Atividade Prática 3 - Aprendizado de Máquina

Considere a base de dados Wine Quality:
https://www.kaggle.com/datasets/yasserh/wine-quality-dataset

É um problema de classificação, sendo o atributo classe definido por:
quality (score between 0 and 10)

Observações:

1. Avaliar a base de dados com os algoritmos Árvore de Decisão e KNN;
2. Avaliar o desempenho dos algoritmos com e sem a normalização por padronização (Standard Scaler);
3. Enviar os arquivo ipynb e pdf do código fonte

Responsável: Marcos Angelo Cemim

Importação inicial

In [ ]:
```
import opendatasets as od

import pandas as pd

from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

import statsmodels.api as sm
```

Download dataset & Load into pandas.dataframe

In [ ]:
```
owner_user = 'yasserh'
project_name = 'wine-quality-dataset'
file_name = 'WineQT.csv'
```

In [ ]:
```
try:
    data = pd.read_csv(f'{project_name}/{file_name}')
except FileNotFoundError:
    od.download(f'https://www.kaggle.com/datasets/{owner_user}/{project_name}')
    data = pd.read_csv(f'{project_name}/{file_name}')
```

EDA

In [ ]:
```
data.describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%
fixed acidity	1143.0	8.311111	1.747595	4.60000	7.10000	7.90000	9.100000
volatile acidity	1143.0	0.531339	0.179633	0.12000	0.39250	0.52000	0.640000
citric acid	1143.0	0.268364	0.196686	0.00000	0.09000	0.25000	0.420000
residual sugar	1143.0	2.532152	1.355917	0.90000	1.90000	2.20000	2.600000
chlorides	1143.0	0.086933	0.047267	0.01200	0.07000	0.07900	0.090000
free sulfur dioxide	1143.0	15.615486	10.250486	1.00000	7.00000	13.00000	21.000000
total sulfur dioxide	1143.0	45.914698	32.782130	6.00000	21.00000	37.00000	61.000000
density	1143.0	0.996730	0.001925	0.99007	0.99557	0.99668	0.997845
pH	1143.0	3.311015	0.156664	2.74000	3.20500	3.31000	3.400000
sulphates	1143.0	0.657708	0.170399	0.33000	0.55000	0.62000	0.730000
alcohol	1143.0	10.442111	1.082196	8.40000	9.50000	10.20000	11.100000
quality	1143.0	5.657043	0.805824	3.00000	5.00000	6.00000	6.000000
Id	1143.0	804.969379	463.997116	0.00000	411.00000	794.00000	1209.500000

In [ ]:
```
data.head()
```

Out[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphate
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.6
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.6
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.5
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.5

In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1143 entries, 0 to 1142
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1143 non-null   float64
 1   volatile acidity      1143 non-null   float64
 2   citric acid           1143 non-null   float64
 3   residual sugar        1143 non-null   float64
 4   chlorides             1143 non-null   float64
 5   free sulfur dioxide   1143 non-null   float64
 6   total sulfur dioxide  1143 non-null   float64
 7   density               1143 non-null   float64
 8   pH                    1143 non-null   float64
 9   sulphates             1143 non-null   float64
 10  alcohol               1143 non-null   float64
 11  quality               1143 non-null   int64  
 12  Id                    1143 non-null   int64  
dtypes: float64(11), int64(2)
memory usage: 116.2 KB
```

In [ ]:
```
data.isna().sum()
```

Out[ ]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
Id                      0
dtype: int64
```

In [ ]:
```
data.shape
```

Out[ ]:
```
(1143, 13)
```

In [ ]:
```
data.columns
```

Out[ ]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality', 'Id'],
      dtype='object')
```

In [ ]:
```
data.drop(['Id'], axis = 1, inplace=True)
```

In [ ]:
```
for c in data.columns:
    print(f'{c:20s} {len(data[c].unique())}', end=' ')
    if len(data[c].unique()) < 10:
        print(data[c].unique())
    else:
        print()
```

```
fixed acidity         91
volatile acidity      135
citric acid           77
residual sugar        80
chlorides             131
free sulfur dioxide   53
total sulfur dioxide  138
density               388
pH                    87
sulphates             89
alcohol               61
quality               6 [5 6 7 4 8 3]
```

Aplicação do algoritmo

Preparação

Transformação dos dados categóricos

In [ ]:
```
le = LabelEncoder()
data['quality'] = le.fit_transform(data['quality'])
data.head()
```

Out[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphate
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.6
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.6
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.5
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.5

Dividindo a base original

In [ ]:
```
X = data.drop(['quality'], axis=True).values
y = data['quality'].values
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.7, test_siz
```

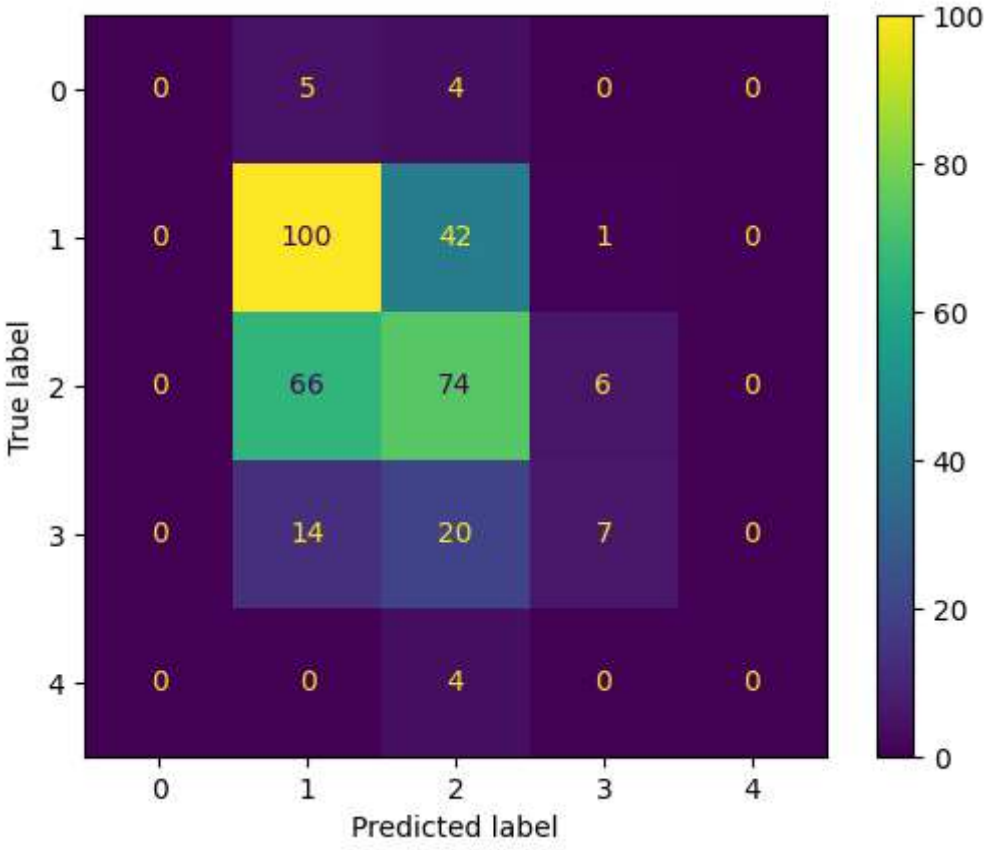
Aplicando KNN

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=13, metric = 'euclidean')
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
knn_acc = accuracy_score(y_test, y_pred_knn)
print(f'Precisão: {knn_acc:.4%}')

