Red Wine Quality data

Name : Naseem mohammed

ID : 455816136

Research Problem:

Analyzing the physicochemical and sensory variables of red variants of Portuguese "Vinho Verde" wine to understand the factors that contribute to the quality of the wine. The research aims to identify the key physicochemical properties that influence the sensory output variables and develop a model that can accurately predict the quality of red wine based on these input variables. The goal is to provide insights into the relationship between the physicochemical characteristics and sensory perception, which can aid in the production and evaluation of high-quality red wine varieties.

In [1]:
```
import pandas as pd
import numpy as np
import sklearn
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
data = pd.read_csv("winequality-red.csv")
```

In [3]:
```
data.head()
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4

In [4]:
```
data.shape
```

Out[4]:
```
(1599, 12)
```

In [5]:
```
data.dtypes #knowledge of data type helps for computation
```

Out[5]:
```
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

In [6]:
```
data.info() #Print a concise summary of a DataFrame.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [7]:
```
data.describe() #helps us to understand how data has been spread across the tab
# count :- the number of NoN-empty rows in a feature.
# mean :- mean value of that feature.
# std :- Standard Deviation Value of that feature.
# min :- minimum value of that feature.
# max :- maximum value of that feature.
# 25%, 50%, and 75% are the percentile/quartile of each features.
```

Out[7]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfu dioxid
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.00000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.46779
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.89532
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.00000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.00000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.00000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.00000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.00000

Data Cleaning

- Dropping duplicate values
- Checking NULL values
- Checking for 0 value and replacing it

In [8]:
```
data=data.drop_duplicates()
```

In [9]:
```
data.isnull().sum()
```

Out[9]:
```
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
dtype: int64
```

In [10]:
```
print(data[data['fixed acidity']==0].shape[0])
print(data[data['volatile acidity']==0].shape[0])
print(data[data['citric acid']==0].shape[0])
print(data[data['residual sugar']==0].shape[0])
print(data[data['chlorides']==0].shape[0])
print(data[data['free sulfur dioxide']==0].shape[0])
print(data[data['total sulfur dioxide']==0].shape[0])
print(data[data['density']==0].shape[0])
print(data[data['pH']==0].shape[0])
print(data[data['sulphates']==0].shape[0])
print(data[data['alcohol']==0].shape[0])
print(data[data['quality']==0].shape[0])
```

```
0
0
118
0
0
0
0
0
0
0
0
0
```

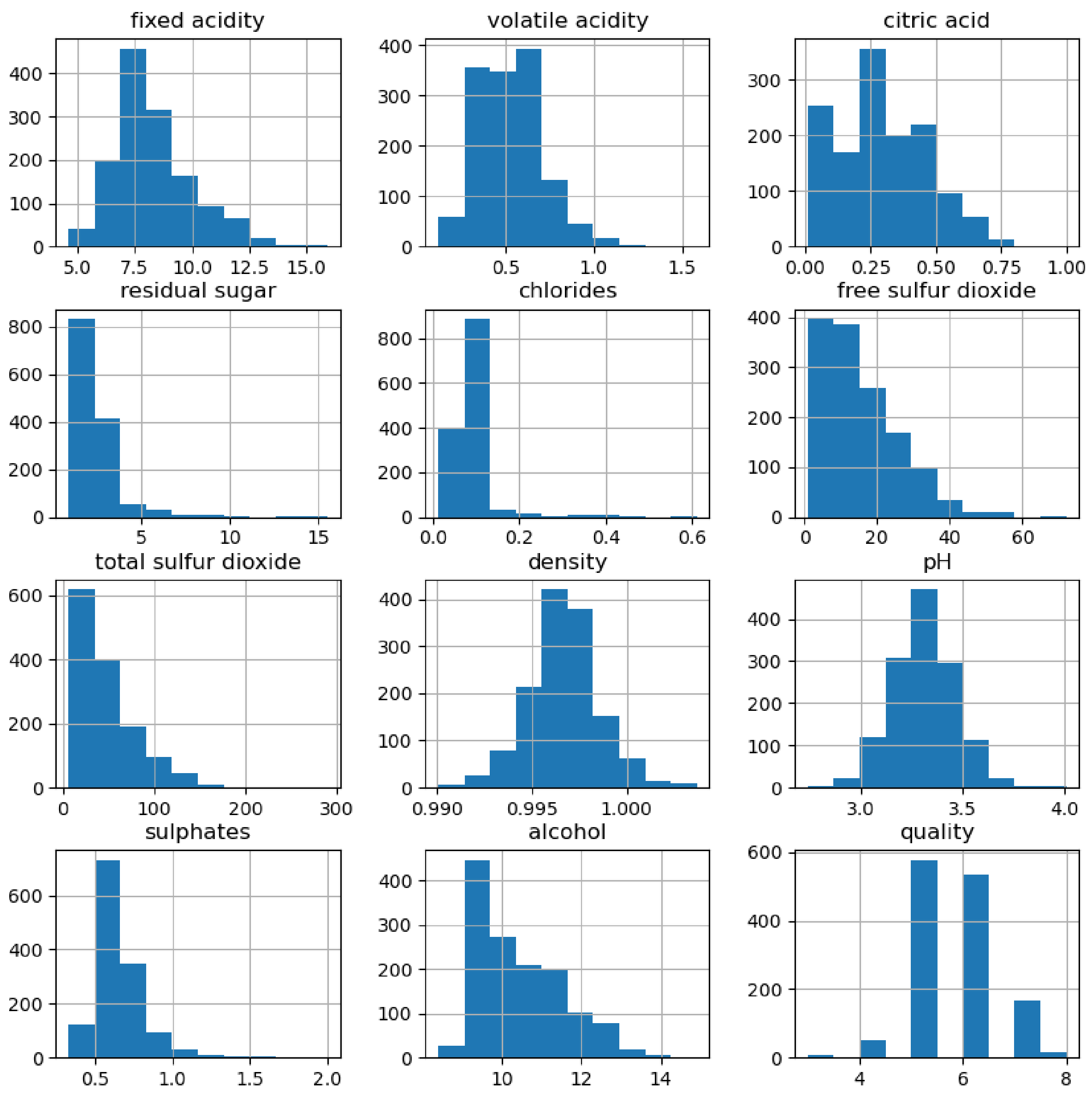
outliers :-

In [11]:
```
data['citric acid']=data['citric acid'].replace(0,data['citric acid'].mean())#r
```

In [12]:
```
data['quality'].unique()
```

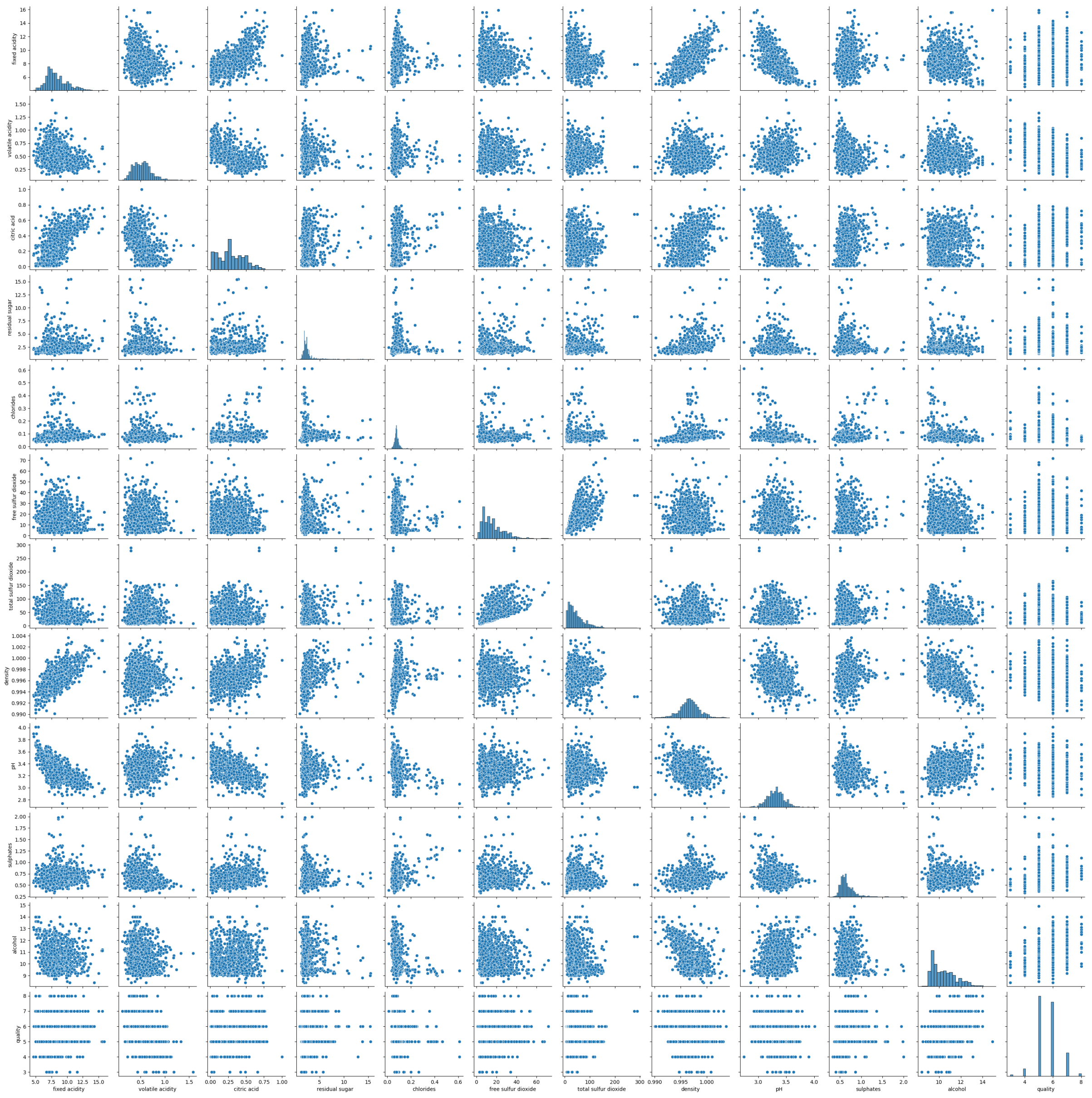
Out[12]:
```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

