       1.277552     1.306497     1.452713     1.535520     1.618096
```

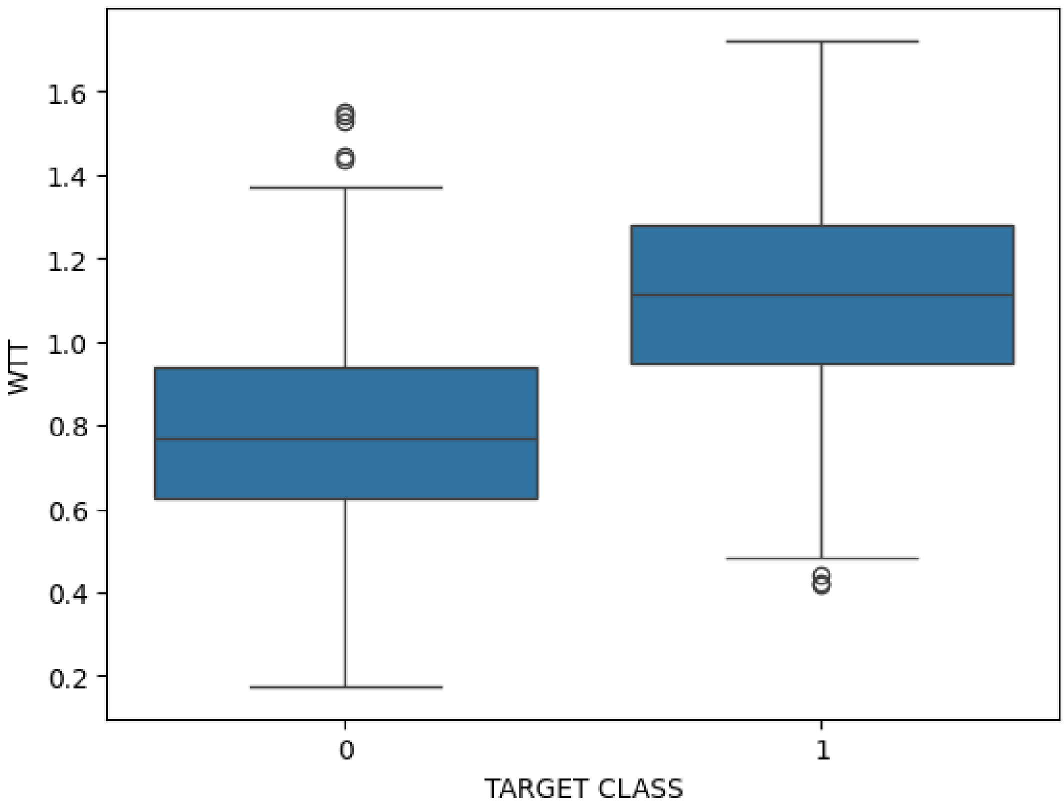
```
max       1.666902     1.713342     1.785420     1.885690     1.893950

       TARGET CLASS
count    1000.00000
mean        0.50000
std         0.50025
min         0.00000
25%         0.00000
50%         0.50000
75%         1.00000
90%         1.00000
max         1.00000
```

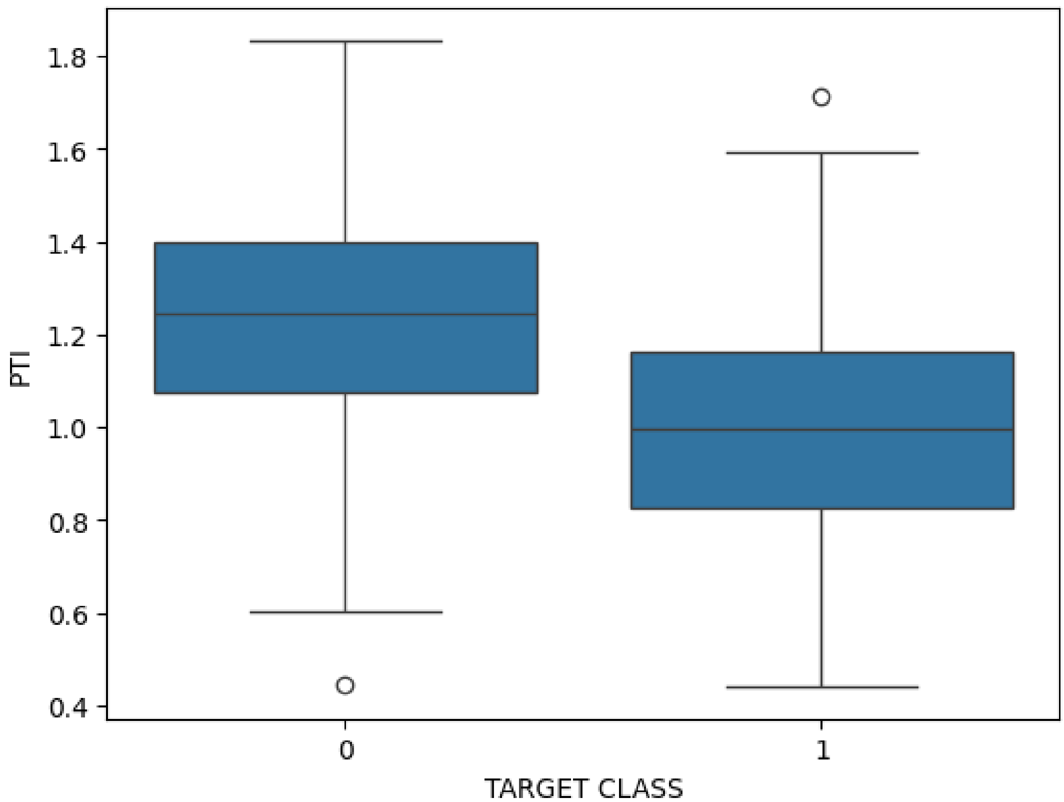
[7]:
```