cm = confusion_matrix(y_test, y_pred_knn)
```

```
disp = ConfusionMatrixDisplay(cm).plot()
plt.show()
```

```
Precisão: 52.7697%
```



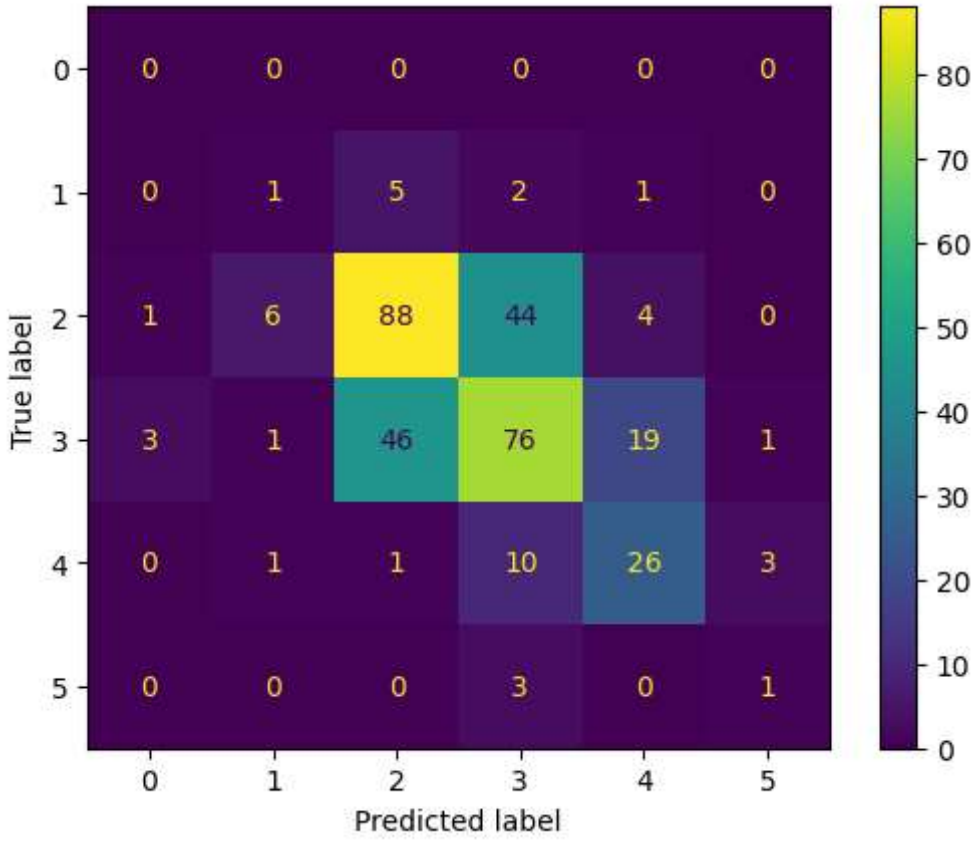
Aplicando arvore de decisão

In [ ]:
```
dectree = DecisionTreeClassifier(criterion="entropy")
dectree.fit(X_train, y_train)
y_pred_dt = dectree.predict(X_test)
dt_acc = accuracy_score(y_test, y_pred_dt)

print(f'Precisão: {dt_acc:.4%}')

cm = confusion_matrix(y_test, y_pred_dt)
disp = ConfusionMatrixDisplay(cm).plot()
plt.show()
```

```
Precisão: 55.9767%
```

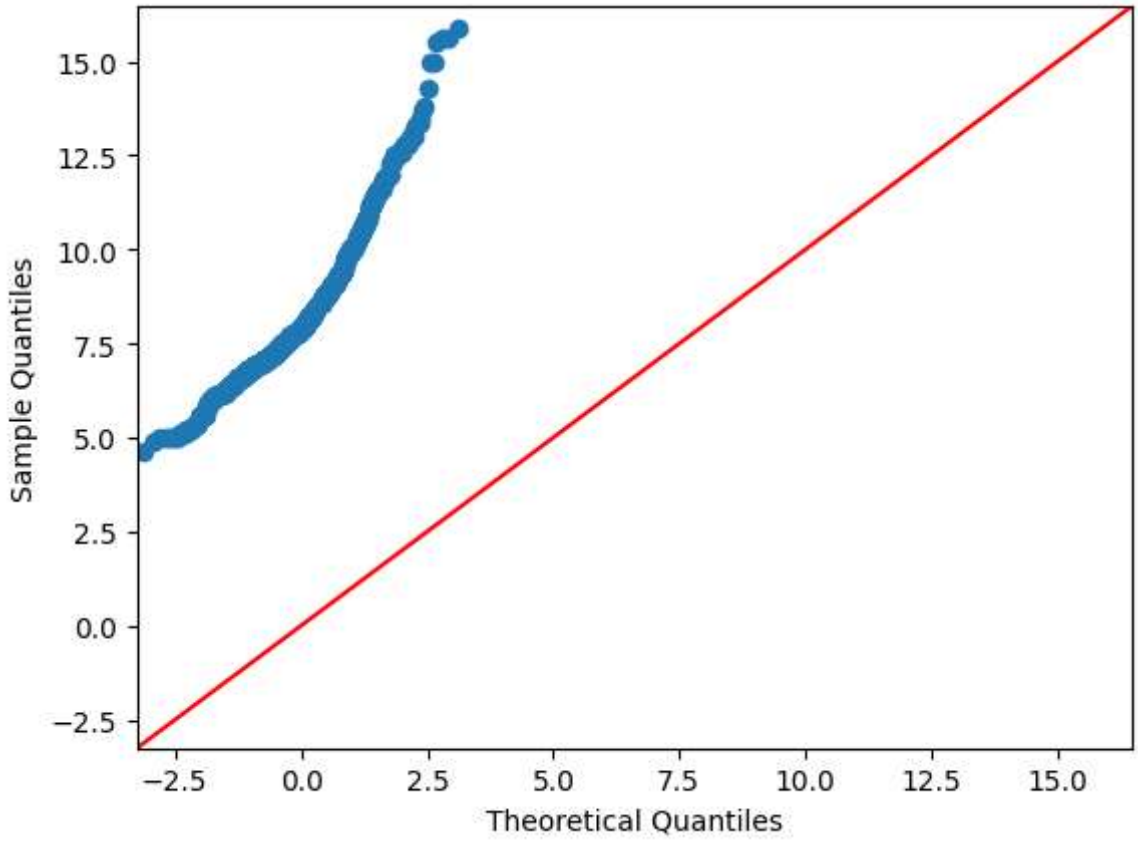


Testando normalidade, normalizando e dividindo a nova base

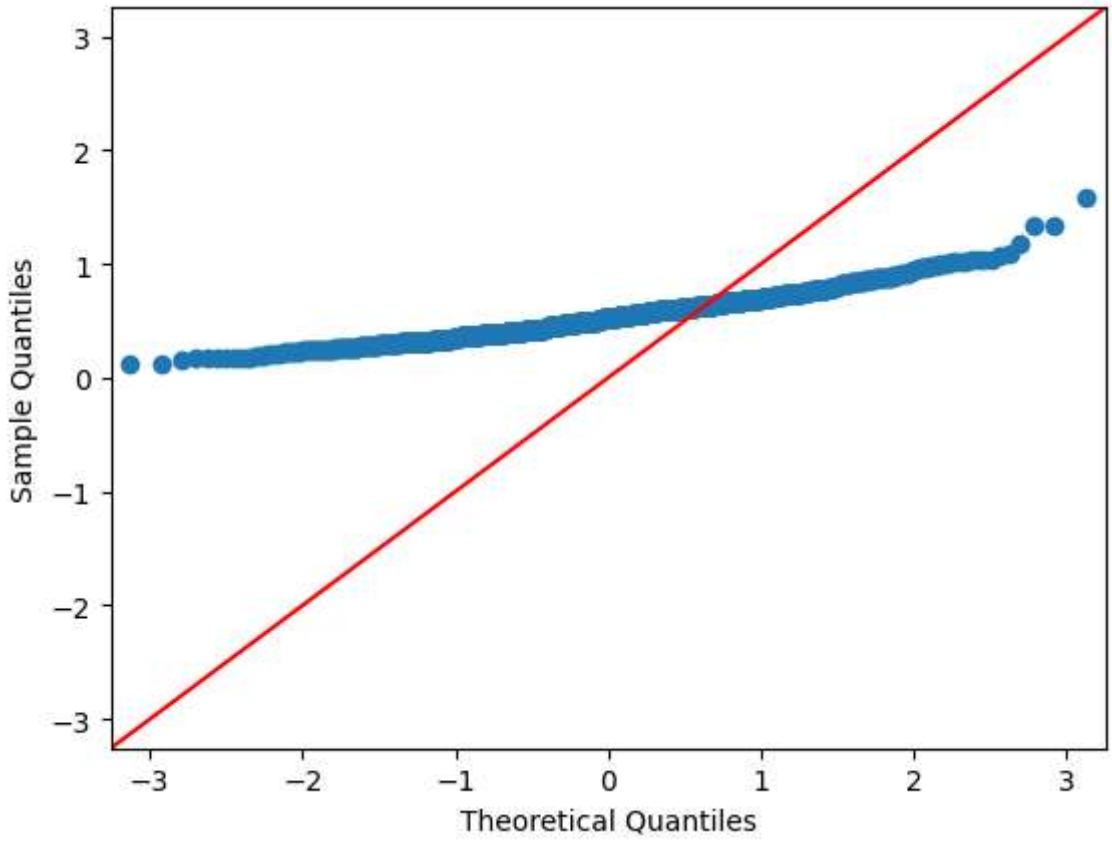
In [ ]:
```
from scipy import stats
for c in data.columns:

