```
#WINE QUALITY PREDICTION

#IMPORTING LIBRARIES
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

#LOADING DATASET
import pandas as pd
df=pd.read_csv('WineQT.csv')
print("Successfully Imported data")
df
```

```
Successfully Imported data

      fixed acidity  volatile acidity  citric acid  residual sugar
chlorides  \
0               7.4             0.700         0.00             1.9
0.076
1               7.8             0.880         0.00             2.6
0.098
2               7.8             0.760         0.04             2.3
0.092
3              11.2             0.280         0.56             1.9
0.075
4               7.4             0.700         0.00             1.9
0.076
...             ...               ...          ...             ...
...
1138            6.3             0.510         0.13             2.3
0.076
1139            6.8             0.620         0.08             1.9
0.068
1140            6.2             0.600         0.08             2.0
0.090
1141            5.9             0.550         0.10             2.2
0.062
1142            5.9             0.645         0.12             2.0
0.075

      free sulfur dioxide  total sulfur dioxide  density    pH
sulphates  \
0                    11.0                  34.0  0.99780  3.51
0.56
1                    25.0                  67.0  0.99680  3.20
0.68
2                    15.0                  54.0  0.99700  3.26
0.65
3                    17.0                  60.0  0.99800  3.16
```

```
0.58
4                    11.0                  34.0  0.99780  3.51
0.56
...                   ...                   ...      ...   ...
...
1138                 29.0                  40.0  0.99574  3.42
0.75
1139                 28.0                  38.0  0.99651  3.42
0.82
1140                 32.0                  44.0  0.99490  3.45
0.58
1141                 39.0                  51.0  0.99512  3.52
0.76
1142                 32.0                  44.0  0.99547  3.57
0.71

      alcohol  quality    Id
0         9.4        5     0
1         9.8        5     1
2         9.8        5     2
3         9.8        6     3
4         9.4        5     4
...       ...      ...   ...
1138     11.0        6  1592
1139      9.5        6  1593
1140     10.5        5  1594
1141     11.2        6  1595
1142     10.2        5  1597

[1143 rows x 13 columns]
```

```
df.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
```

```python
print(df.shape)
```

```
(1143, 13)
```

```python
#DESCRIPTION
df.describe()
```

```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    1143.000000       1143.000000  1143.000000     1143.000000
mean        8.311111          0.531339     0.268364        2.532152
std         1.747595          0.179633     0.196686        1.355917
min         4.600000          0.120000     0.000000        0.900000
25%         7.100000          0.392500     0.090000        1.900000
50%         7.900000          0.520000     0.250000        2.200000
75%         9.100000          0.640000     0.420000        2.600000
max        15.900000          1.580000     1.000000       15.500000

         chlorides  free sulfur dioxide  total sulfur dioxide
density  \
count  1143.000000          1143.000000           1143.000000
1143.000000
mean      0.086933            15.615486             45.914698
0.996730
std       0.047267            10.250486             32.782130
0.001925
min       0.012000             1.000000              6.000000
0.990070
25%       0.070000             7.000000             21.000000
0.995570
50%       0.079000            13.000000             37.000000
0.996680
75%       0.090000            21.000000             61.000000
0.997845
max       0.611000            68.000000            289.000000
1.003690

                pH    sulphates      alcohol      quality           Id

count  1143.000000  1143.000000  1143.000000  1143.000000  1143.000000

mean      3.311015     0.657708    10.442111     5.657043   804.969379

std       0.156664     0.170399     1.082196     0.805824   463.997116

min       2.740000     0.330000     8.400000     3.000000     0.000000

25%       3.205000     0.550000     9.500000     5.000000   411.000000
```

```
50%       3.310000    0.620000   10.200000    6.000000   794.000000

75%       3.400000    0.730000   11.100000    6.000000  1209.500000

max       4.010000    2.000000   14.900000    8.000000  1597.000000
```

```
#FINDING NULL VALUES
df.isnull().sum()
```

```
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
```

```
#CORRELATION
df.corr()
```

```
                      fixed acidity  volatile acidity  citric acid  \
fixed acidity              1.000000         -0.250728     0.673157
volatile acidity          -0.250728          1.000000    -0.544187
citric acid                0.673157         -0.544187     1.000000
residual sugar             0.171831         -0.005751     0.175815
chlorides                  0.107889          0.056336     0.245312
free sulfur dioxide       -0.164831         -0.001962    -0.057589
total sulfur dioxide      -0.110628          0.077748     0.036871
density                    0.681501          0.016512     0.375243
pH                        -0.685163          0.221492    -0.546339
sulphates                  0.174592         -0.276079     0.331232
alcohol                   -0.075055         -0.203909     0.106250
quality                    0.121970         -0.407394     0.240821
Id                        -0.275826         -0.007892    -0.139011

                      residual sugar  chlorides  free sulfur
dioxide  \
fixed acidity               0.171831   0.107889            -0.164831

volatile acidity           -0.005751   0.056336            -0.001962

citric acid                 0.175815   0.245312            -0.057589
```

```
residual sugar            1.000000   0.070863             0.165339
chlorides                 0.070863   1.000000             0.015280
free sulfur dioxide       0.165339   0.015280             1.000000
total sulfur dioxide      0.190790   0.048163             0.661093
density                   0.380147   0.208901            -0.054150
pH                       -0.116959  -0.277759             0.072804
sulphates                 0.017475   0.374784             0.034445
alcohol                   0.058421  -0.229917            -0.047095
quality                   0.022002  -0.124085            -0.063260
Id                       -0.046344  -0.088099             0.095268

                      total sulfur dioxide   density        pH
sulphates  \
fixed acidity                    -0.110628  0.681501 -0.685163
0.174592
volatile acidity                  0.077748  0.016512  0.221492  -
0.276079
citric acid                       0.036871  0.375243 -0.546339
0.331232
residual sugar                    0.190790  0.380147 -0.116959
0.017475
chlorides                         0.048163  0.208901 -0.277759
0.374784
free sulfur dioxide               0.661093 -0.054150  0.072804
0.034445
total sulfur dioxide              1.000000  0.050175 -0.059126
0.026894
density                           0.050175  1.000000 -0.352775
0.143139
pH                               -0.059126 -0.352775  1.000000  -
0.185499
sulphates                         0.026894  0.143139 -0.185499
1.000000
alcohol                          -0.188165 -0.494727  0.225322
0.094421
quality                          -0.183339 -0.175208 -0.052453
0.257710
Id                               -0.107389 -0.363926  0.132904  -
0.103954
```

```
                      alcohol   quality        Id
fixed acidity        -0.075055  0.121970 -0.275826
volatile acidity     -0.203909 -0.407394 -0.007892
citric acid           0.106250  0.240821 -0.139011
residual sugar        0.058421  0.022002 -0.046344
chlorides            -0.229917 -0.124085 -0.088099
free sulfur dioxide  -0.047095 -0.063260  0.095268
total sulfur dioxide -0.188165 -0.183339 -0.107389
density              -0.494727 -0.175208 -0.363926
pH                    0.225322 -0.052453  0.132904
sulphates             0.094421  0.257710 -0.103954
alcohol               1.000000  0.484866  0.238087
quality               0.484866  1.000000  0.069708
Id                    0.238087  0.069708  1.000000
```