ls = list(a.columns)
sns.pairplot(a)
```

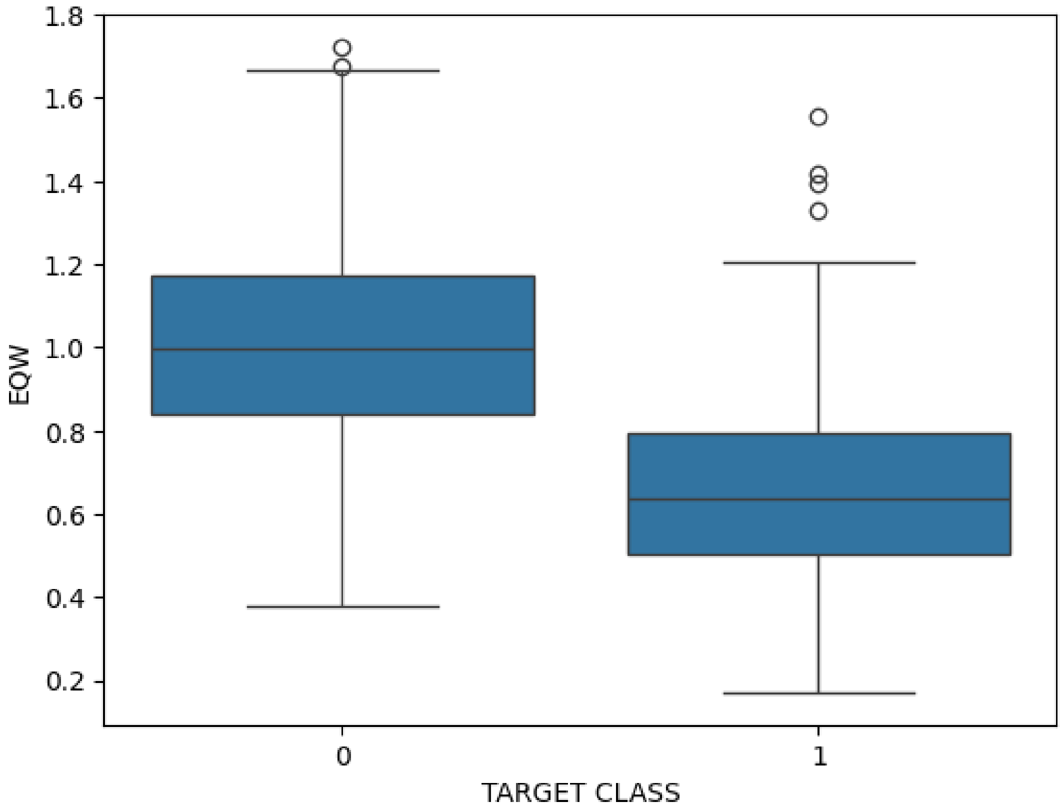
[7]: <seaborn.axisgrid.PairGrid at 0x7a260f222680>

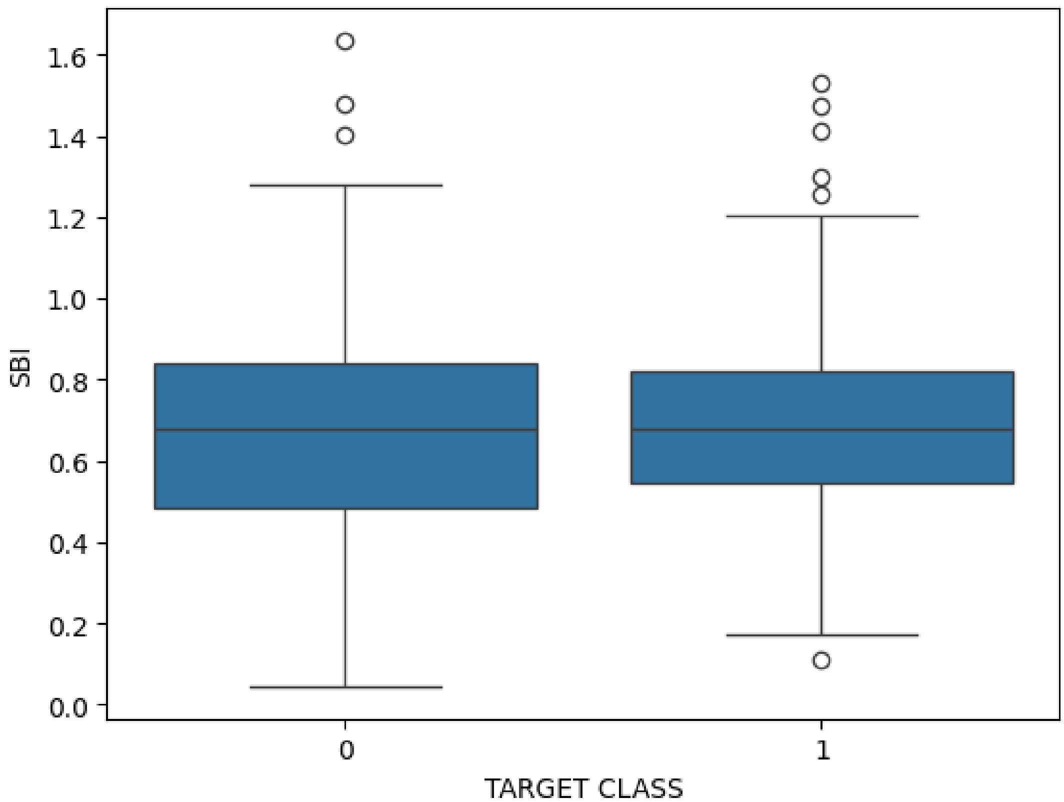


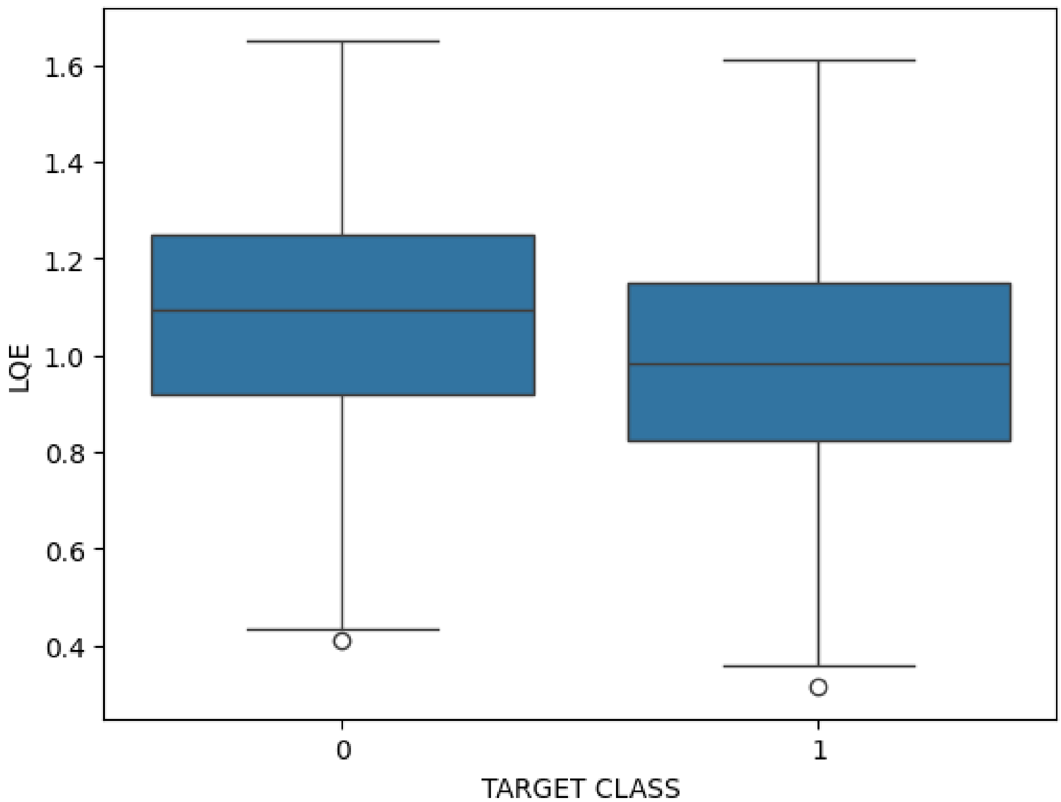
[9]:
```
for i in range(len(ls)-1):
  sns.boxplot(x='TARGET CLASS', y=ls[i], data=a)
  plt.show()
```

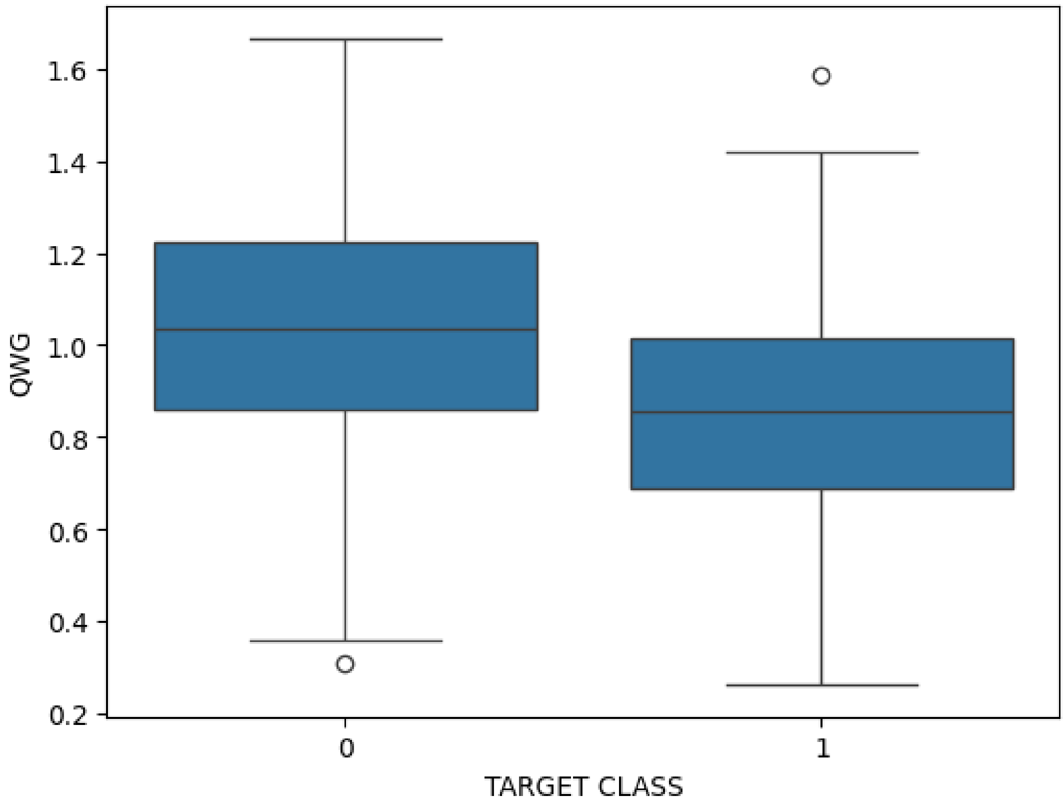

1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
WTT
0
1
TARGET CLASS