    res = stats.normaltest(data[c])
    print(c, res.statistic)
    disp = sm.qqplot(data[c], line='45')
    plt.show()
```

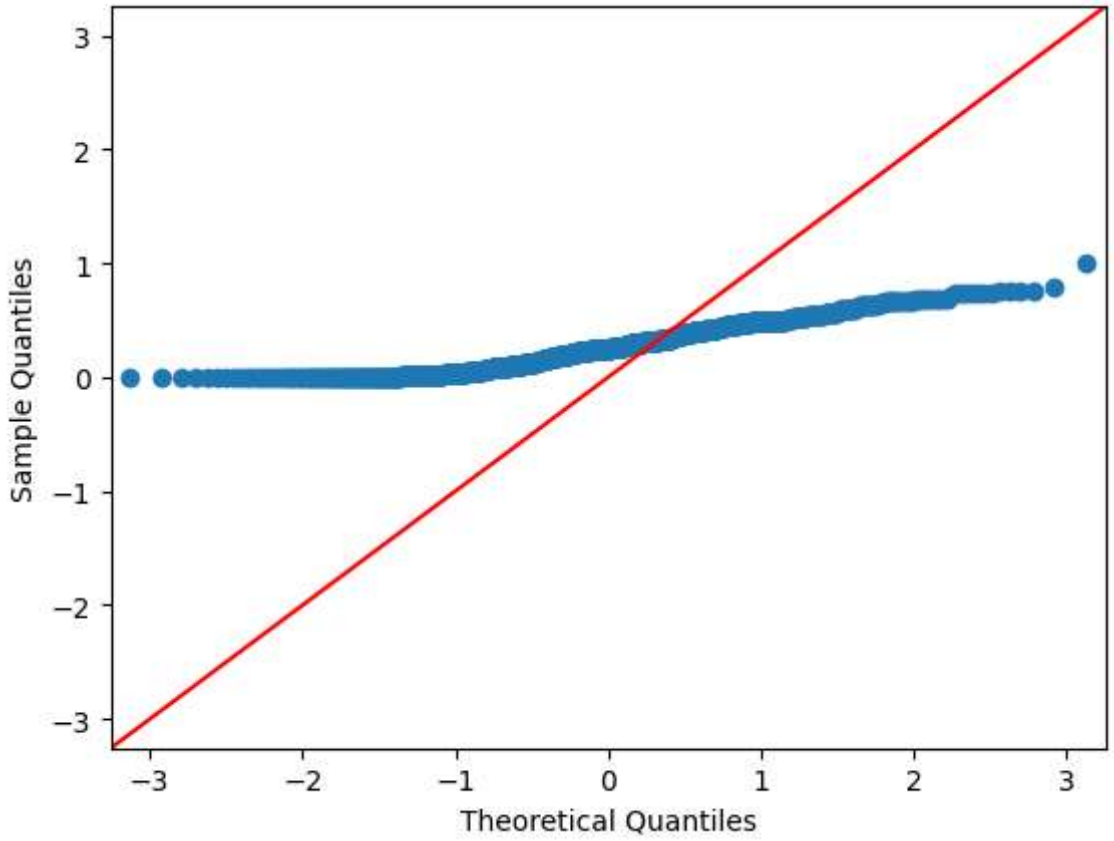
```
fixed acidity 182.74633288718823
```



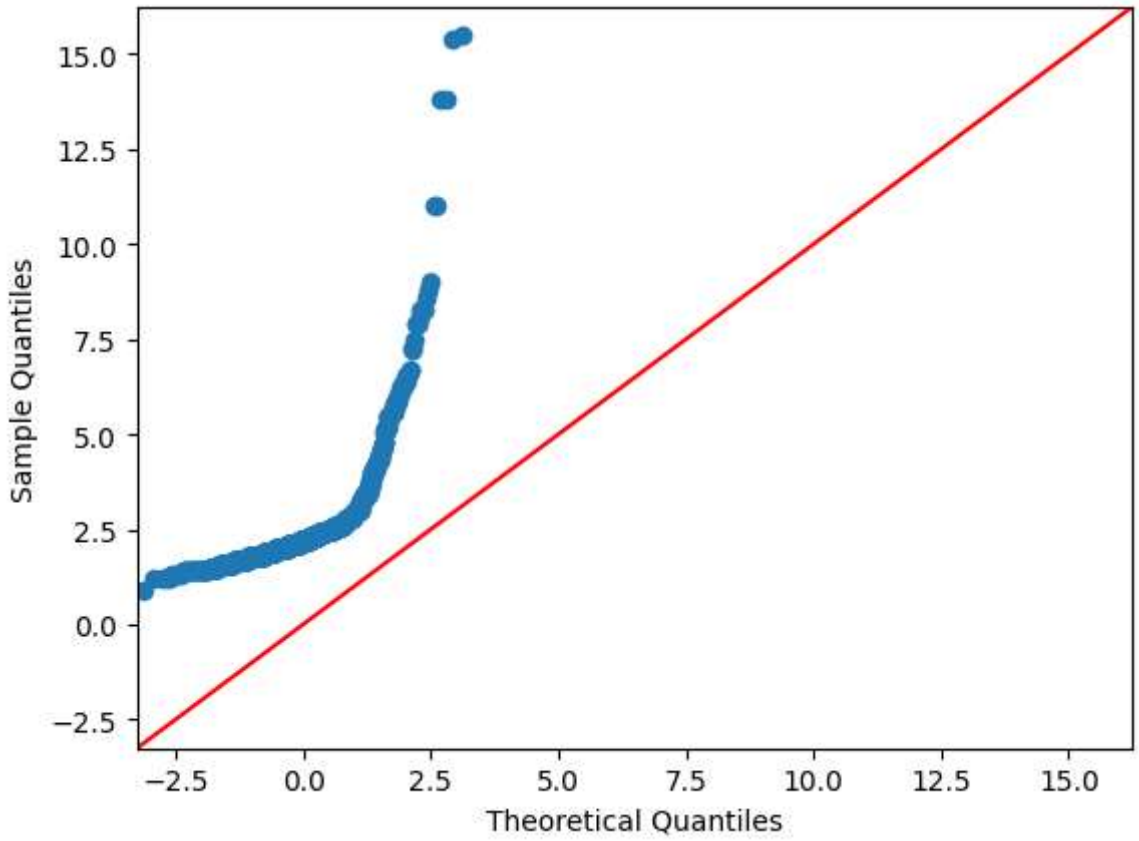