In [13]:
```
data.hist(bins=10,figsize=(10,10))
plt.show()
```



```
In [14]: #Check correleation between the variables using Seaborn's pairplot.
         sns.pairplot(data)
```

```
Out[14]: <seaborn.axisgrid.PairGrid at 0x1b134ee9130>
```



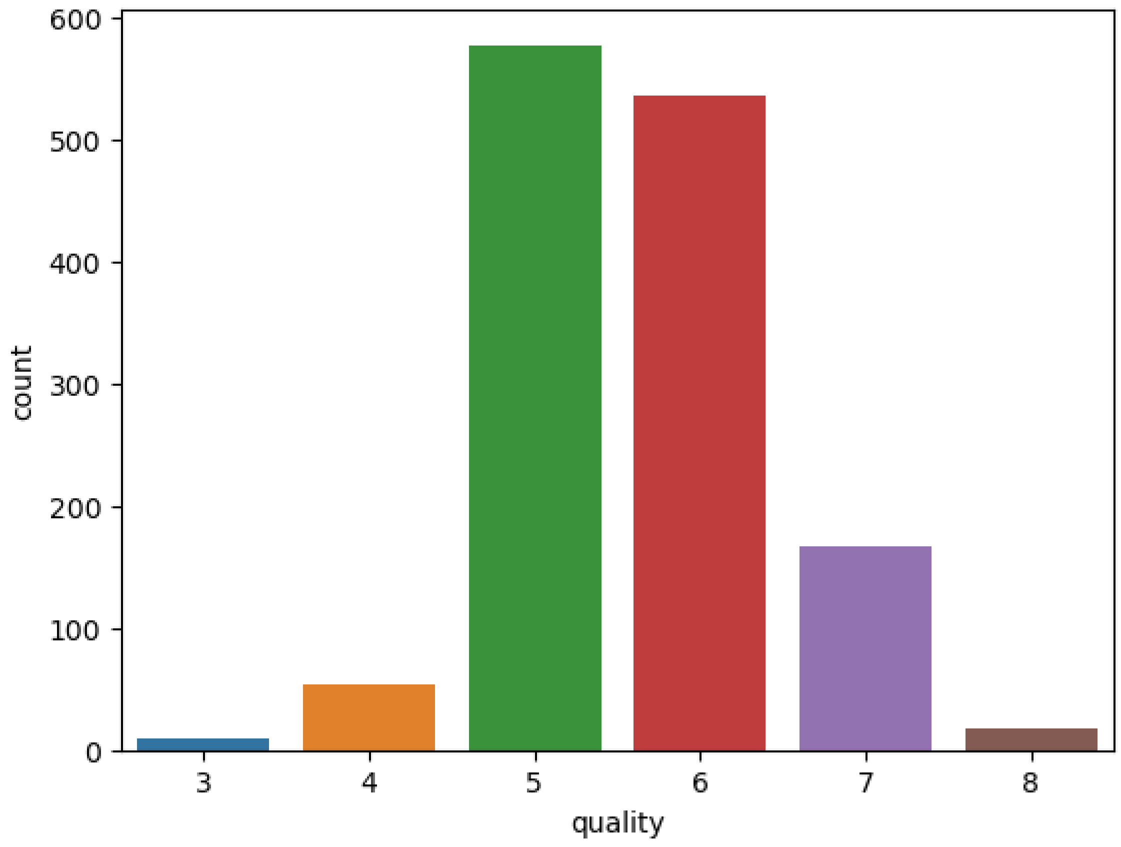
No correlation between the fields as seen on the pairplot

```
In [15]: #count of each target variable
         from collections import Counter
         Counter(data['quality'])
```

```
Out[15]: Counter({5: 577, 6: 535, 7: 167, 4: 53, 8: 17, 3: 10})
```

```
In [16]: #count of the target variable
         sns.countplot(x='quality', data=data)
```

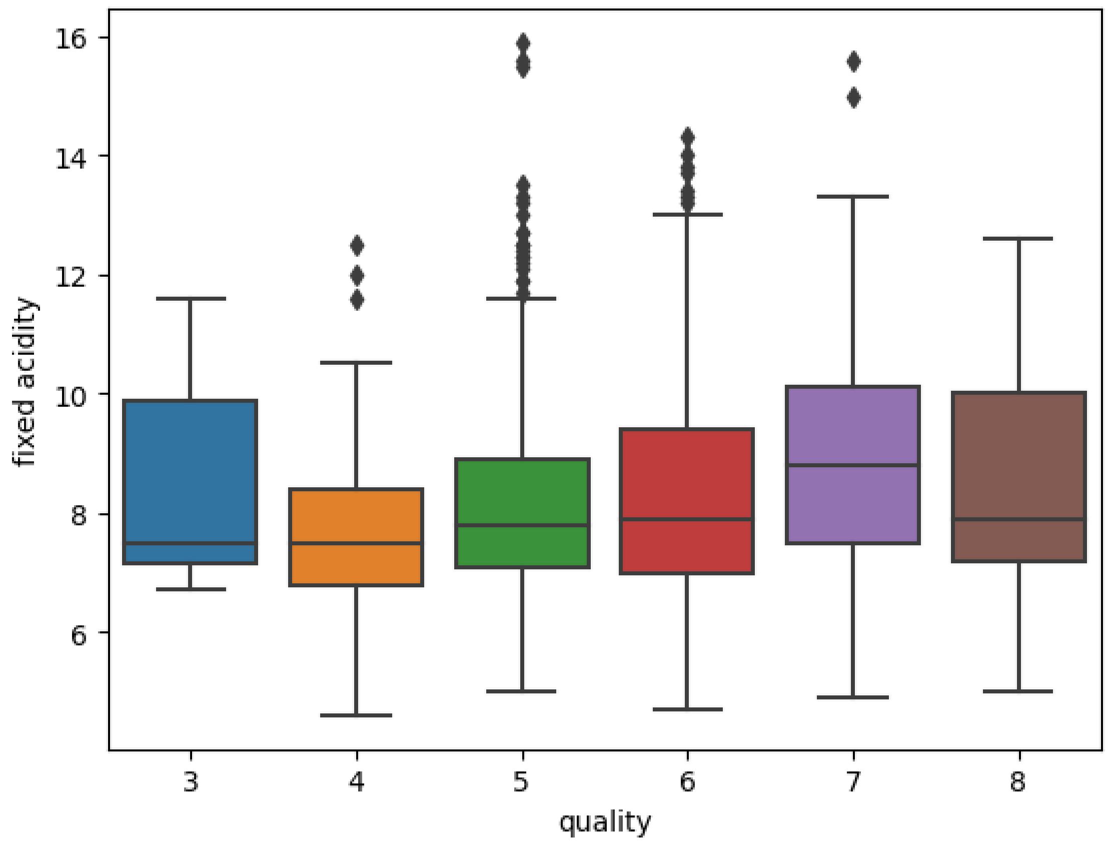
```
Out[16]: <AxesSubplot:xlabel='quality', ylabel='count'>
```