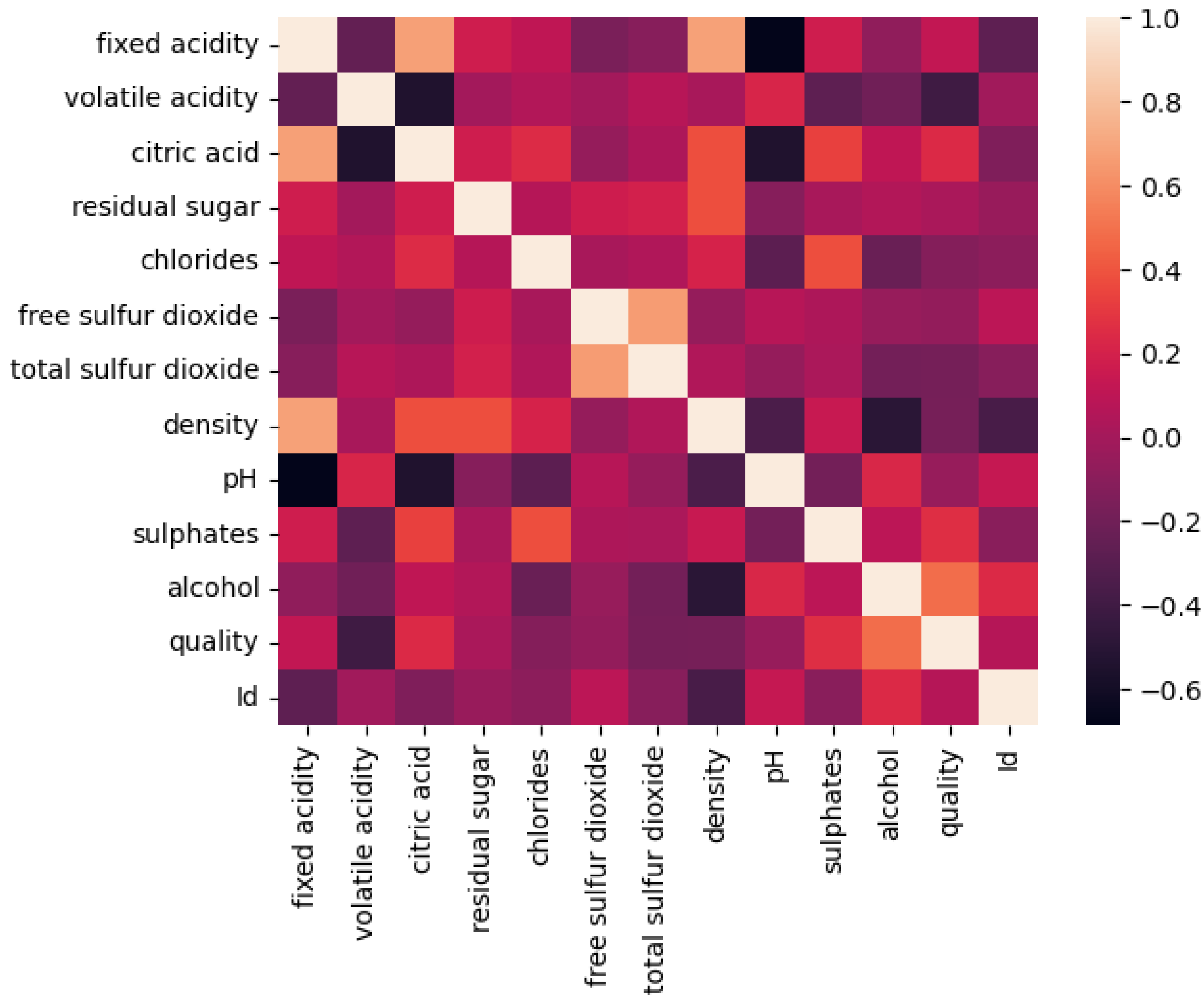
```
df.groupby('quality').mean()
```

```
         fixed acidity  volatile acidity  citric acid  residual sugar  
\
quality                                                                 

3             8.450000          0.897500     0.211667        2.666667   

4             7.809091          0.700000     0.165758        2.566667   

5             8.161077          0.585280     0.240124        2.540476   

6             8.317749          0.504957     0.263680        2.444805   

7             8.851049          0.393671     0.386573        2.760140   

8             8.806250          0.410000     0.432500        2.643750   


         chlorides  free sulfur dioxide  total sulfur dioxide  
density  \
quality                                                         

3         0.105333             8.166667             24.500000  
0.997682   
4         0.094788            14.848485             40.606061  
0.996669   
5         0.091770            16.612836             55.299172  
0.997073   
6         0.085281            15.215368             39.941558  
0.996610   
7         0.075217            14.538462             37.489510  
0.996071   
8         0.070187            11.062500             29.375000  
0.995553   
```

```
               pH  sulphates    alcohol           Id
quality
3        3.361667   0.550000   9.691667  1121.166667
4        3.391212   0.637879  10.260606   692.848485
5        3.302091   0.613375   9.902277   753.925466
6        3.323788   0.676537  10.655339   854.625541
7        3.287133   0.743566  11.482634   830.349650
8        3.240625   0.766250  11.937500   797.875000
```

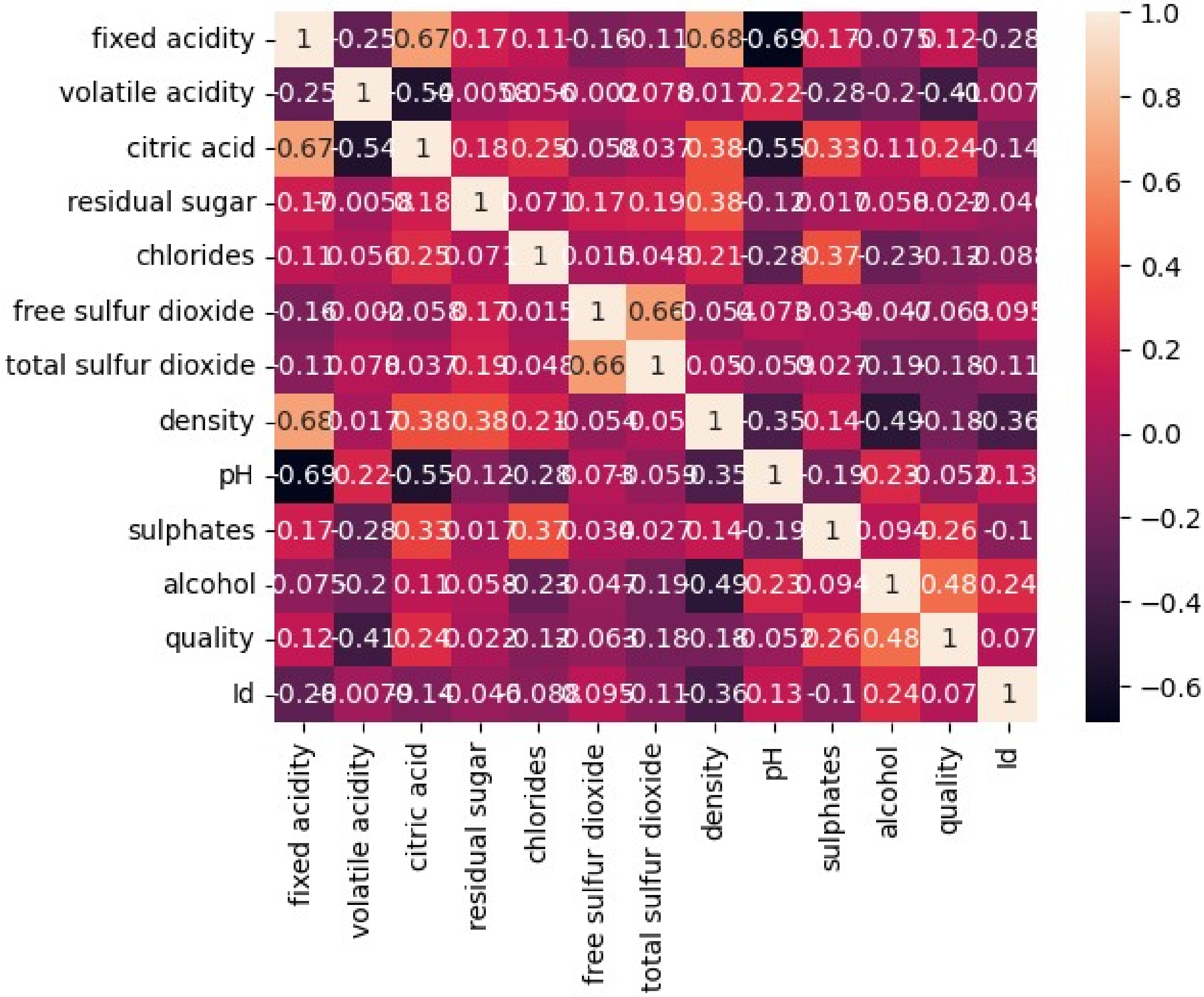
```
#HEATMAP FOR EXPRESSING CORRELATION
import seaborn as sns
sns.heatmap(df.corr())
```

```
<Axes: >
```