```
volatile acidity 110.0654289871664
```



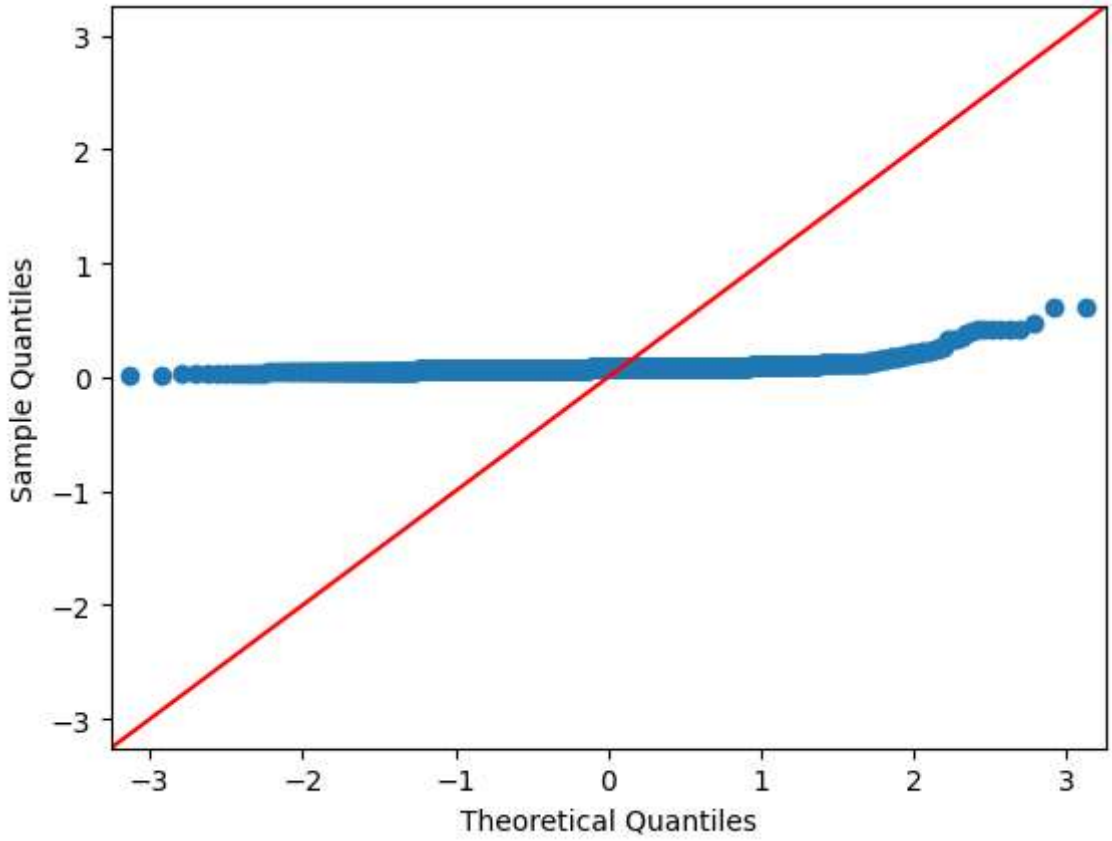

```
citric acid 87.97087069318528
```



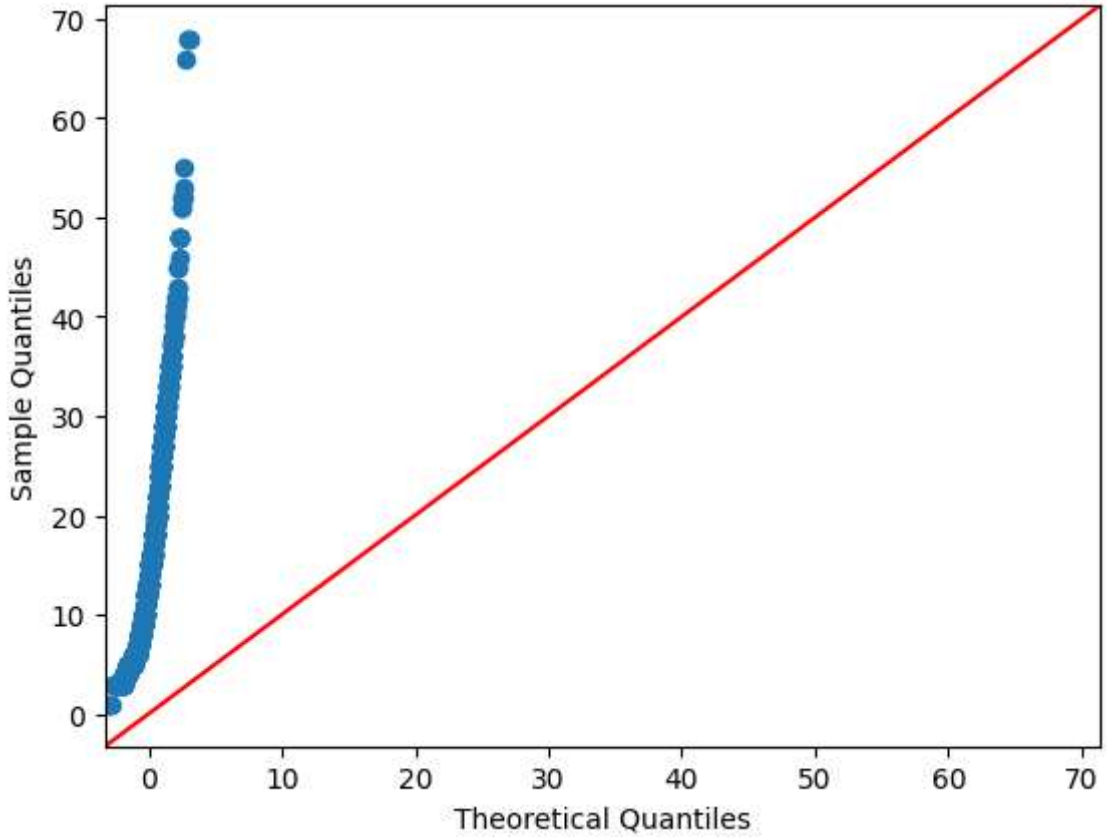

```
residual sugar 1068.7882569291467
```



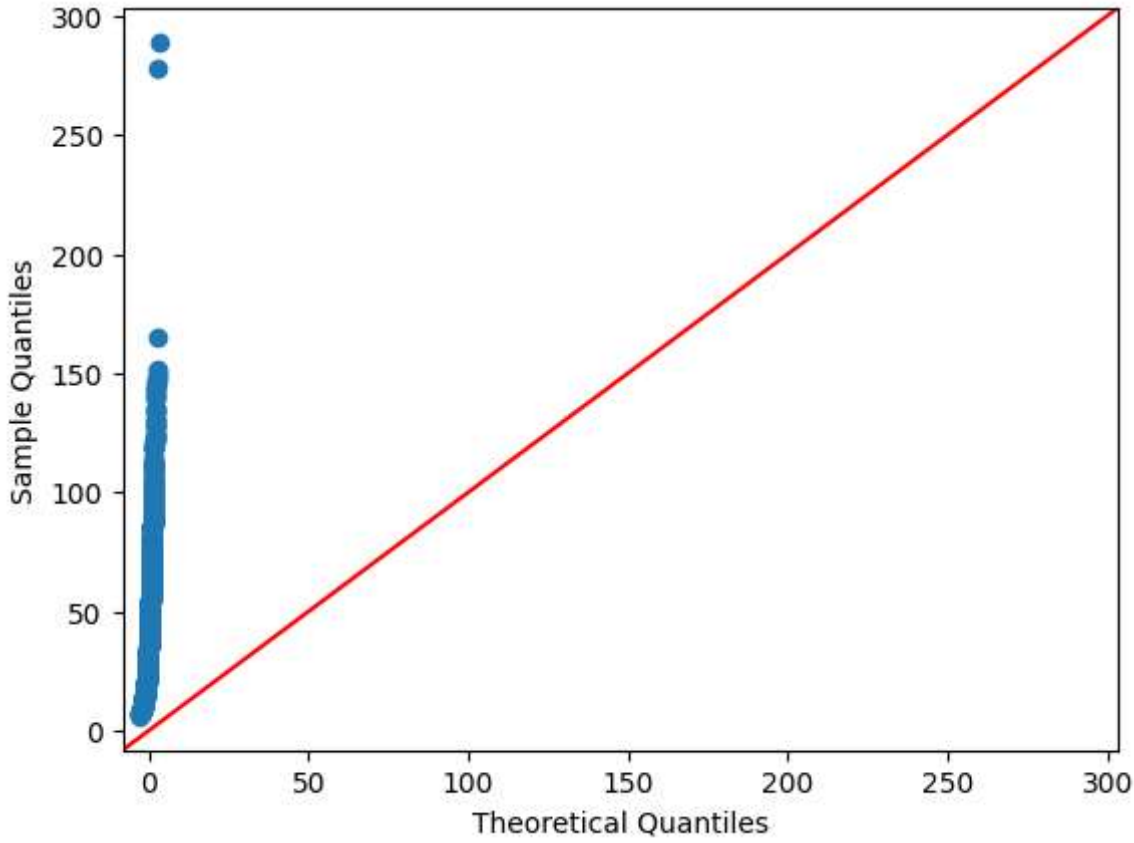