In [17]:
```
#Plot a boxplot to check for Outliers
#Target variable is Quality. So will plot a boxplot each column against target
sns.boxplot('quality', 'fixed acidity', data = data)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variables as keyword args: x, y. From version 0.1
2, the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretatio
n.
  warnings.warn(
```

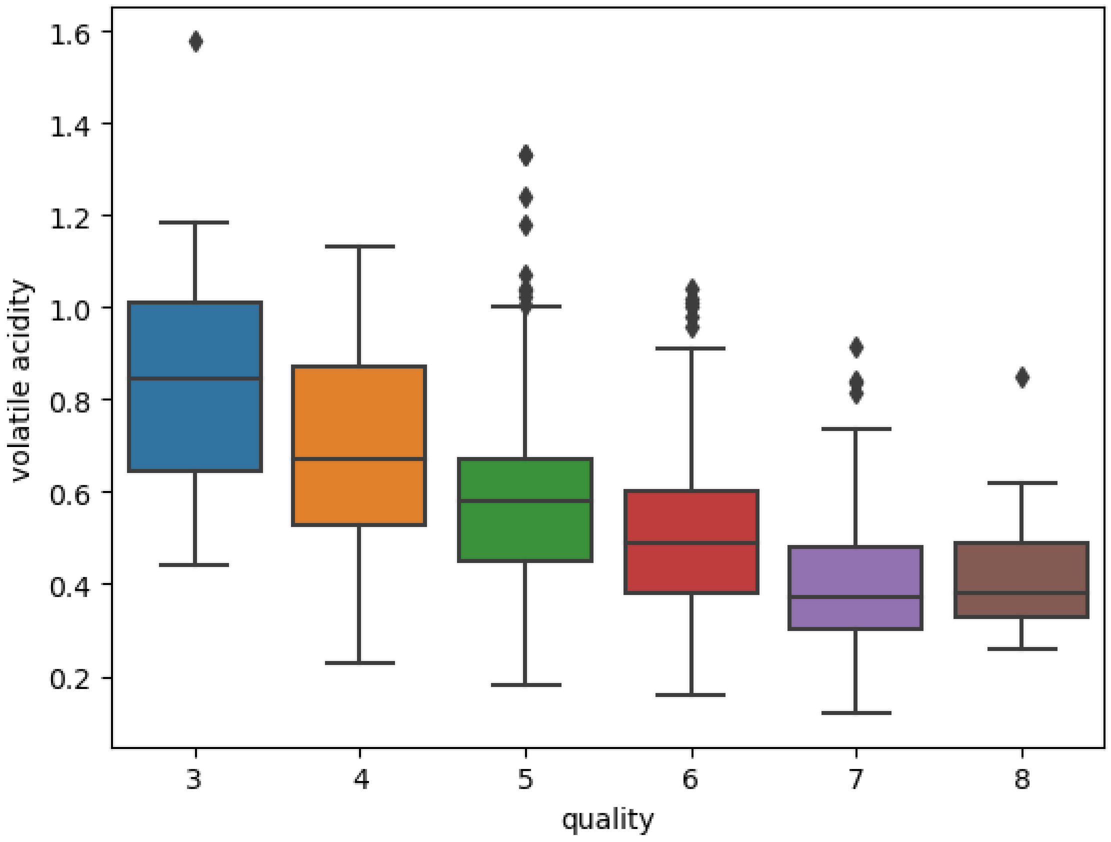
Out[17]:
```
<AxesSubplot:xlabel='quality', ylabel='fixed acidity'>
```



```
In [18]: sns.boxplot('quality', 'volatile acidity', data = data)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variables as keyword args: x, y. From version 0.1
2, the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretatio
n.
  warnings.warn(
```

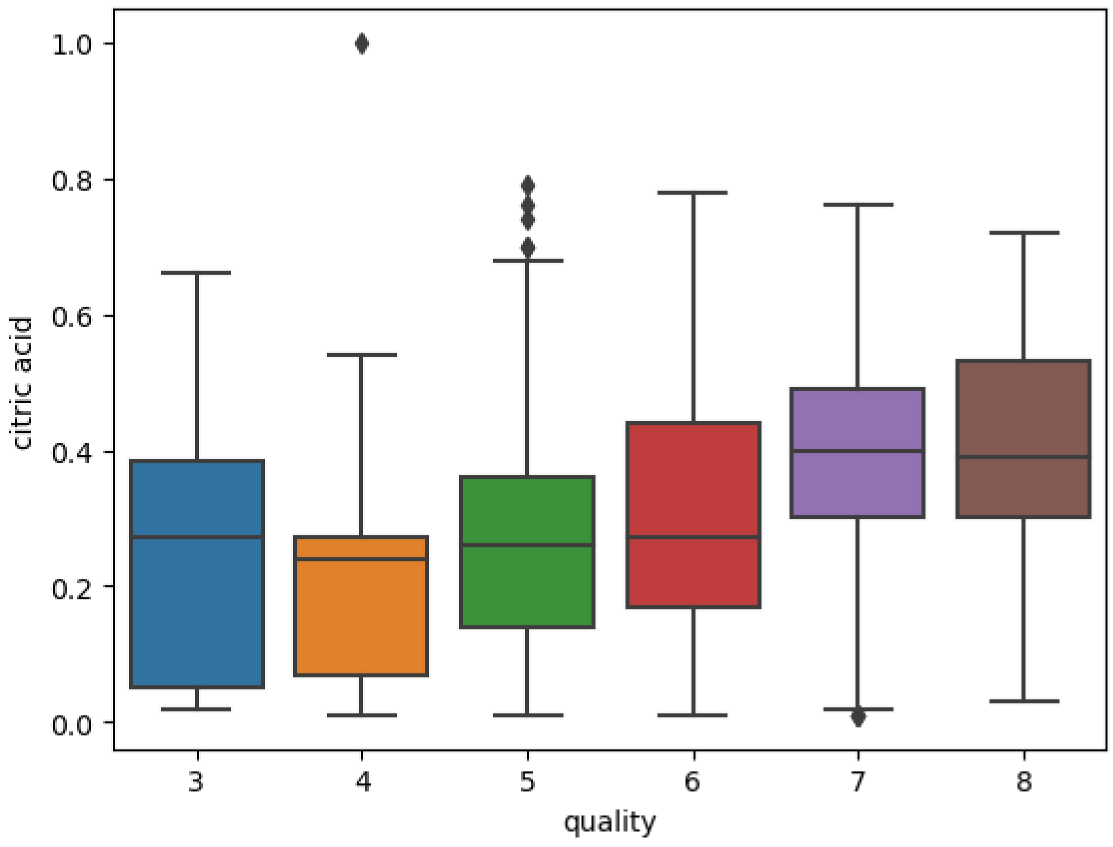
```
Out[18]: <AxesSubplot:xlabel='quality', ylabel='volatile acidity'>
```



```
In [19]: sns.boxplot('quality', 'citric acid', data = data)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variables as keyword args: x, y. From version 0.1
2, the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretatio
n.
  warnings.warn(
```