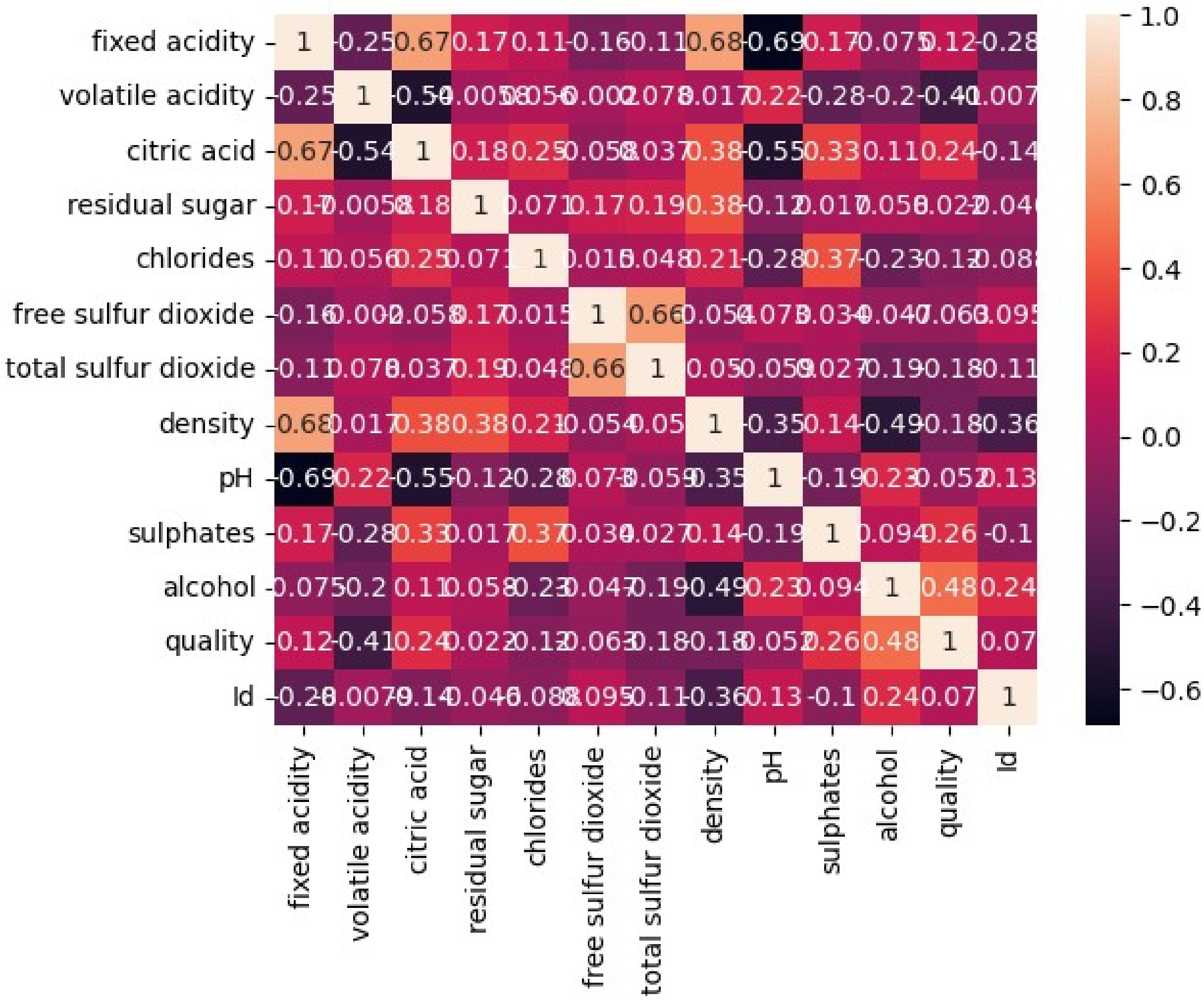

```
import seaborn as sns
sns.heatmap(df.corr(),annot=True)
```

```
<Axes: >
```

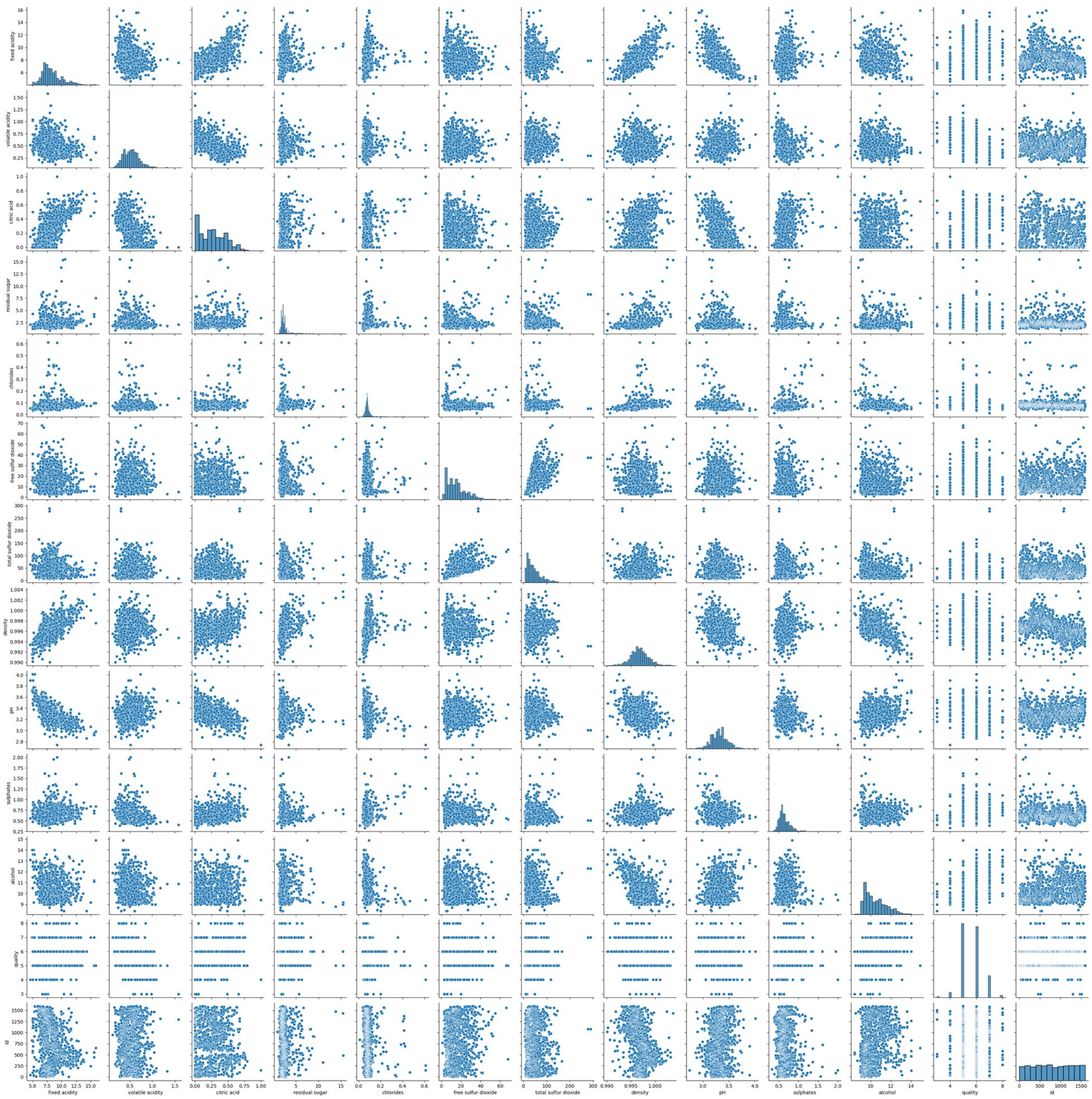



```
df[['fixed acidity','volatile acidity']].corr()
sns.heatmap(df.corr(),annot=True)
```

```
<Axes: >
```