```
chlorides 1341.8699066427403
```



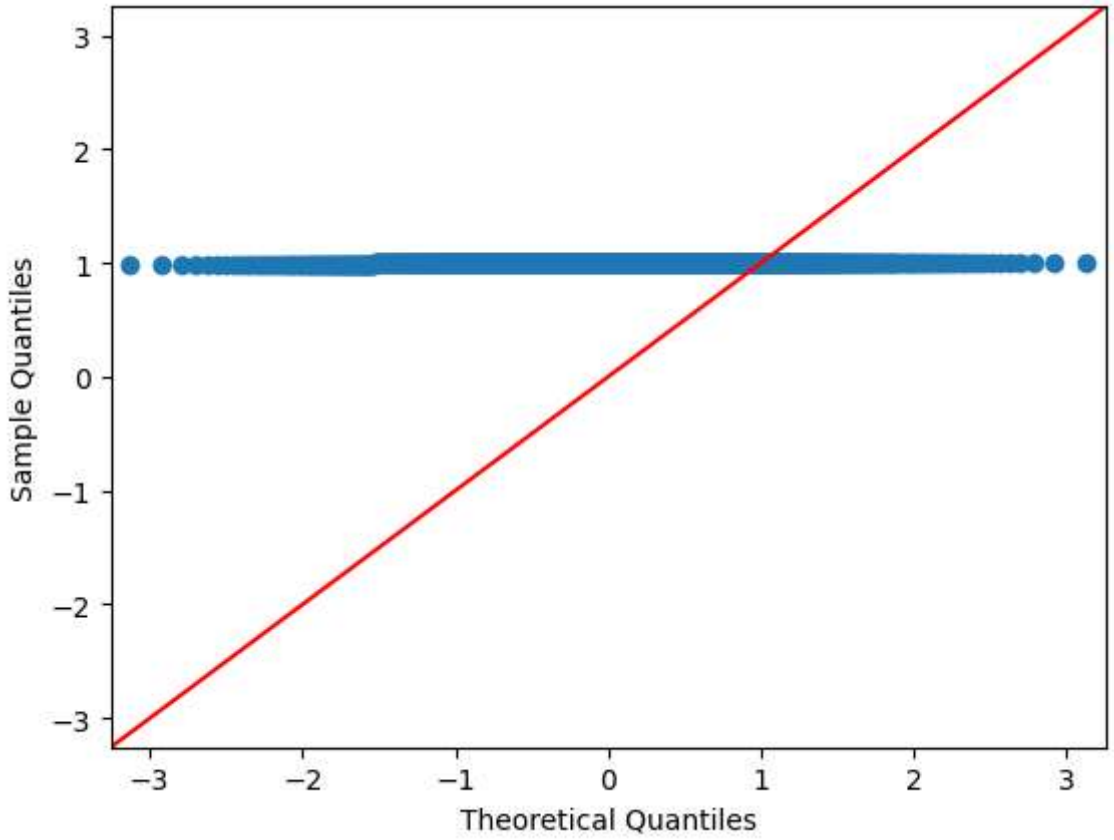
free sulfur dioxide 239.52492155433637



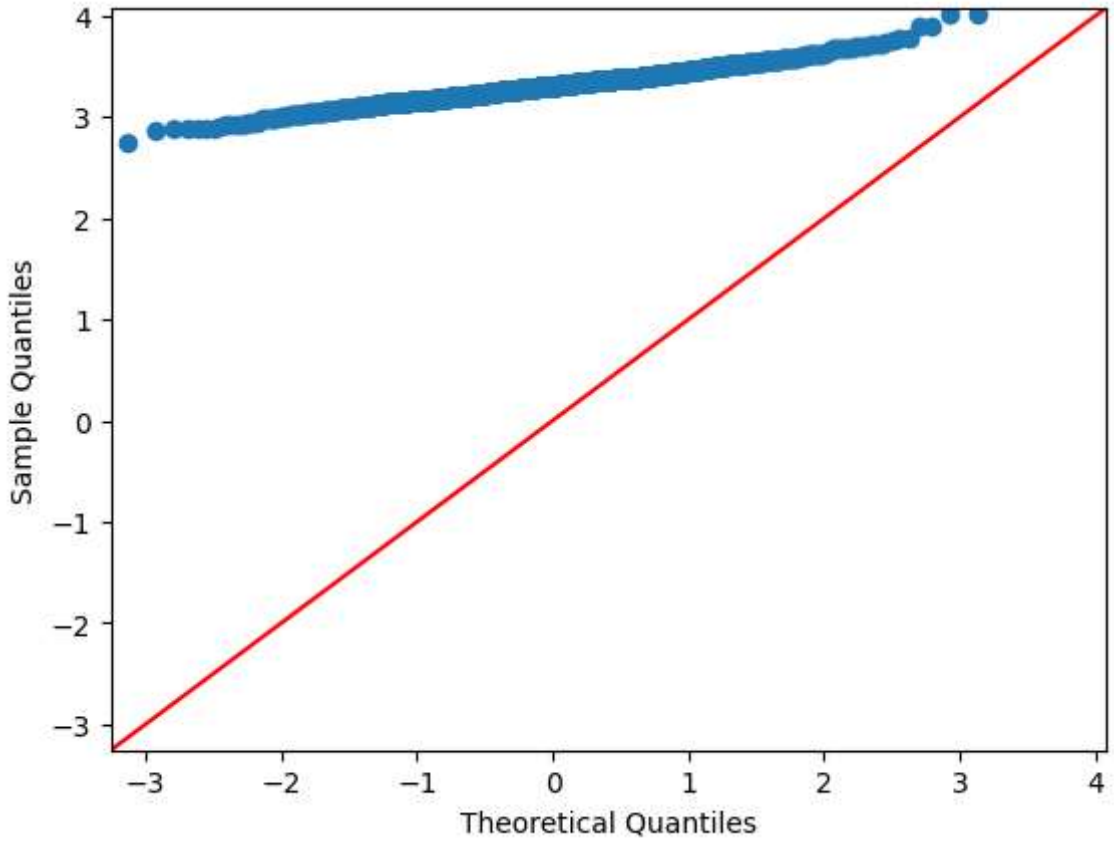
total sulfur dioxide 410.9501276204486



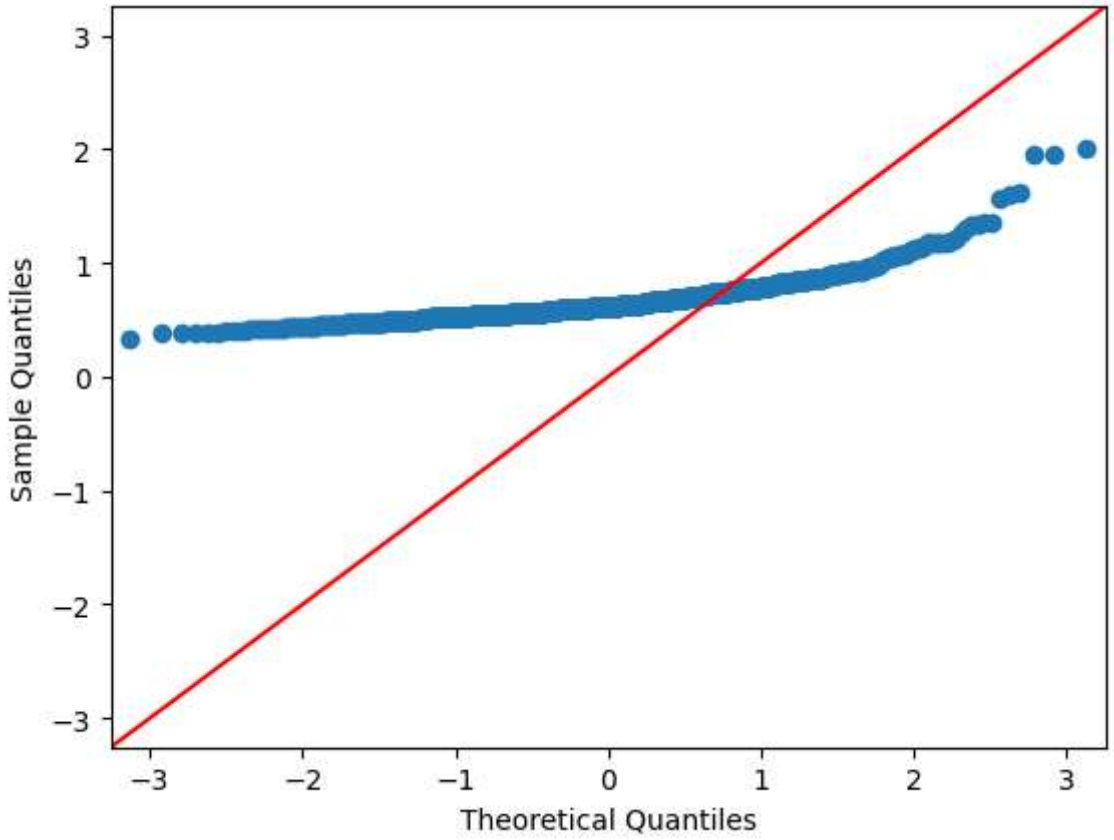