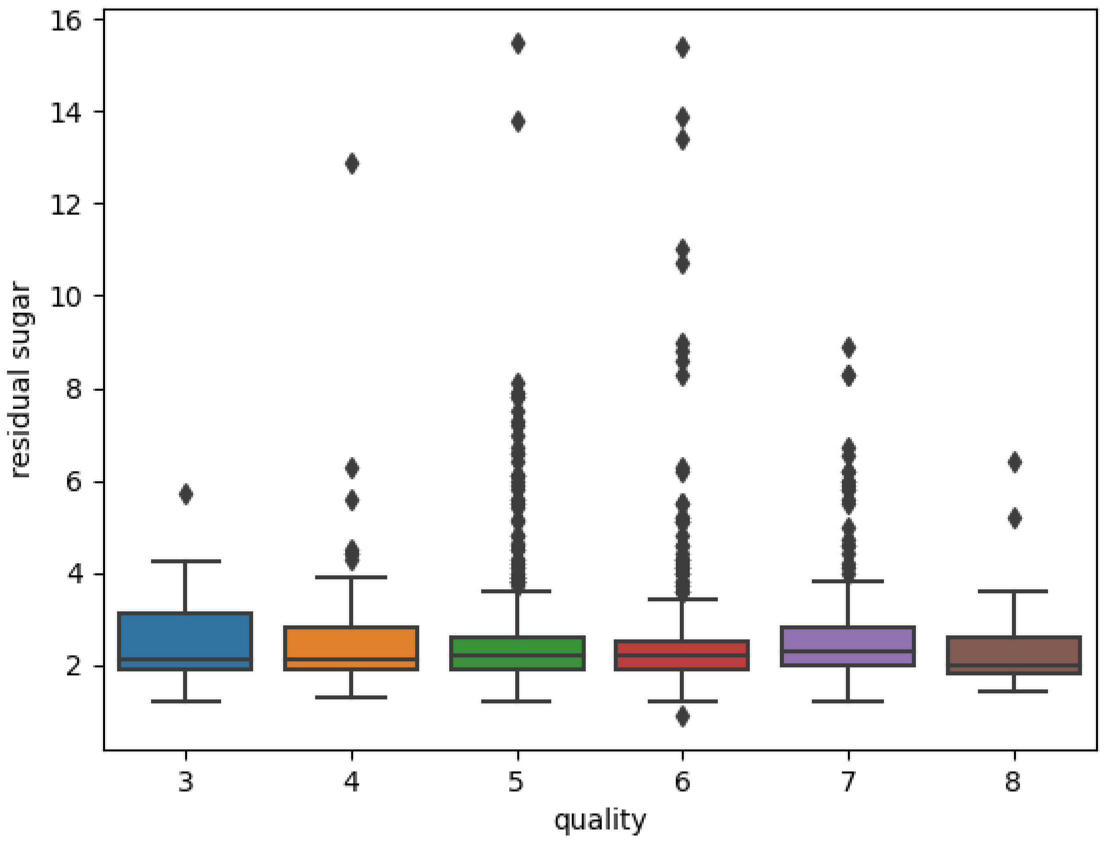
```
Out[19]: <AxesSubplot:xlabel='quality', ylabel='citric acid'>
```



```
In [20]: sns.boxplot('quality', 'residual sugar', data = data)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variables as keyword args: x, y. From version 0.1
2, the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretatio
n.
  warnings.warn(
```

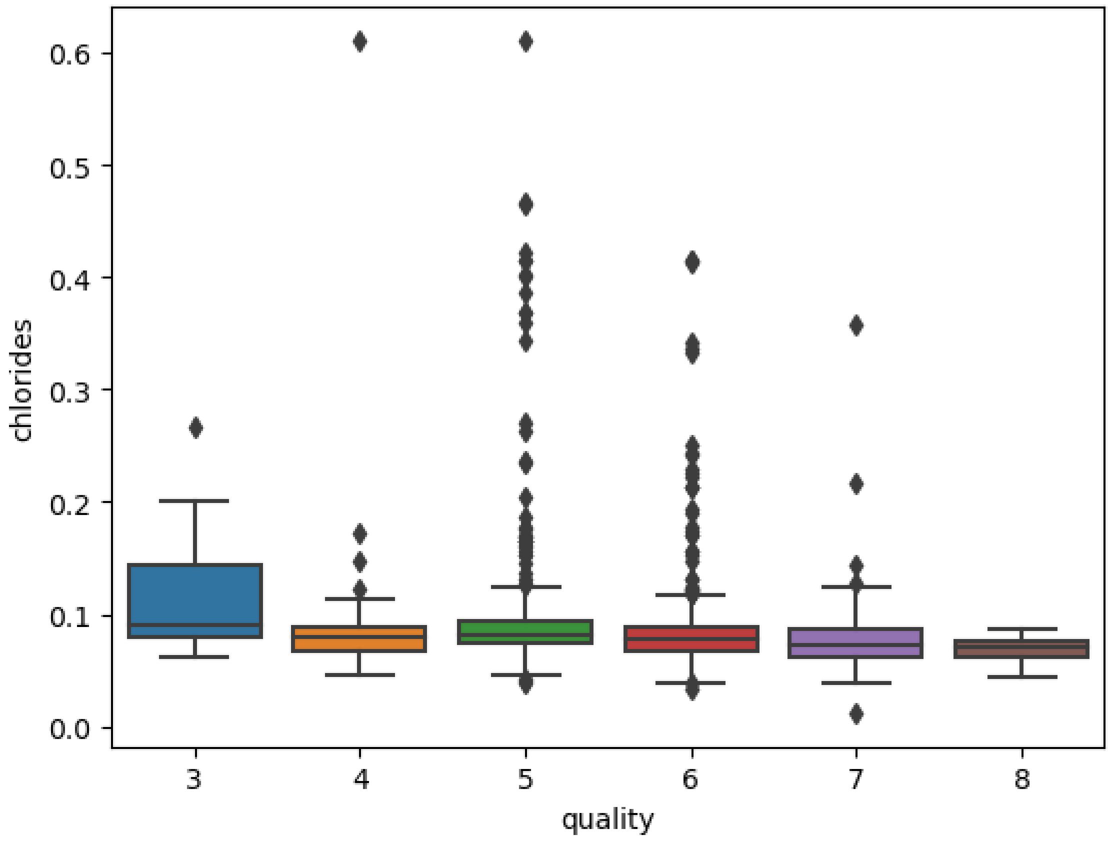
```
Out[20]: <AxesSubplot:xlabel='quality', ylabel='residual sugar'>
```



```
In [21]: sns.boxplot('quality', 'chlorides', data = data)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variables as keyword args: x, y. From version 0.1
2, the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretatio
n.
  warnings.warn(
```

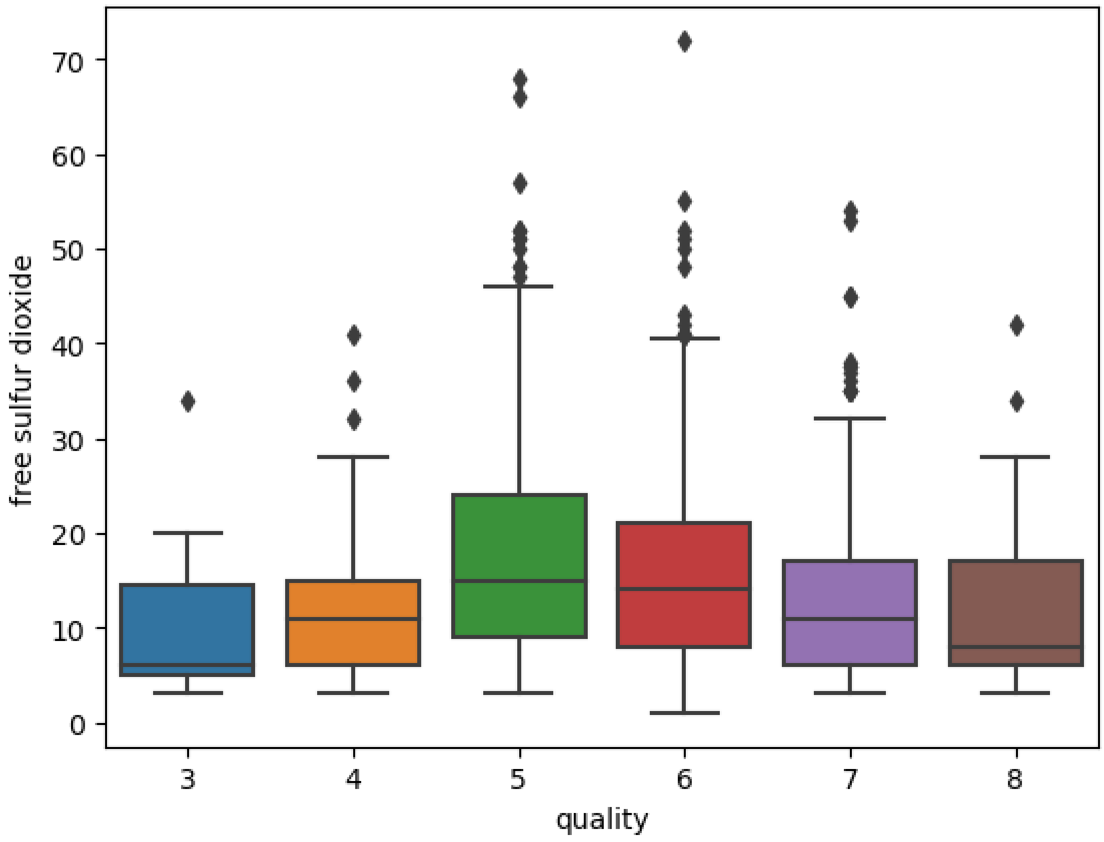
```
Out[21]: <AxesSubplot:xlabel='quality', ylabel='chlorides'>
```



```
In [22]: sns.boxplot('quality', 'free sulfur dioxide', data = data)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variables as keyword args: x, y. From version 0.1
2, the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretatio
n.
  warnings.warn(
```