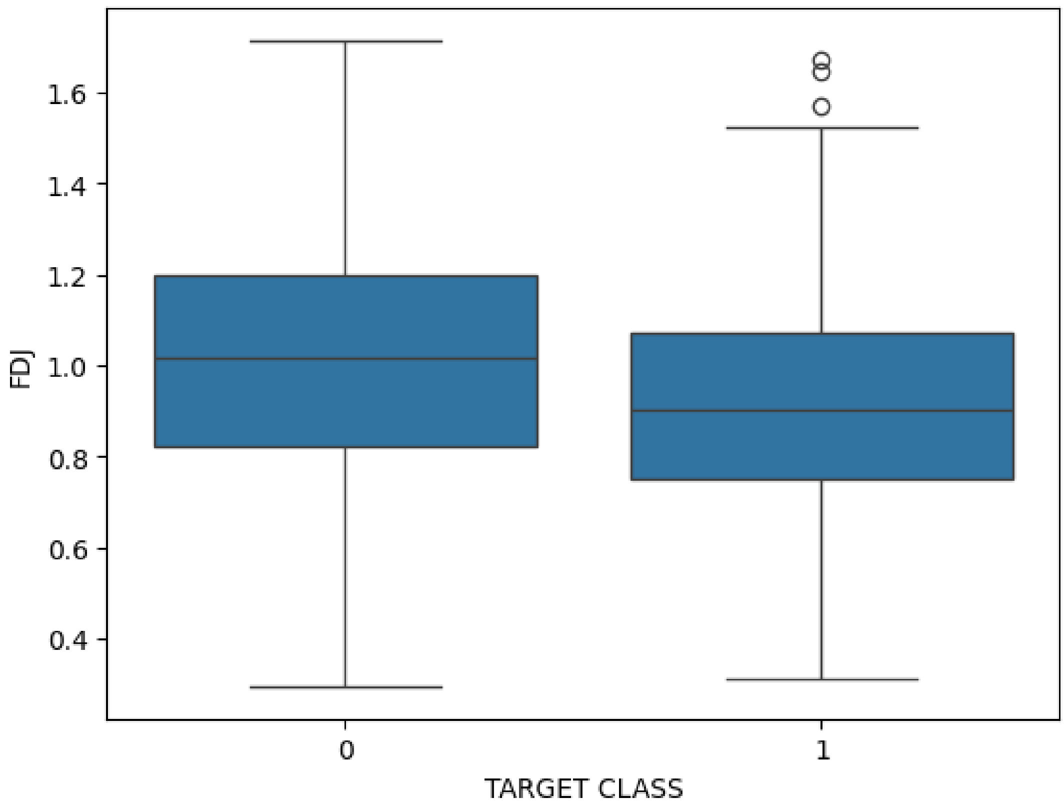

1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
PTI
0
1
TARGET CLASS

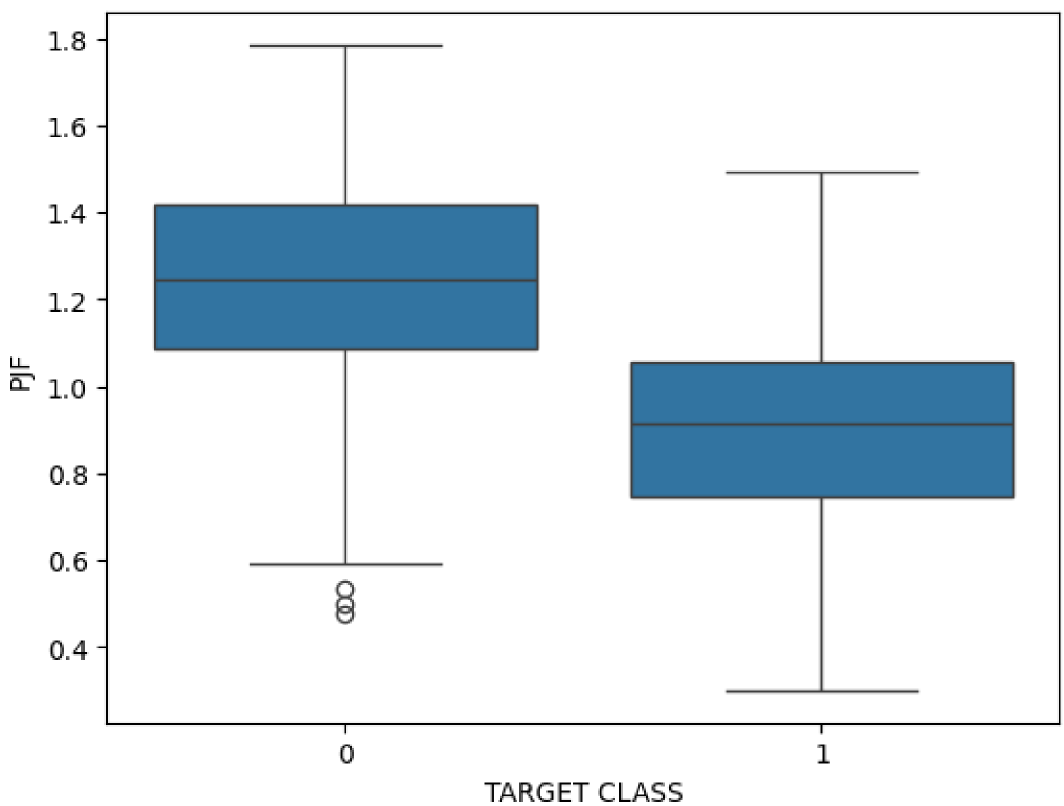

1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
EQW
0
1
TARGET CLASS

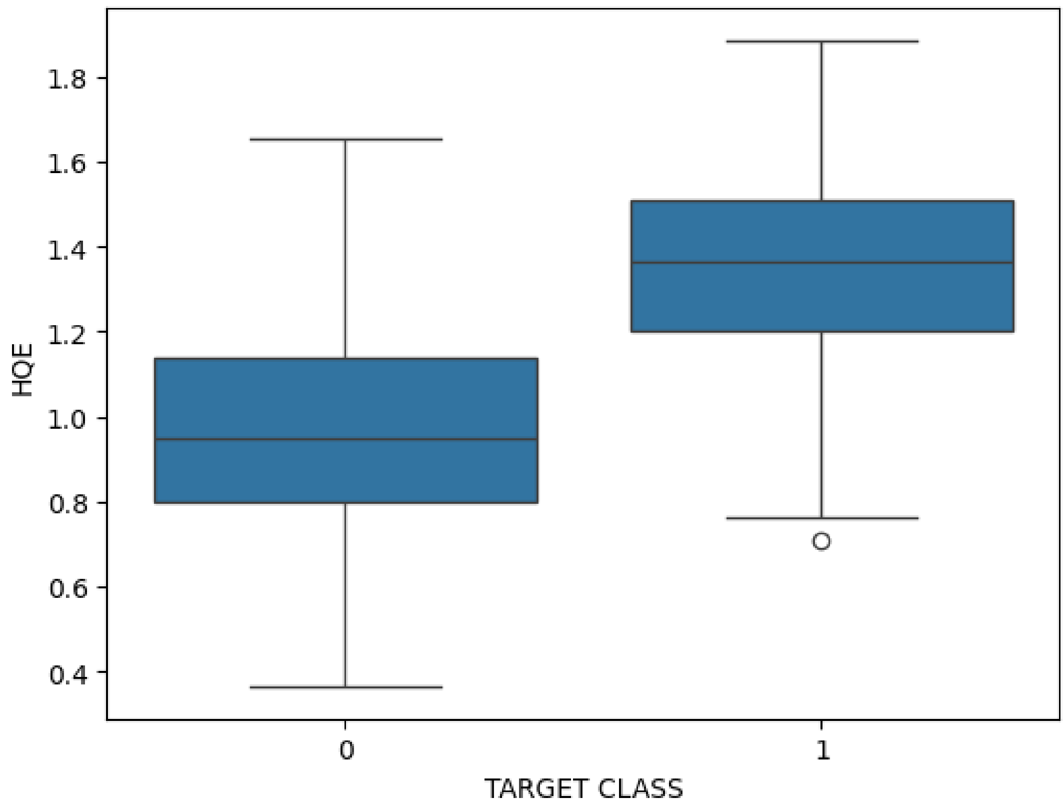

1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
SBI
0
1
TARGET CLASS

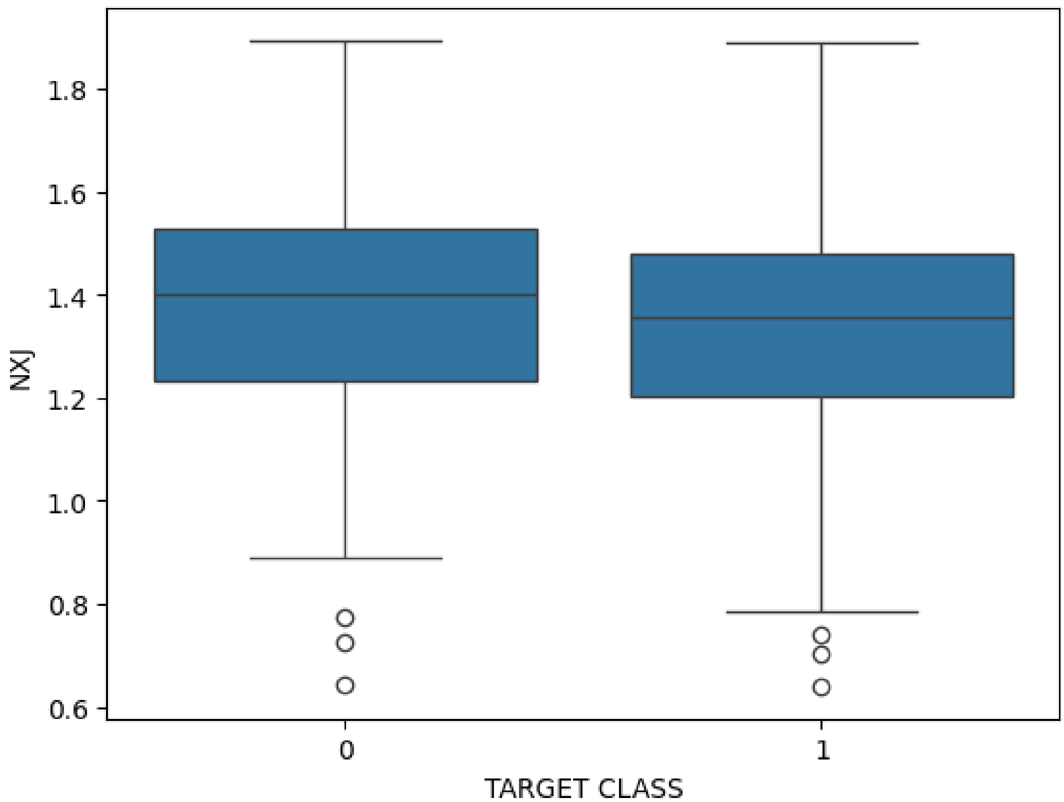

1.6
1.4
1.2
1.0
0.8
0.6
0.4
LQE
0
1
TARGET CLASS