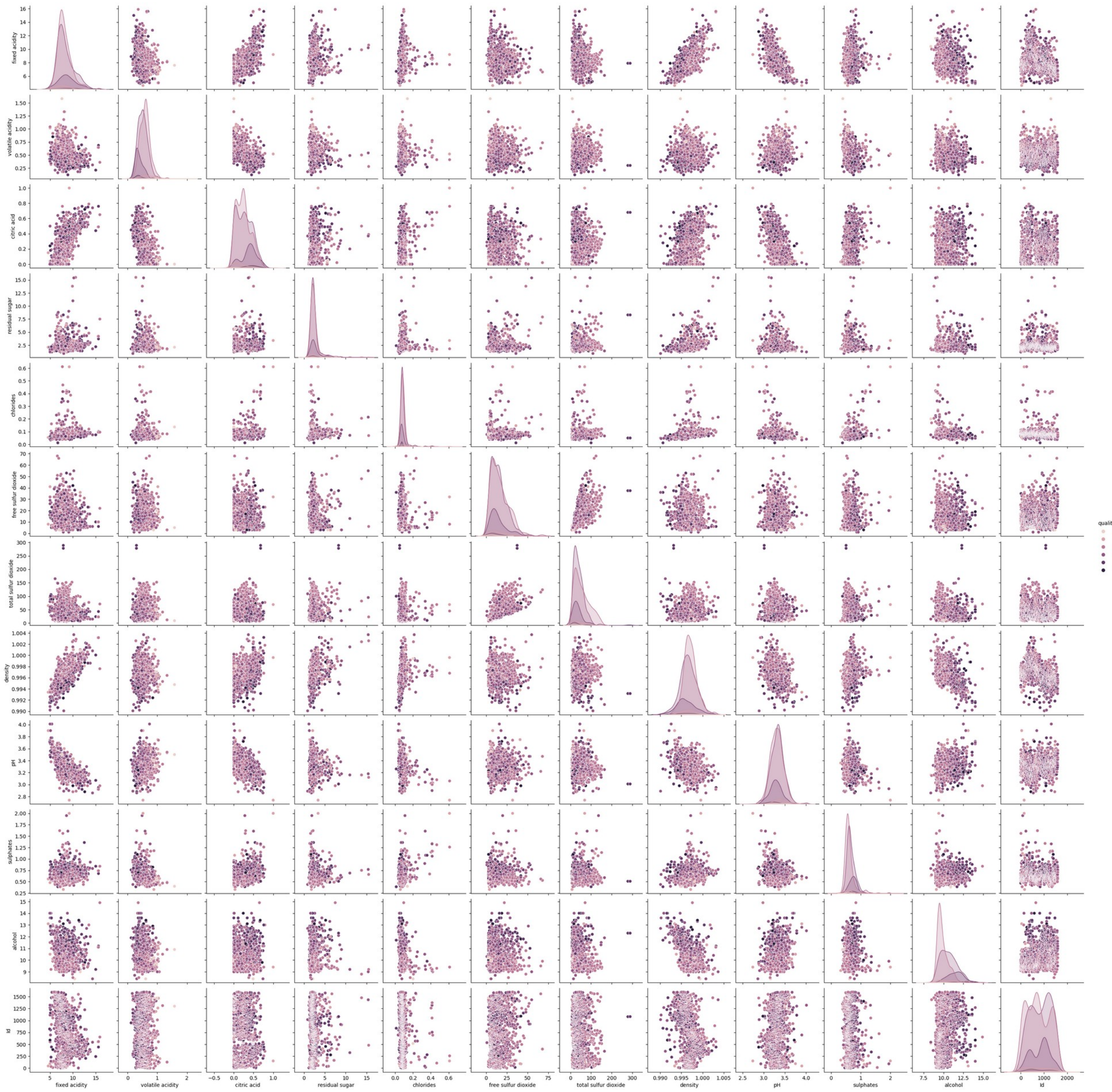



```
#PAIR PLOT
sns.pairplot(df)

<seaborn.axisgrid.PairGrid at 0x259dd2f6e10>
```

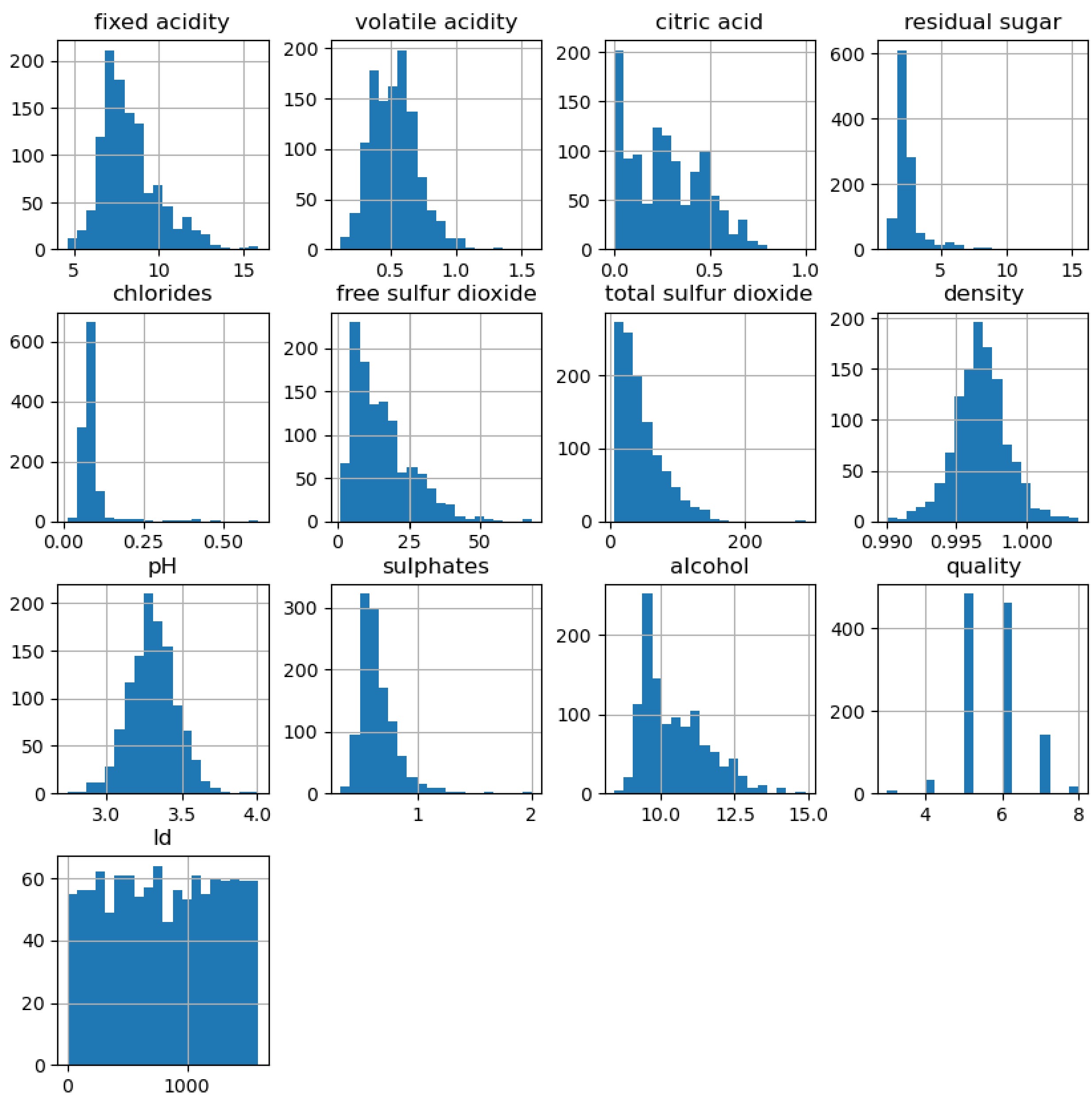



```
sns.pairplot(df,hue='quality')
```

```
<seaborn.axisgrid.PairGrid at 0x26ecd8772d0>
```