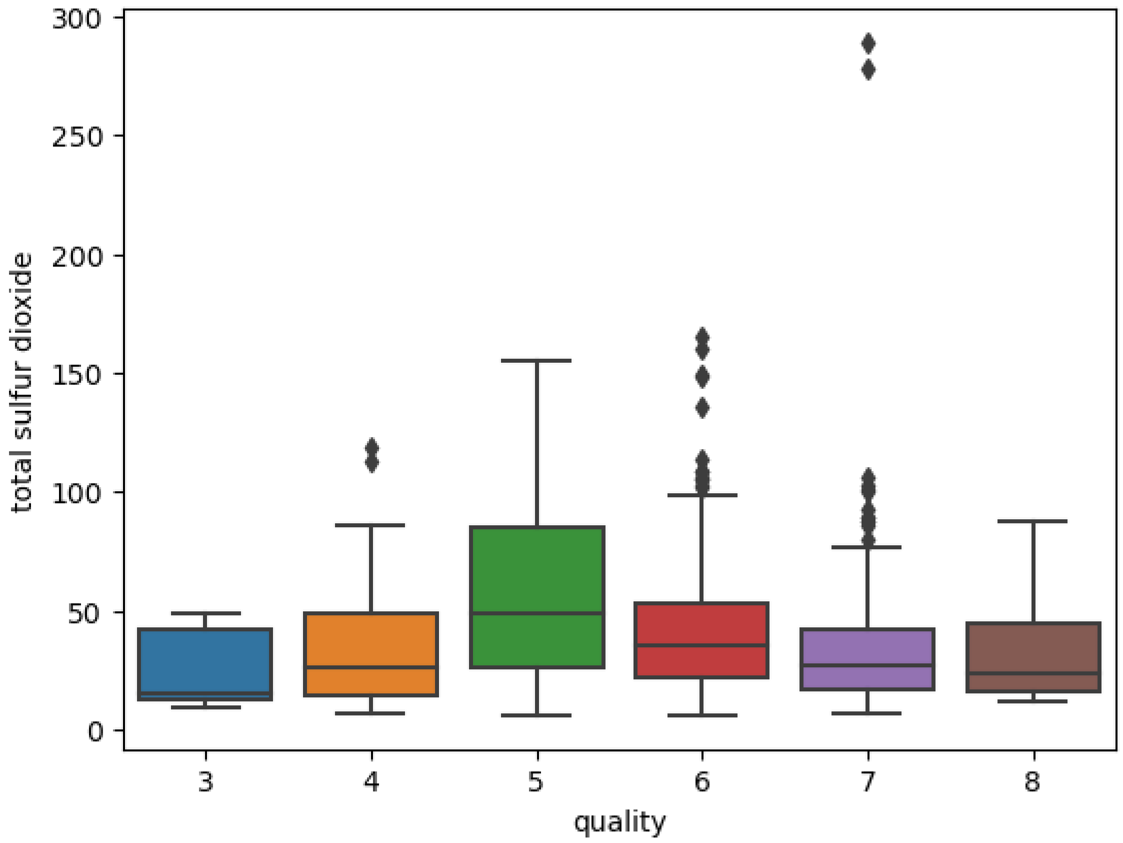
```
Out[22]: <AxesSubplot:xlabel='quality', ylabel='free sulfur dioxide'>
```



```
In [23]: sns.boxplot('quality', 'total sulfur dioxide', data = data)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variables as keyword args: x, y. From version 0.1
2, the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretatio
n.
  warnings.warn(
```

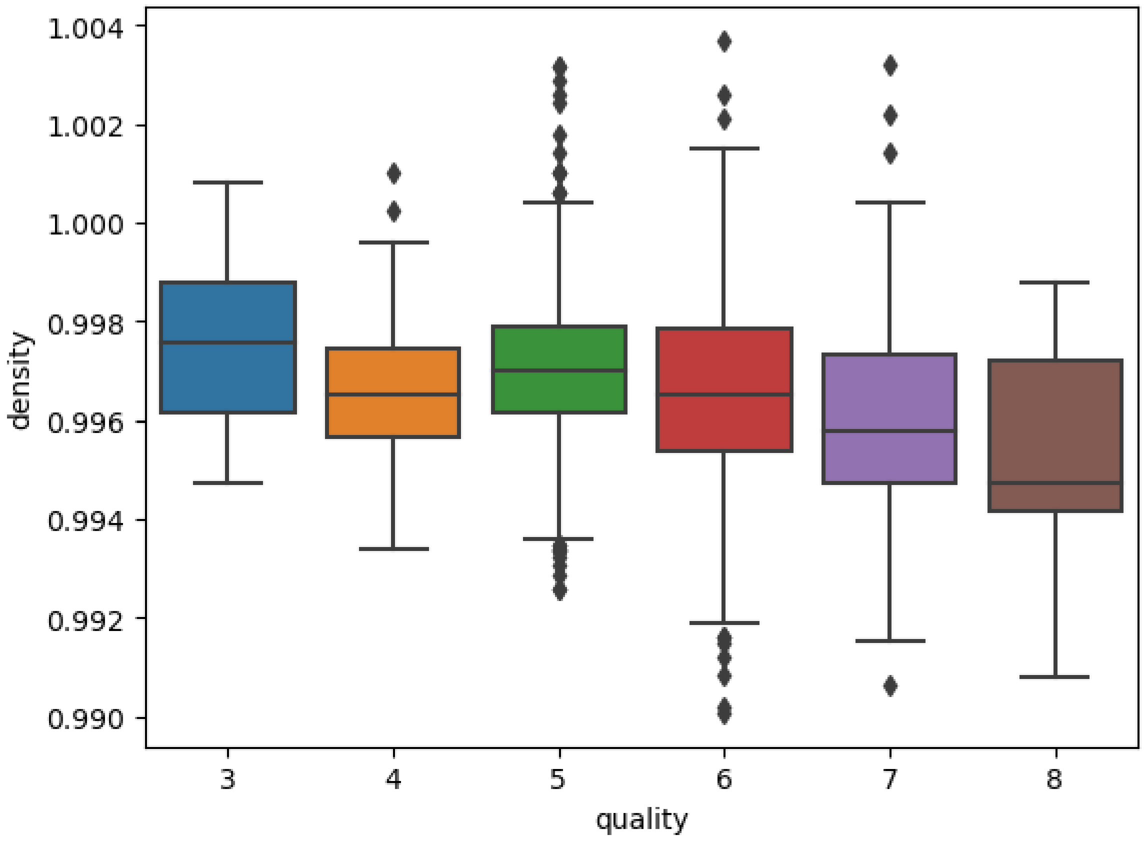
```
Out[23]: <AxesSubplot:xlabel='quality', ylabel='total sulfur dioxide'>
```



In [24]:
```
sns.boxplot('quality', 'density', data = data)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variables as keyword args: x, y. From version 0.1
2, the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretatio
n.
  warnings.warn(
```