```
density 21.88158500337791
```



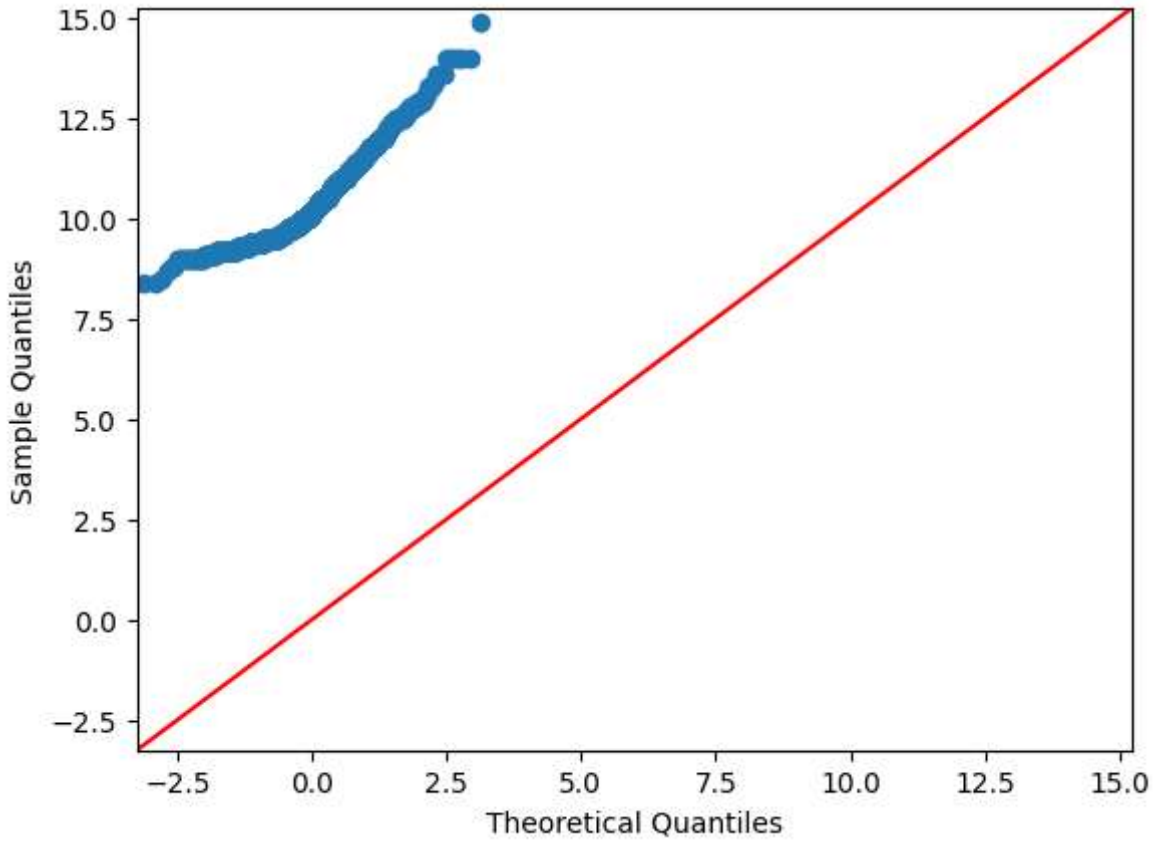

```
pH 30.27193701732856
```



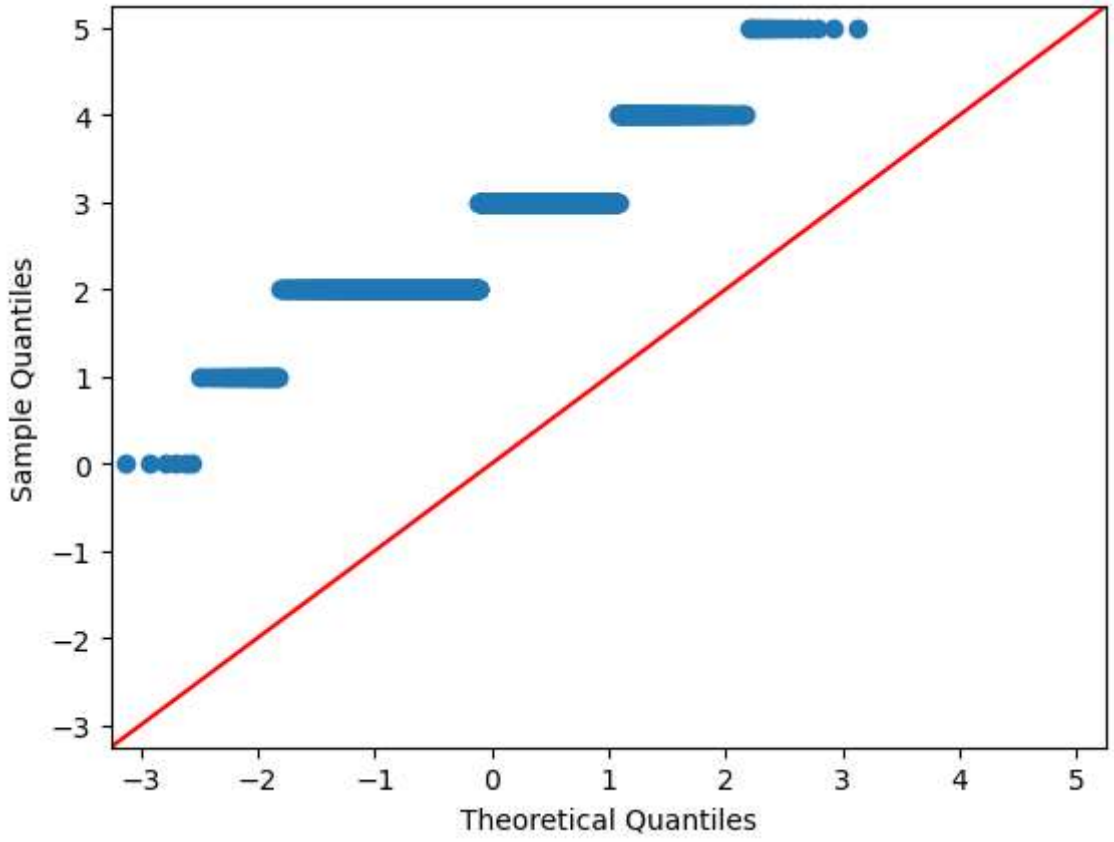

```
sulphates 670.1879021910112
```




```
alcohol 111.56735159965109
```




```
quality 19.13680020029779
```



Pelo encontrado acima, não é praticamente viável realizar a normalização dos dados.

Porém, para fins didáticos, realizarei a normalização e aplicarei novamente os algoritmos

In [ ]:
```
std=StandardScaler()
data_scaled = data.copy()
```