1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
QWG
0
1
TARGET CLASS


1.6
1.4
1.2
1.0
0.8
0.6
0.4
FDJ
0
1
TARGET CLASS


1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
PJF
0
1
TARGET CLASS


1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
HQE
0
1
TARGET CLASS



[10]:
```
scaler = StandardScaler()
scaler.fit(a.drop('TARGET CLASS', axis=1))
scaled_features = scaler.transform(a.drop('TARGET CLASS', axis=1))
```

[11]:
```
a_feat = pd.DataFrame(scaled_features, columns=a.columns[:-1])
print(a_feat.head())
```

```
        WTT       PTI       EQW       SBI       LQE       QWG       FDJ  \
0 -0.123542  0.185907 -0.913431  0.319629 -1.033637 -2.308375 -0.798951
1 -1.084836 -0.430348 -1.025313  0.625388 -0.444847 -1.152706 -1.129797
2 -0.788702  0.339318  0.301511  0.755873  2.031693 -0.870156  2.599818
3  0.982841  1.060193 -0.621399  0.625299  0.452820 -0.267220  1.750208
4  1.139275 -0.640392 -0.709819 -0.057175  0.822886 -0.936773  0.596782

        PJF       HQE       NXJ
0 -1.482368 -0.949719 -0.643314
1 -0.202240 -1.828051  0.636759
2  0.285707 -0.682494 -0.377850
3  1.066491  1.241325 -1.026987
4 -1.472352  1.040772  0.276510
```

[18]:
```
from sklearn.model_selection import train_test_split
# Corrected the variable name from scaled_featuX_train to scaled_features
X_train, X_test, y_train, y_test = train_test_split(scaled_features, a['TARGET␣
↪CLASS'], test_size=0.30, random_state=101)
```

[19]:
```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
```

[19]:
```
KNeighborsClassifier(n_neighbors=1)
```

[21]:
```
# Corrected the variable name from X_tests to X_test
p = knn.predict(X_test)
```

[22]:
```
conf_max = confusion_matrix(y_test, p)
print("Confusion Matrix:\n", conf_max)
print("\nClassification Report:\n", classification_report(y_test,p))
```

```
Confusion Matrix:
 [[151   8]
 [ 15 126]]

Classification Report:
               precision    recall  f1-score   support

           0       0.91      0.95      0.93       159
           1       0.94      0.89      0.92       141

    accuracy                           0.92       300
   macro avg       0.92      0.92      0.92       300
weighted avg       0.92      0.92      0.92       300
```