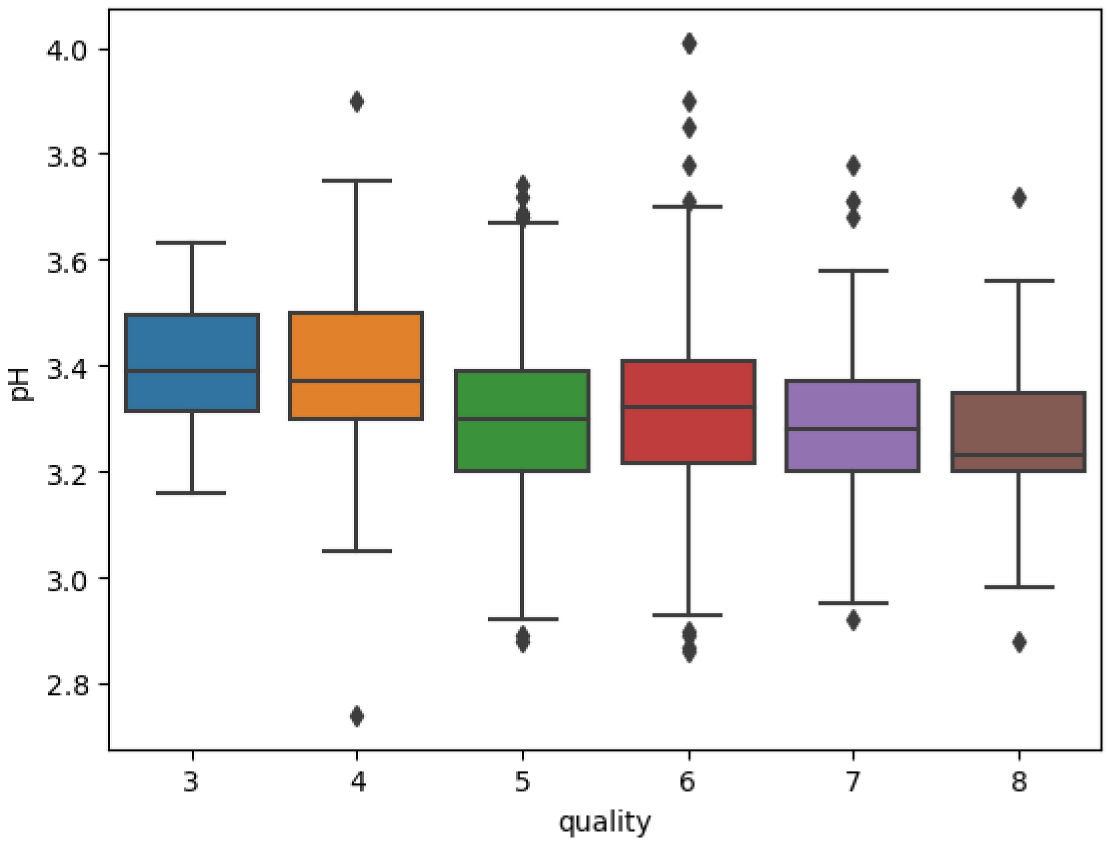
Out[24]:
```
<AxesSubplot:xlabel='quality', ylabel='density'>
```



```
In [25]: sns.boxplot('quality', 'pH', data = data)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variables as keyword args: x, y. From version 0.1
2, the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretatio
n.
  warnings.warn(
```

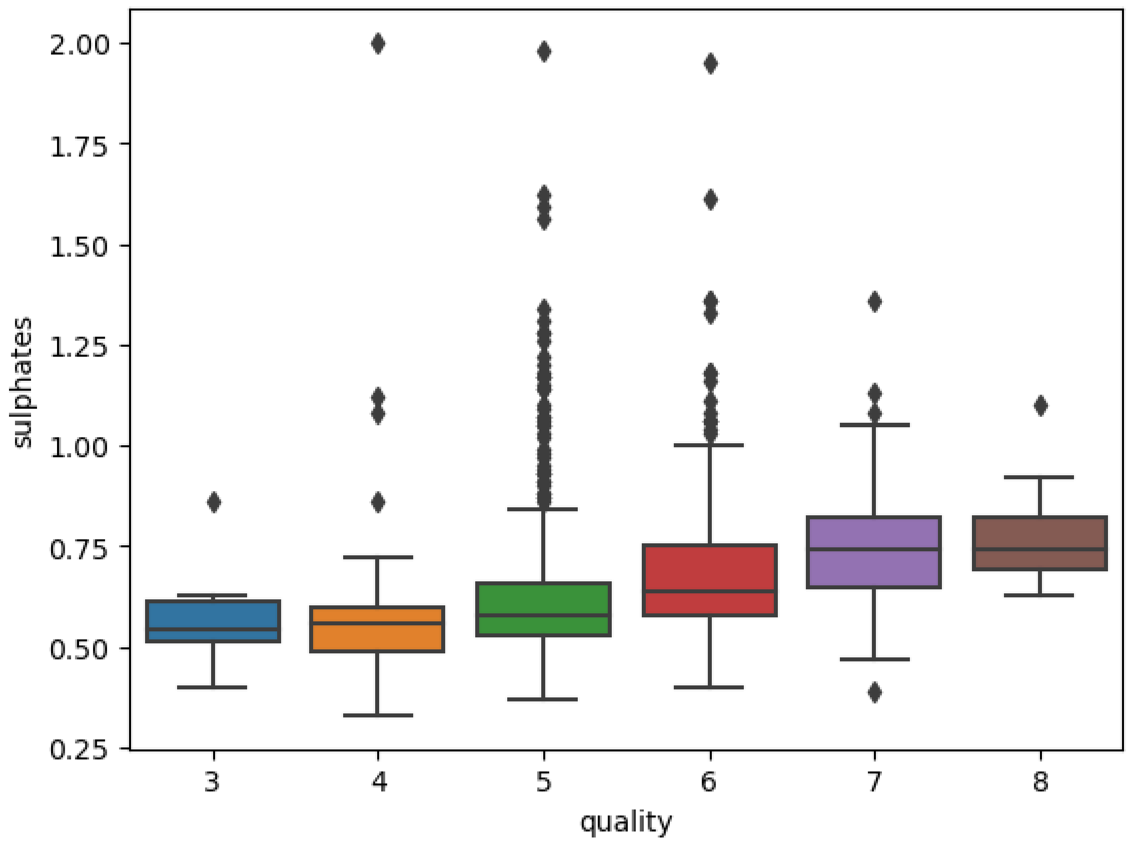
```
Out[25]: <AxesSubplot:xlabel='quality', ylabel='pH'>
```



In [26]:
```
sns.boxplot('quality', 'sulphates', data = data)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variables as keyword args: x, y. From version 0.1
2, the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretatio
n.
  warnings.warn(
```

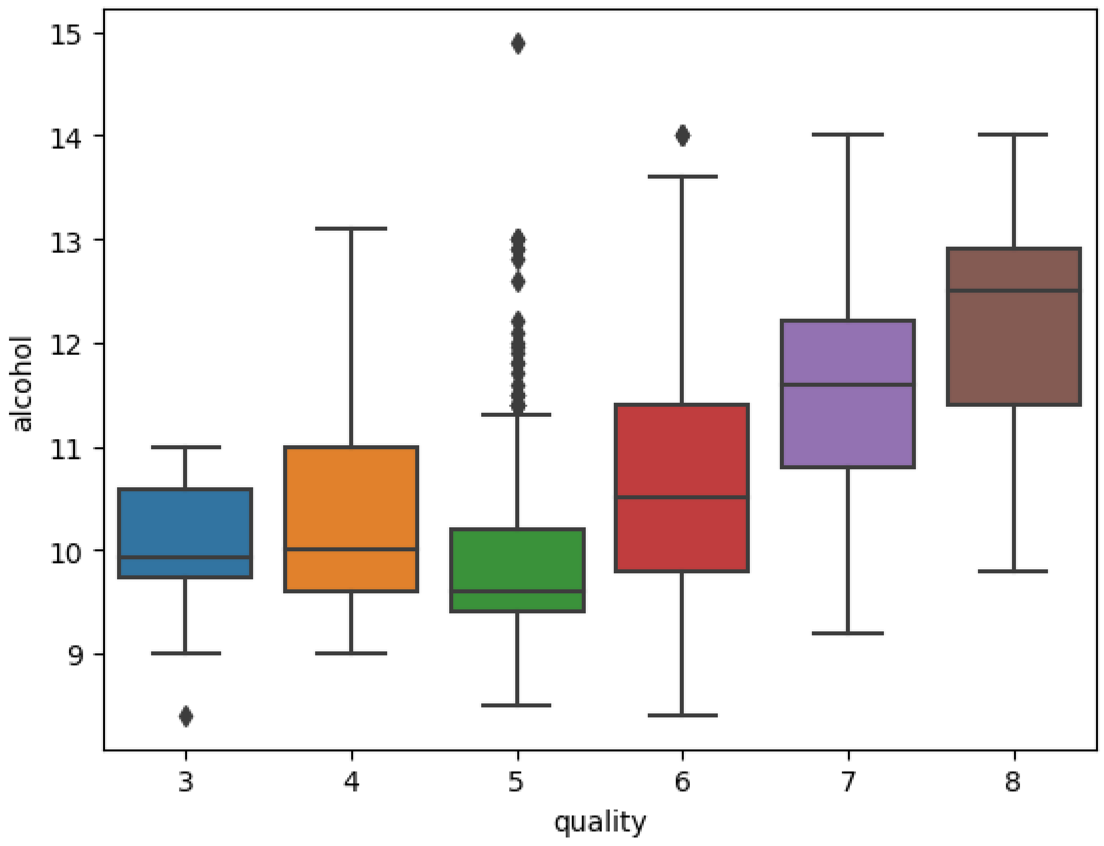
Out[26]:
```
<AxesSubplot:xlabel='quality', ylabel='sulphates'>
```



```
In [27]: sns.boxplot('quality', 'alcohol', data = data)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variables as keyword args: x, y. From version 0.1
2, the only valid positional argument will be `data`, and passing other argum
ents without an explicit keyword will result in an error or misinterpretatio
n.
  warnings.warn(
```

```
Out[27]: <AxesSubplot:xlabel='quality', ylabel='alcohol'>
```