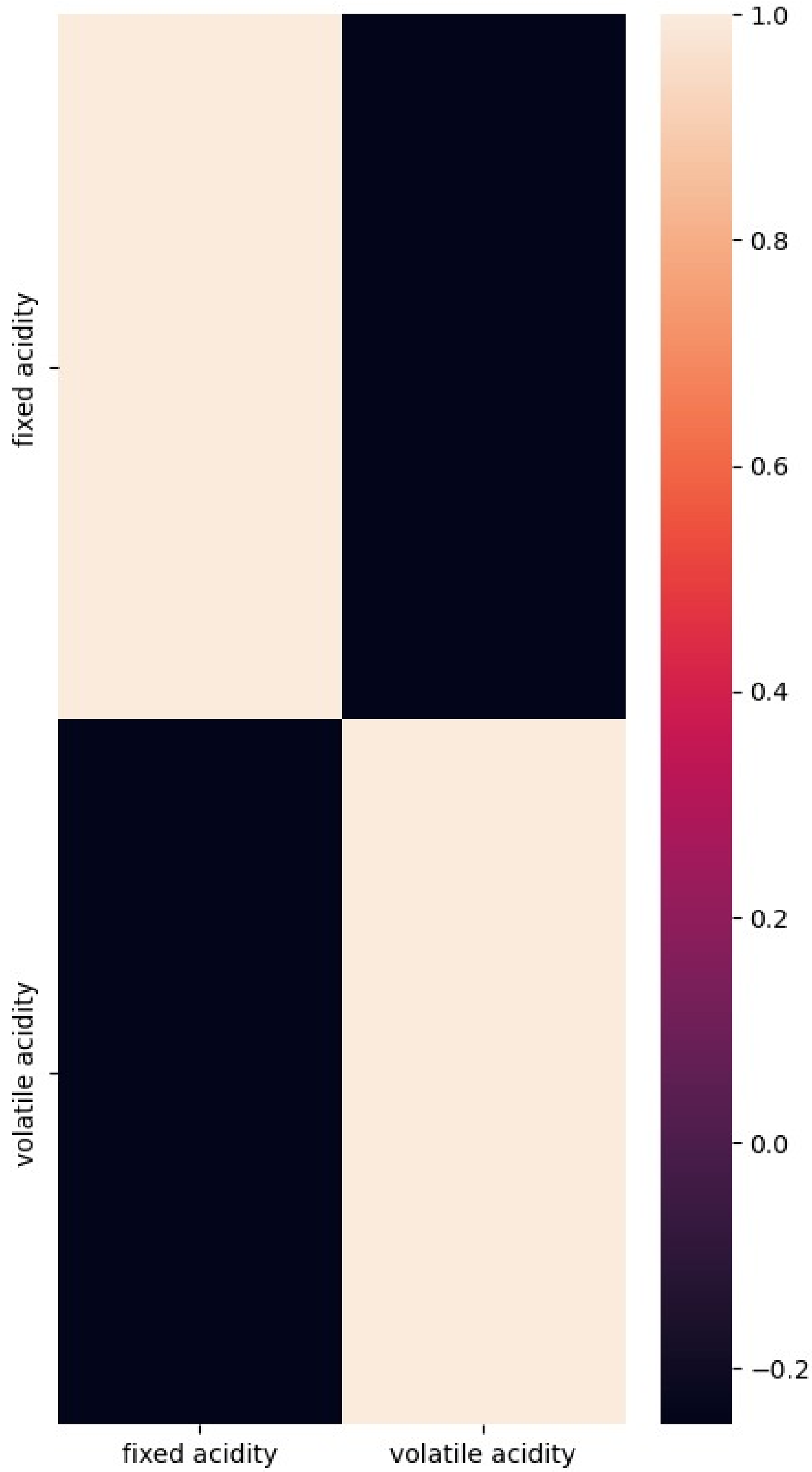


```
#HISTOGRAM
import matplotlib.pyplot as plt
df.hist(bins=20,figsize=(10, 10))
plt.show()
```



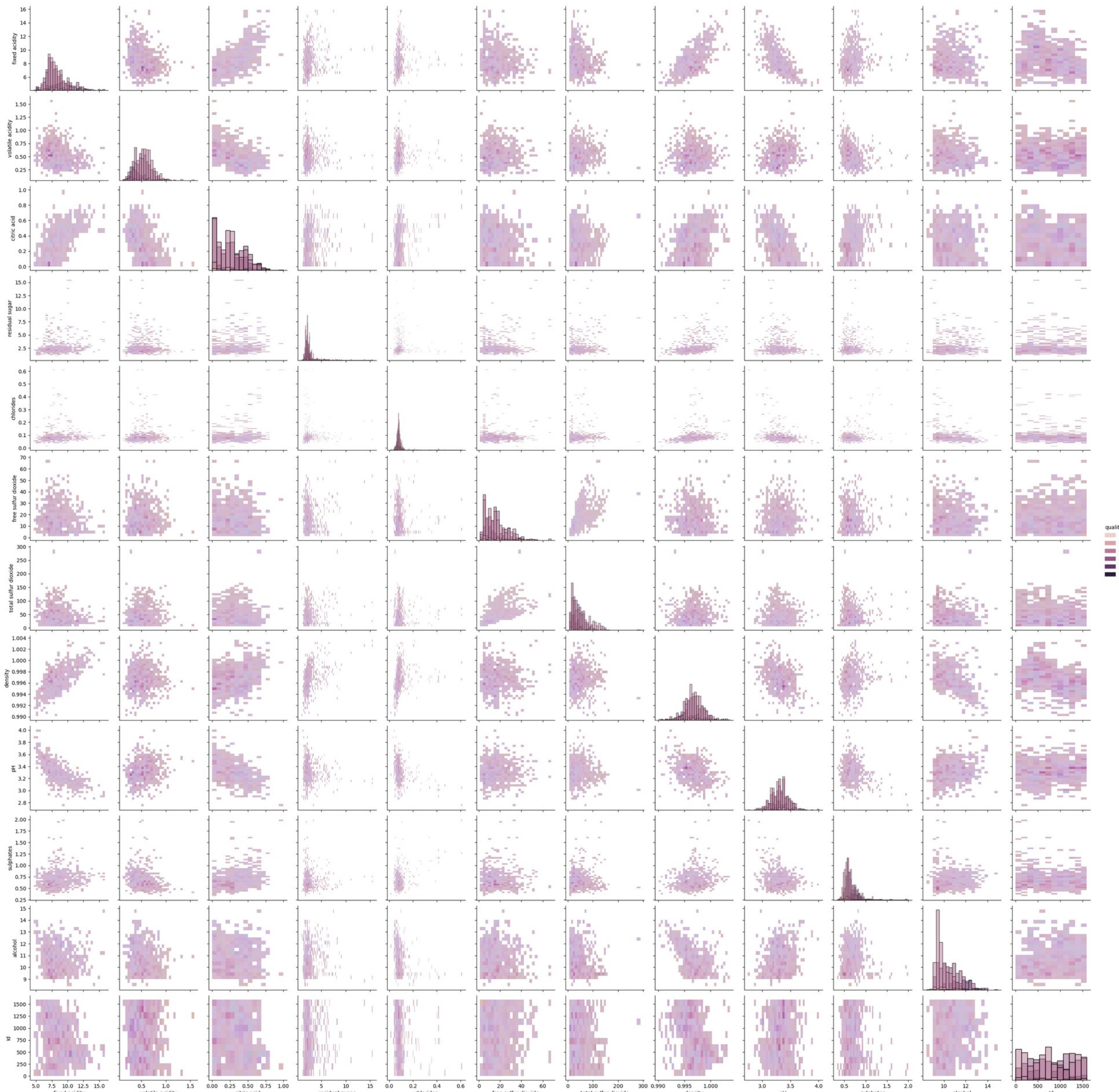

```
import matplotlib.pyplot as plt
import seaborn as sns
df[["fixed acidity","volatile acidity"]].corr()
plt.figure(figsize=(5,10))#if we write above then it show different
effect
sns.heatmap(df[["fixed acidity","volatile acidity"]].corr())
#plt.figure(figsize=(5,10))#if we write below the plt.figure then
condition changes and does not show any effect
```

```
<Axes: >
```