```
data_scaled[['fixed acidity',
             'volatile acidity',
             'citric acid',
             'residual sugar',
             'chlorides',
             'free sulfur dioxide',
             'total sulfur dioxide',
             'density',
             'pH',
             'sulphates',
             'alcohol'
            ]] = std.fit_transform(
                data_scaled[['fixed acidity',
             'volatile acidity',
             'citric acid',
             'residual sugar',
             'chlorides',
             'free sulfur dioxide',
             'total sulfur dioxide',
             'density',
             'pH',
             'sulphates',
             'alcohol'
            ]]
            )
```

In [ ]:
```
X_scaled = data_scaled.drop(['quality'], axis=True).values
y_scaled = data_scaled['quality'].values
```

In [ ]:
```
X_train_scaled, X_test_scaled, y_train_scaled, y_test_scaled = train_test_split(
```

Dados normalizados

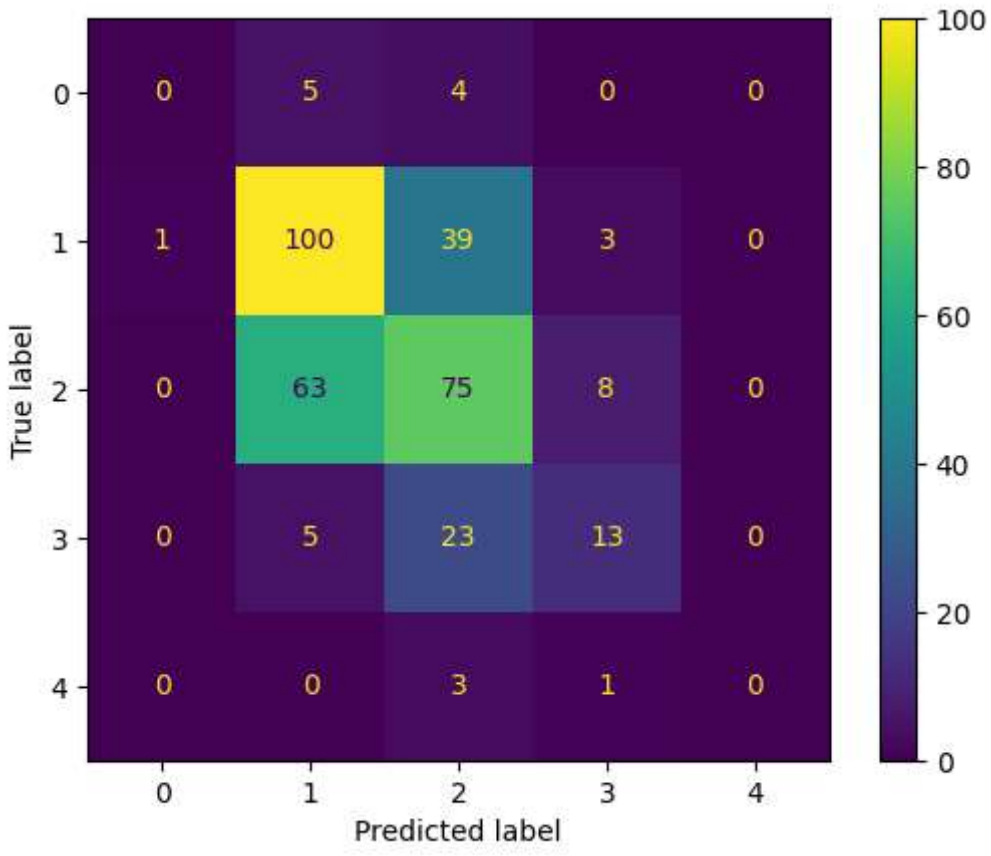
Aplicando KNN

In [ ]:
```
knn_scaled = KNeighborsClassifier(n_neighbors=13, metric='euclidean')
knn_scaled.fit(X_train_scaled,y_train_scaled)
y_pred_knn_scaled = knn_scaled.predict(X_test_scaled)
knn_acc_scaled = accuracy_score(y_test_scaled, y_pred_knn_scaled)

print(f'Precisão: {knn_acc_scaled:.4%}')

cm = confusion_matrix(y_test, y_pred_knn_scaled)
disp = ConfusionMatrixDisplay(cm).plot()
plt.show()
```

```
Precisão: 54.8105%
```



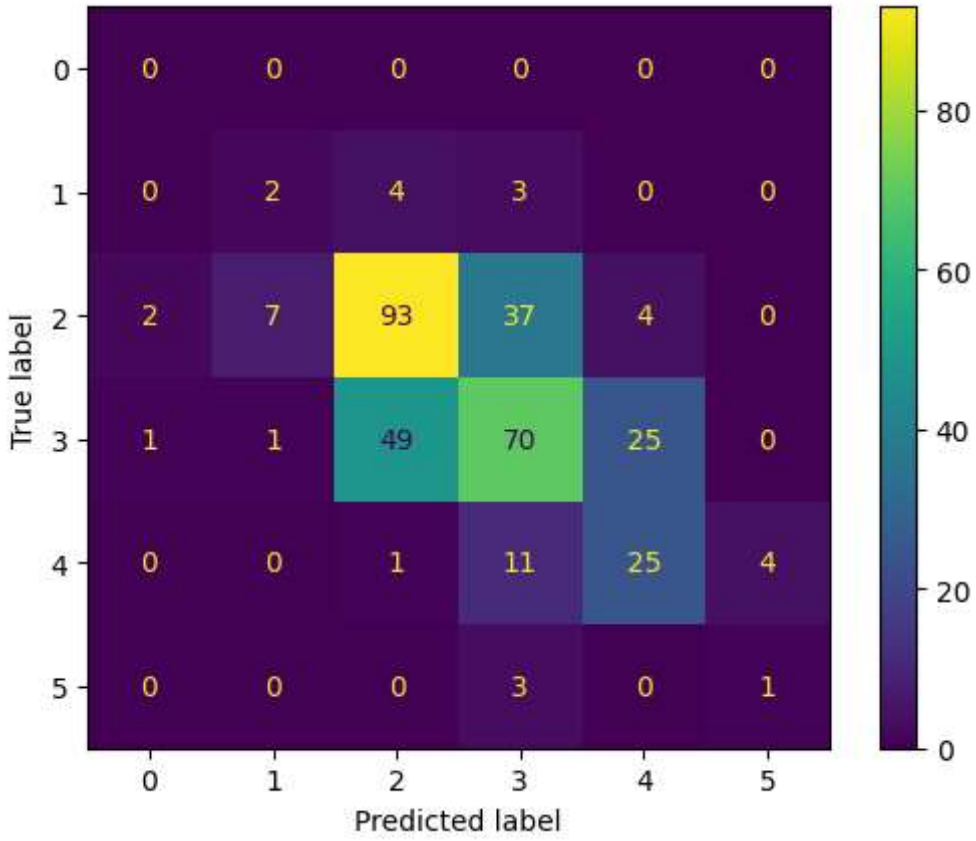
Aplicando DecTree

In [ ]:
```
dectree_scaled = DecisionTreeClassifier(criterion="entropy")
dectree_scaled.fit(X_train_scaled,y_train_scaled)
y_pred_dt_scaled = dectree_scaled.predict(X_test_scaled)
dt_acc_scaled = accuracy_score(y_test_scaled, y_pred_dt_scaled)

print(f'Precisão: {dt_acc_scaled:.4%}')

cm = confusion_matrix(y_test, y_pred_dt_scaled)
disp = ConfusionMatrixDisplay(cm).plot()
plt.show()
```