In [28]:
```
#boxplots show many outliers for quite a few columns. Describe the dataset to g
data.describe()
#fixed acidity - 25% - 7.1 and 50% - 7.9. Not much of a variance. Could explain
#volatile acididty - similar reasoning
#citric acid - seems to be somewhat uniformly distributed
#residual sugar - min - 0.9, max - 15!! Waaaaay too much difference. Could expl
#chlorides - same as residual sugar. Min - 0.012, max - 0.611
#free sulfur dioxide, total suflur dioxide - same explanation as above
```

Out[28]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfu dioxid
count	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.00000
mean	8.310596	0.529478	0.295979	2.523400	0.088124	15.893304	46.82597
std	1.736990	0.183031	0.176722	1.352314	0.049377	10.447270	33.40894
min	4.600000	0.120000	0.010000	0.900000	0.012000	1.000000	6.00000
25%	7.100000	0.390000	0.160000	1.900000	0.070000	7.000000	22.00000
50%	7.900000	0.520000	0.272333	2.200000	0.079000	14.000000	38.00000
75%	9.200000	0.640000	0.430000	2.600000	0.091000	21.000000	63.00000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.00000

In [29]:
```
#next we shall create a new column called Review. This column will contain the
#1 - Bad
#2 - Average
#3 - Excellent
#This will be split in the following way.
#1,2,3 --> Bad
#4,5,6,7 --> Average
#8,9,10 --> Excellent
#Create an empty list called Reviews
reviews = []
for i in data['quality']:
    if i >= 1 and i <= 3:
        reviews.append('1')
    elif i >= 4 and i <= 7:
        reviews.append('2')
    elif i >= 8 and i <= 10:
        reviews.append('3')
data['Reviews'] = reviews
```

In [30]:
```
#view final data
data.columns
```

Out[30]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality', 'Reviews'],
      dtype='object')
```

```
In [31]: data['Reviews'].unique()
```

```
Out[31]: array(['2', '3', '1'], dtype=object)
```

```
In [32]: Counter(data['Reviews'])
```

```
Out[32]: Counter({'2': 1332, '3': 17, '1': 10})
```

Split the x and y variables

```
In [33]: x = data.iloc[:,:11]
         y = data['Reviews']
```

```
In [34]: x.head(10)
```

Out[34]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alco
0	7.4	0.70	0.272333	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	
1	7.8	0.88	0.272333	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	
2	7.8	0.76	0.040000	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	
3	11.2	0.28	0.560000	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	
5	7.4	0.66	0.272333	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	
6	7.9	0.60	0.060000	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	
7	7.3	0.65	0.272333	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	
8	7.8	0.58	0.020000	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	
9	7.5	0.50	0.360000	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	
10	6.7	0.58	0.080000	1.8	0.097	15.0	65.0	0.9959	3.28	0.54	

```
In [35]: y.head(10)
```

```
Out[35]: 0     2
         1     2
         2     2
         3     2
         5     2
         6     2
         7     2
         8     2
         9     2
         10    2
         Name: Reviews, dtype: object
```

Now scale the data using StandardScalar for PCA

In [36]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x = sc.fit_transform(x)
```

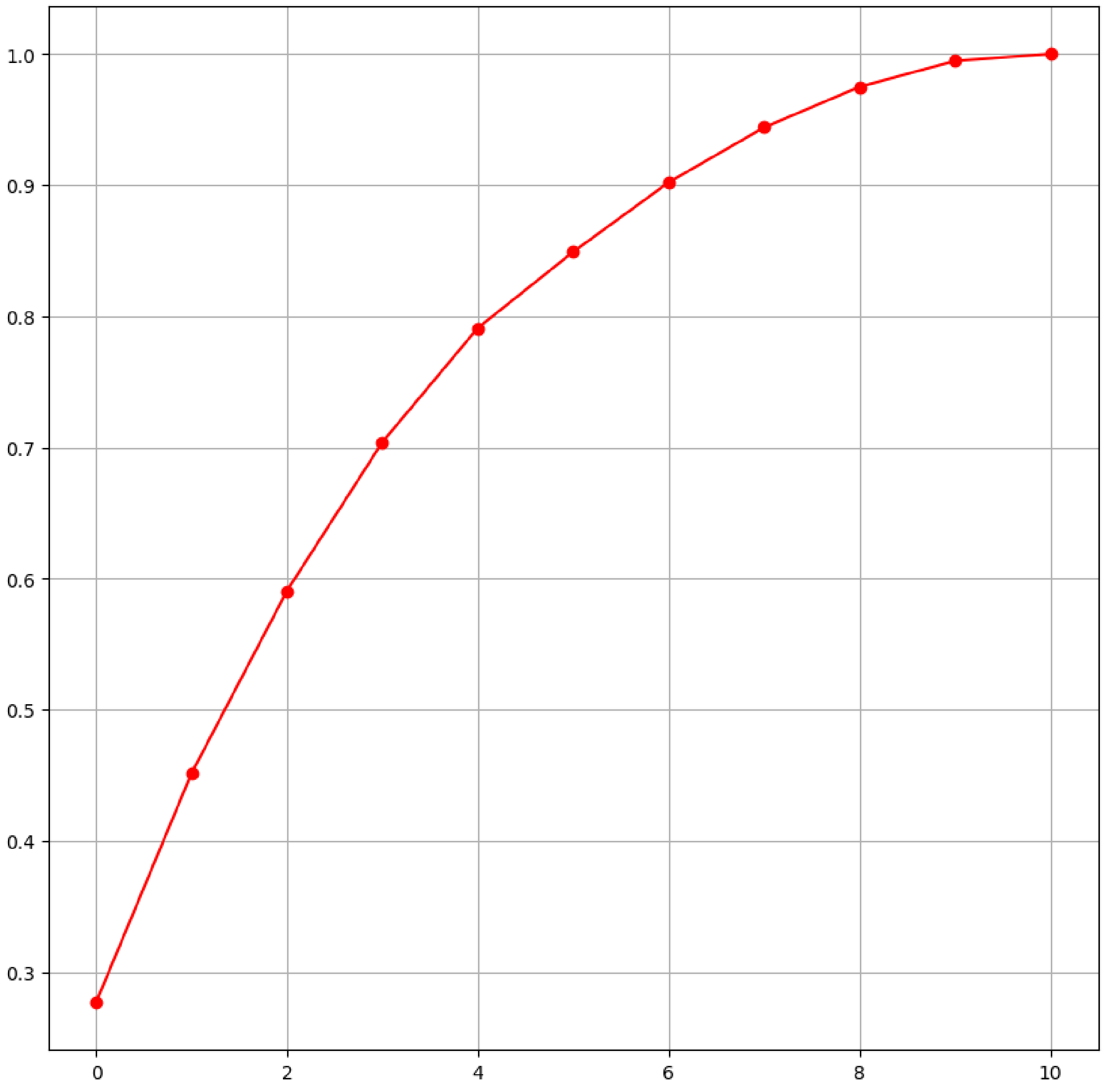
In [37]:
```
#view the scaled features
print(x)
```

```
[[-0.52443096  0.93200015 -0.13385416 ...  1.29187216 -0.57856134
  -0.95437429]
 [-0.29406274  1.91580043 -0.13385416 ... -0.70839548  0.12482157
  -0.5845748 ]
 [-0.29406274  1.25993358 -1.44901794 ... -0.32124691 -0.05102416
  -0.5845748 ]
 ...
 [-1.38831178  0.11216658 -1.10937629 ...  1.35639693  0.59374351
   0.7097234 ]
 [-1.38831178  0.63139451 -0.9961624  ...  1.67902074  0.3006673
  -0.21477532]
 [-1.33071973 -1.19956712  0.98508055 ...  0.51757501  0.00759108
   0.52482366]]
```