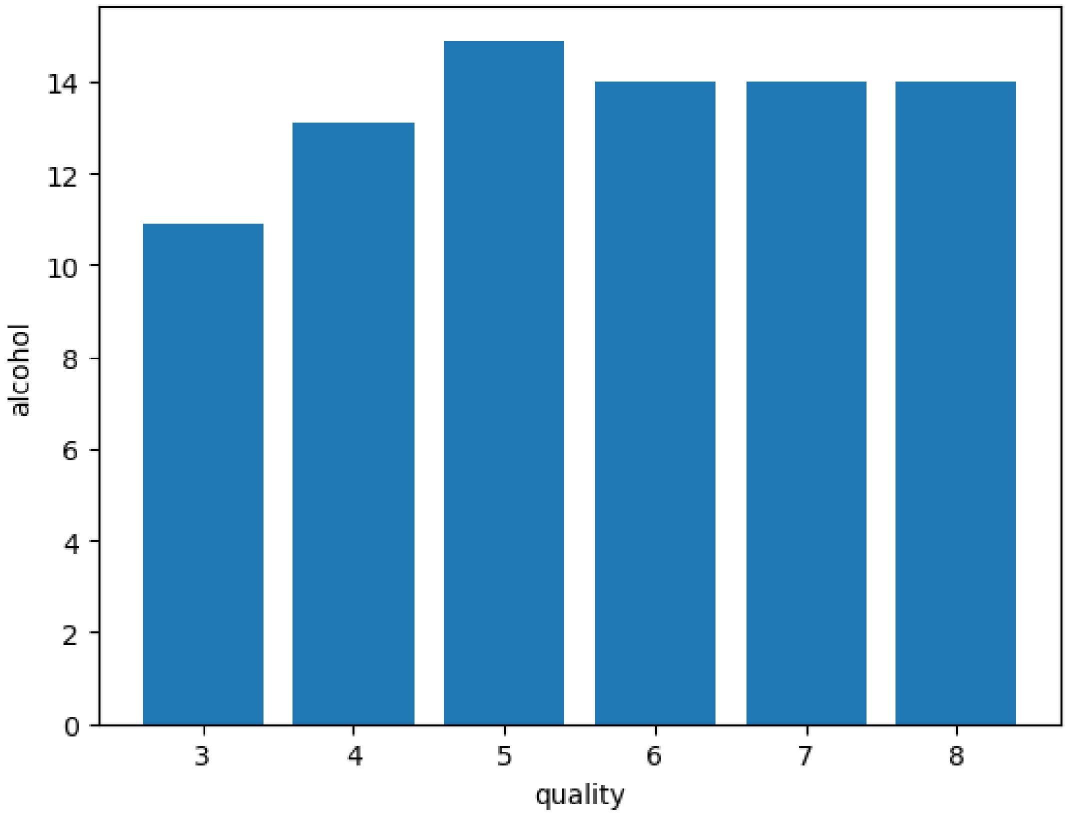



```
sns.pairplot(df,hue='quality',kind='hist')
```

```
<seaborn.axisgrid.PairGrid at 0x259916d9150>
```

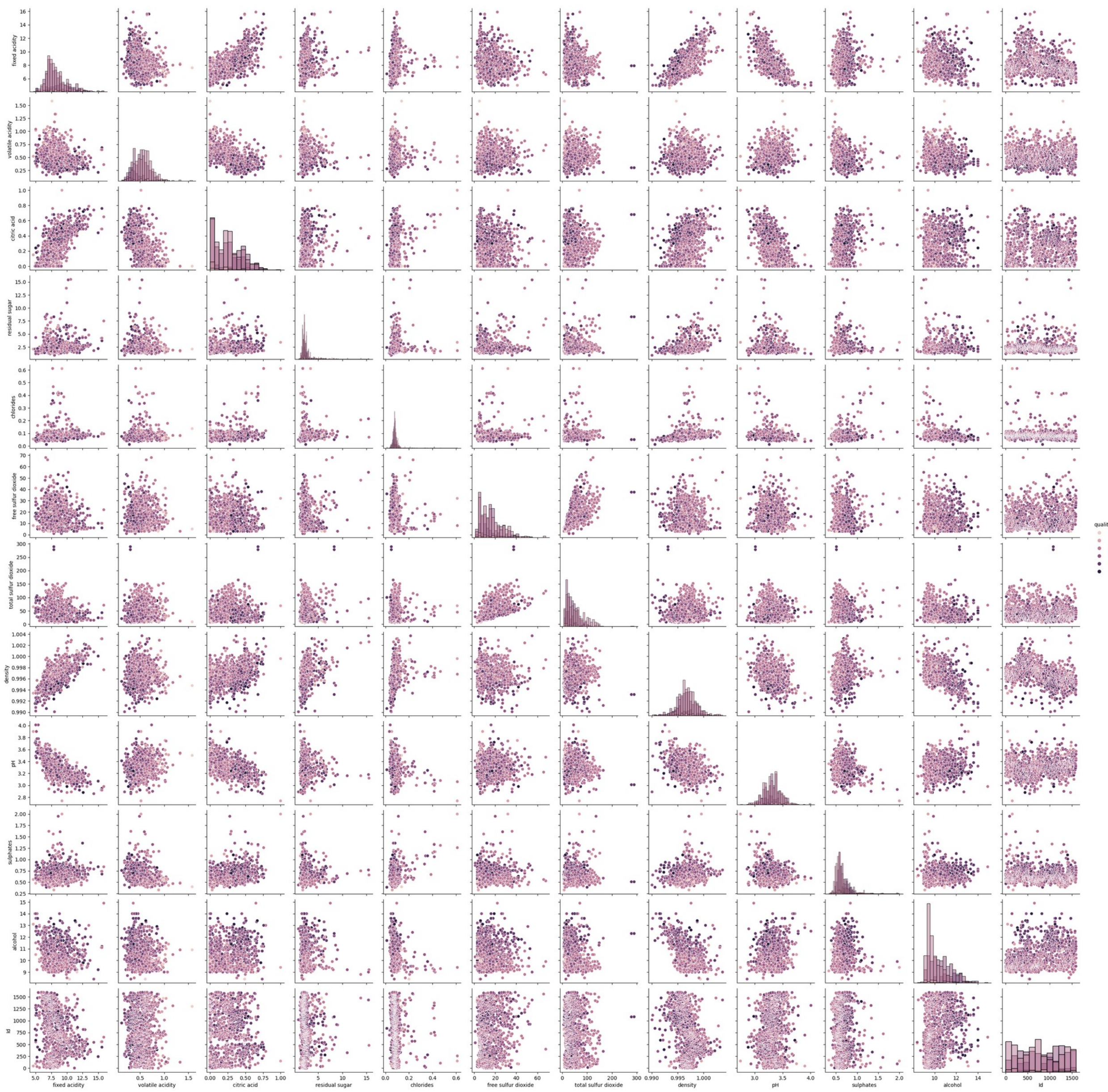



```
#BARGRAPH
plt.bar(df['quality'], df['alcohol'])
plt.xlabel('quality')
plt.ylabel('alcohol')
plt.show()
```