```
Precisão: 55.6851%
```



Comparando os desempenhos

In [ ]:
```
print(f'    KNN sem normalização: {knn_acc:.2%}')
print(f'    KNN com normalização: {knn_acc_scaled:.2%}')
print(f'DecTree sem normalização: {dt_acc:.2%}')
print(f'DecTree com normalização: {dt_acc_scaled:.2%}')
```

```
    KNN sem normalização: 52.77%
    KNN com normalização: 54.81%
DecTree sem normalização: 55.98%
DecTree com normalização: 55.69%
```

Verificando os principais influenciadores

In [ ]:
```
corr_vals = [abs(x) for x in list(data.corr()['quality'])][:-1]
```

In [ ]:
```
main_inf = []
top_n = sorted(corr_vals)[8]
for idx, val in enumerate(corr_vals):
    if val < top_n:
        main_inf.append(list(data.columns)[idx])
```

In [ ]:
```
X_small = data[main_inf].values
y = data['quality'].values
```

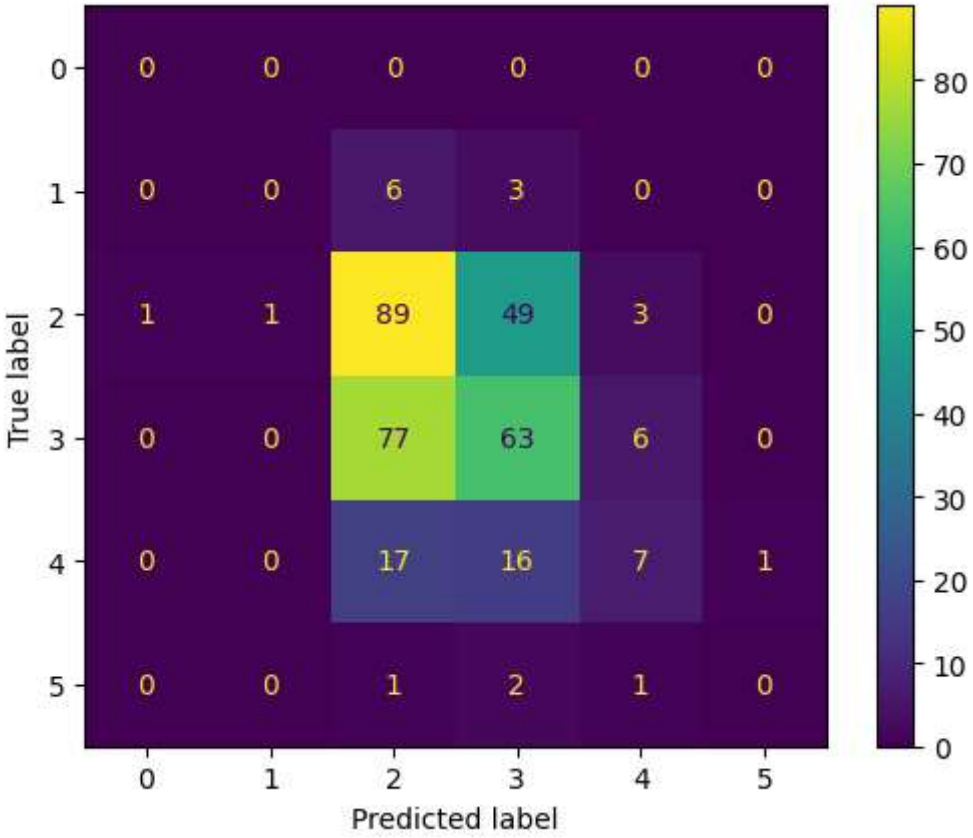
In [ ]:
```
smX_train, smX_test, smy_train, smy_test = train_test_split(X_small,y,train_size
```

In [ ]:
```
smknn = KNeighborsClassifier(n_neighbors=5, metric = 'euclidean')
smknn.fit(smX_train, smy_train)
```

```
smy_pred_knn = smknn.predict(smX_test)
smknn_acc = accuracy_score(smy_test, smy_pred_knn)
print(f'Precisão: {smknn_acc:.4%}')
print(f'Precisão Original: {knn_acc:.4%}')

smcm = confusion_matrix(smy_test, smy_pred_knn)
smdisp = ConfusionMatrixDisplay(smcm).plot()
plt.show()
```

```
Precisão: 46.3557%
Precisão Original: 52.7697%
```

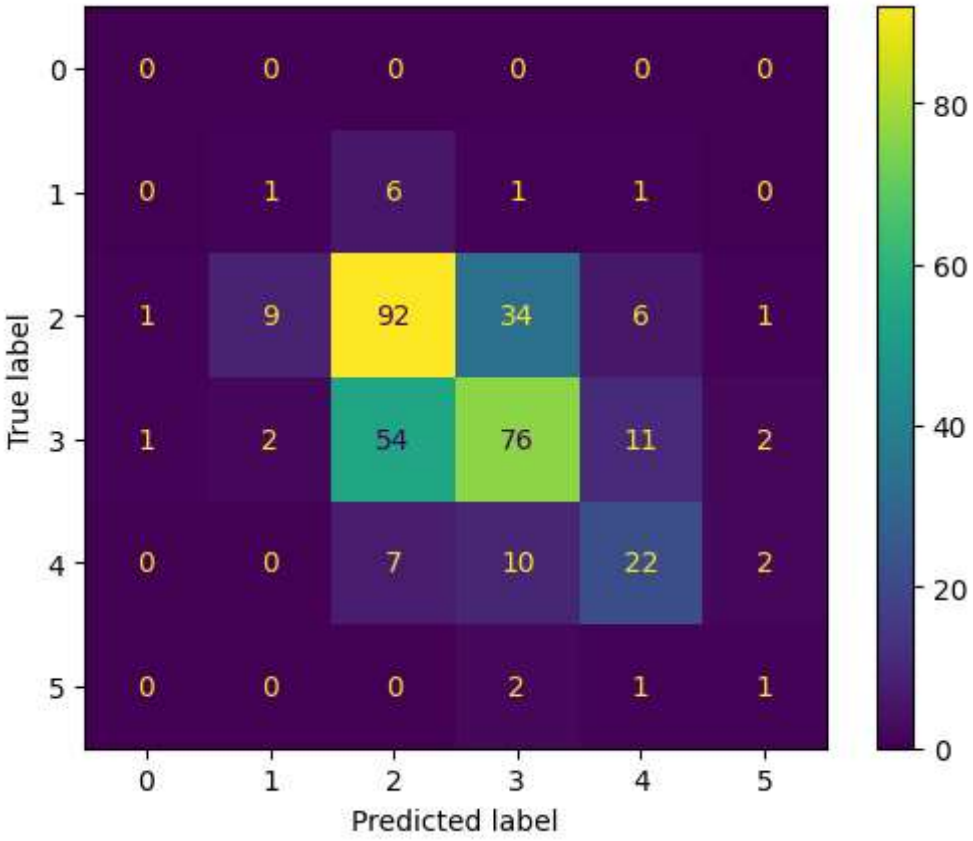


In [ ]:
```
smdectree = DecisionTreeClassifier(criterion="entropy")
smdectree.fit(smX_train, smy_train)
smy_pred_dt = smdectree.predict(smX_test)
smdt_acc = accuracy_score(smy_test, smy_pred_dt)

print(f'Precisão: {smdt_acc:.4%}')
print(f'Precisão Original: {dt_acc:.4%}')

smcm = confusion_matrix(smy_test, smy_pred_dt)
smdisp = ConfusionMatrixDisplay(smcm).plot()
plt.show()
```

```
Precisão: 55.9767%
Precisão Original: 55.9767%
```


0 0 0 0 0 0
0 1 6 1 1 0
1 9 92 34 6 1
1 2 54 76 11 2
0 0 7 10 22 2
0 0 0 2 1 1
True label
Predicted label
0 1 2 3 4 5
0 20 40 60 80