Proceed to perform PCA

In [38]:
```
from sklearn.decomposition import PCA
pca = PCA()
x_pca = pca.fit_transform(x)
```

In [39]:
```
#plot the graph to find the principal components
plt.figure(figsize=(10,10))
plt.plot(np.cumsum(pca.explained_variance_ratio_), 'ro-')
plt.grid()
```



In [40]:
```
#AS per the graph, we can see that 8 principal components attribute for 90% of
#we shall pick the first 8 components for our prediction.
pca_new = PCA(n_components=8)
x_new = pca_new.fit_transform(x)
```

```
In [41]: print(x_new)
```

```
[[-1.01451179  0.2718353  -1.49889643 ... -0.7960005  -0.18491911
  -0.90347932]
 [-0.17094494  1.64643291 -0.55287971 ...  1.04031827 -0.7990951
   0.22048262]
 [-0.81360287  0.95438859 -1.20221747 ...  0.36996483 -0.43989174
   0.8498475 ]
 ...
 [-2.2996115   0.90549326  1.74243863 ... -0.72329127 -0.70168176
   0.02044653]
 [-2.3445851   0.99575987  0.57537753 ... -0.88629658 -0.47033362
  -0.3465109 ]
 [-0.49386885 -0.5769369   1.57183288 ... -0.58641342  1.1040968
  -0.29859698]]
```

Split the data into train and test data

```
In [43]: from sklearn.model_selection import train_test_split
         x_train, x_test, y_train, y_test = train_test_split(x_new, y, test_size = 0.25)
```

```
In [44]: print(x_train.shape)
         print(y_train.shape)
         print(x_test.shape)
         print(y_test.shape)
```

```
(1019, 8)
(1019,)
(340, 8)
(340,)
```

1. Logistic Regression

```
In [50]: from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import classification_report,confusion_matrix
         from sklearn.metrics import f1_score, precision_score, recall_score,accuracy_sc
         lr = LogisticRegression()
         lr.fit(x_train, y_train)
         lr_pred = lr.predict(x_test)
```

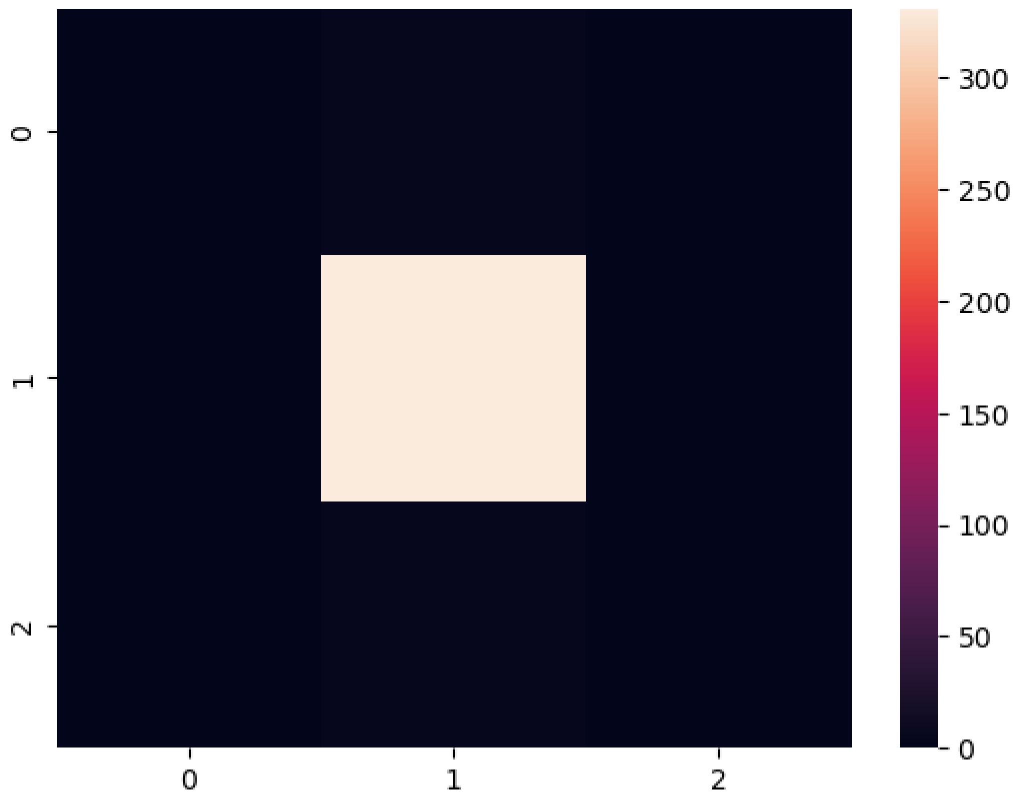
```
In [56]: #print confusion matrix and accuracy score
         lr_conf_matrix = confusion_matrix(y_test, lr_pred)
         lr_acc_score = accuracy_score(y_test, lr_pred)
         print(lr_conf_matrix)
         print(lr_acc_score*100)
```

```
[[  0   4   0]
 [  0 331   0]
 [  0   5   0]]
97.35294117647058
```

```
In [71]: from sklearn.metrics import f1_score, precision_score, recall_score,accuracy_sc

         print("Classification Report is:\n",classification_report(y_test,lr_pred))
         print("\n F1:\n",f1_score(y_test,lr_pred ,average='micro'))
         print("\n Precision score is:\n",precision_score(y_test,lr_pred,average='micro'
         print("\n Recall score is:\n",recall_score(y_test,lr_pred,average='micro'))
         print("\n Confusion Matrix:\n")
         sns.heatmap(confusion_matrix(y_test,lr_pred))
```

```
Classification Report is:
               precision    recall  f1-score   support

           1       0.00      0.00      0.00         4
           2       0.97      1.00      0.99       331
           3       0.00      0.00      0.00         5

    accuracy                           0.97       340
   macro avg       0.32      0.33      0.33       340
weighted avg       0.95      0.97      0.96       340


 F1:
 0.9735294117647059

 Precision score is:
 0.9735294117647059

 Recall score is:
 0.9735294117647059

 Confusion Matrix:


C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Out[71]: <AxesSubplot:>
```



conclusion:

Logistic Regression algorithms was tested on the red wine quality physicochemical properties dataset to predict wine sensory scores. The Logistic regression model achieved the best performance with an accuracy of 97%.

The high F1, precision and recall scores of around 0.97 for the random forest model indicate it is able to accurately classify wines into quality classes (excellent, good, poor etc.) based on objective measurements. This demonstrates that sensory perception of wine quality can be reasonably predicted from chemical attributes alone using machine learning techniques.

Feature importance analysis of the random forest model revealed that attributes like alcohol content, residual sugar, volatile acidity and pH were the strongest determinants of quality ratings. This provides useful insights for winemakers on grape characteristics and winemaking practices that impact sensory quality.

Logistic Regression proved effective in modeling the complex relationship between wine chemistry and organoleptic properties. The models can potentially help optimize viticultural and oenological decisions to consistently produce high quality wines. While the study was limited to a single dataset, the approach shows promise for objective quality assessment and process optimization across different wine regions.

```
In [ ]:
```