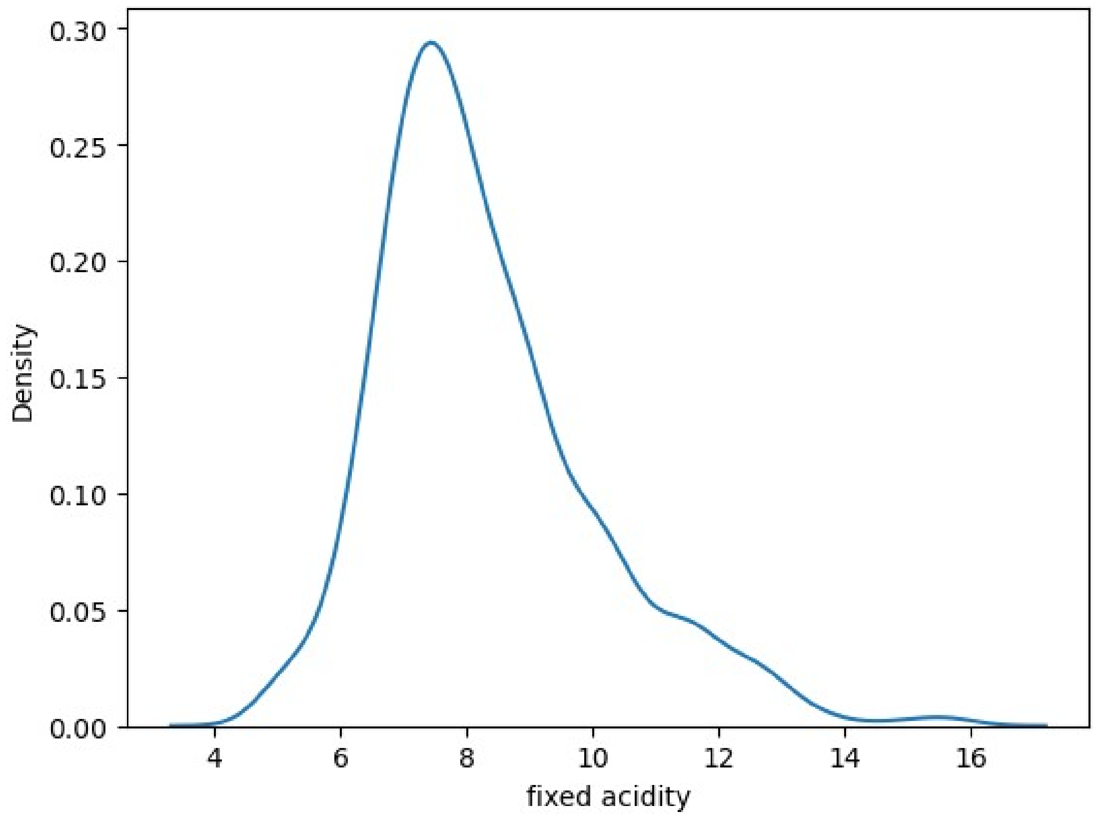

```
sns.pairplot(df,hue='quality',diag_kind='hist')
```

```
<seaborn.axisgrid.PairGrid at 0x26eab8b2950>
```

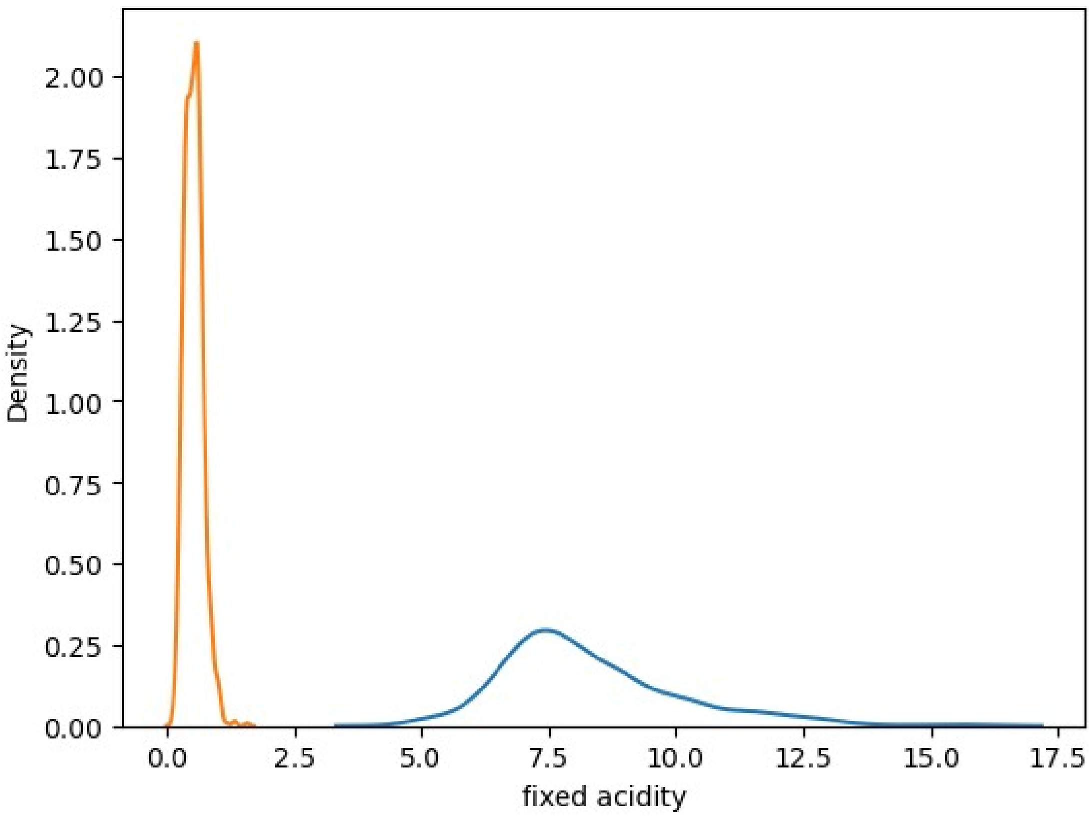


```
#KDE-K for kernal D for density E for estimation
sns.kdeplot(data=df,x=df['fixed acidity'])
```

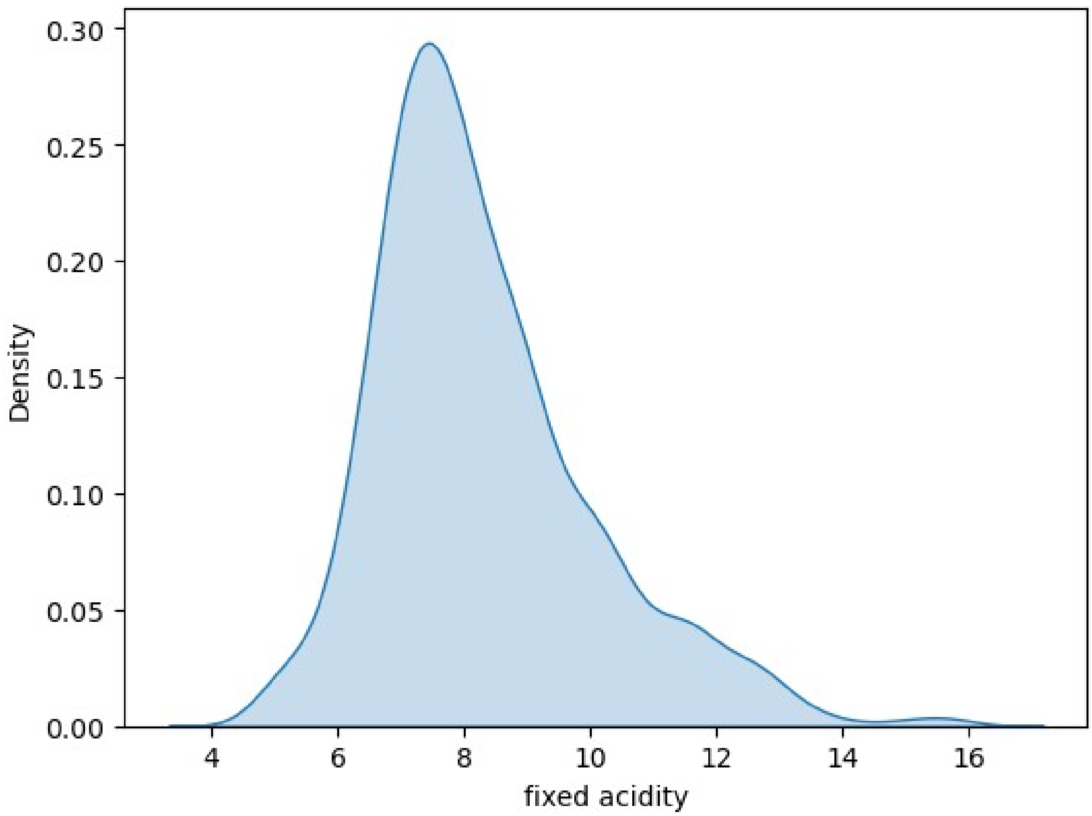
```
<Axes: xlabel='fixed acidity', ylabel='Density'>
```




```
sns.kdeplot(data=df,x=df['fixed acidity'])
sns.kdeplot(data=df,x=df['volatile acidity'])

<Axes: xlabel='fixed acidity', ylabel='Density'>
```



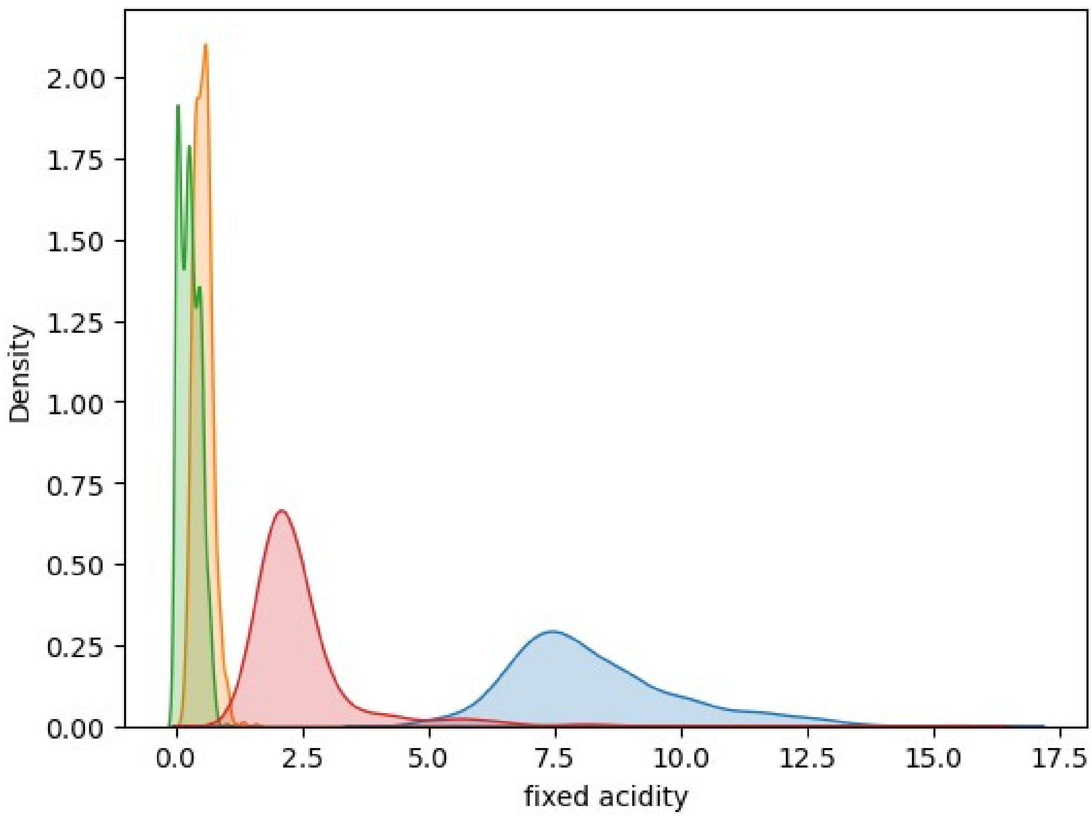

```
sns.kdeplot(data=df,x=df['fixed acidity'],shade=True)

C:\Users\Lenovo\AppData\Local\Temp\ipykernel_70156\1492466574.py:1:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df,x=df['fixed acidity'],shade=True)

<Axes: xlabel='fixed acidity', ylabel='Density'>
```



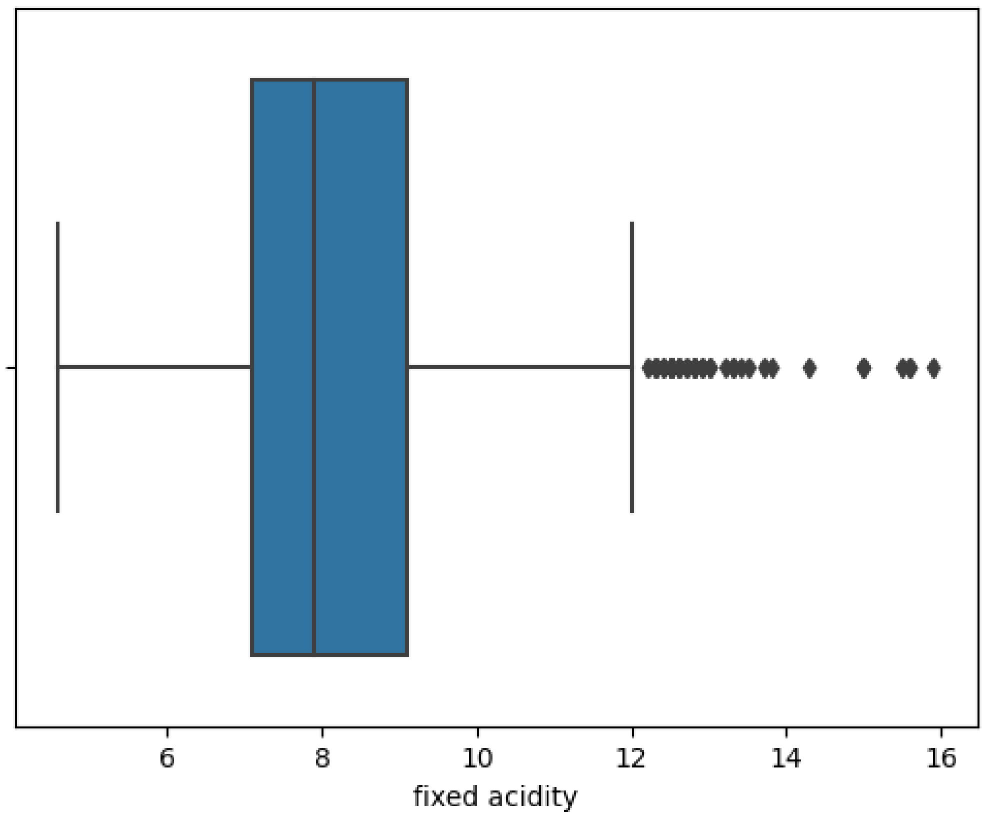

```
sns.kdeplot(data=df,x=df['fixed acidity'],shade=True)
sns.kdeplot(data=df,x=df['volatile acidity'],shade=True)
sns.kdeplot(data=df,x=df['citric acid'],shade=True)
sns.kdeplot(data=df,x=df['residual sugar'],shade=True)
```

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_59200\971205529.py:1:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df,x=df['fixed acidity'],shade=True)
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_59200\971205529.py:2:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df,x=df['volatile acidity'],shade=True)
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_59200\971205529.py:3:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.
```

```
  sns.kdeplot(data=df,x=df['citric acid'],shade=True)
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_59200\971205529.py:4:
FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df,x=df['residual sugar'],shade=True)

<Axes: xlabel='fixed acidity', ylabel='Density'>
```

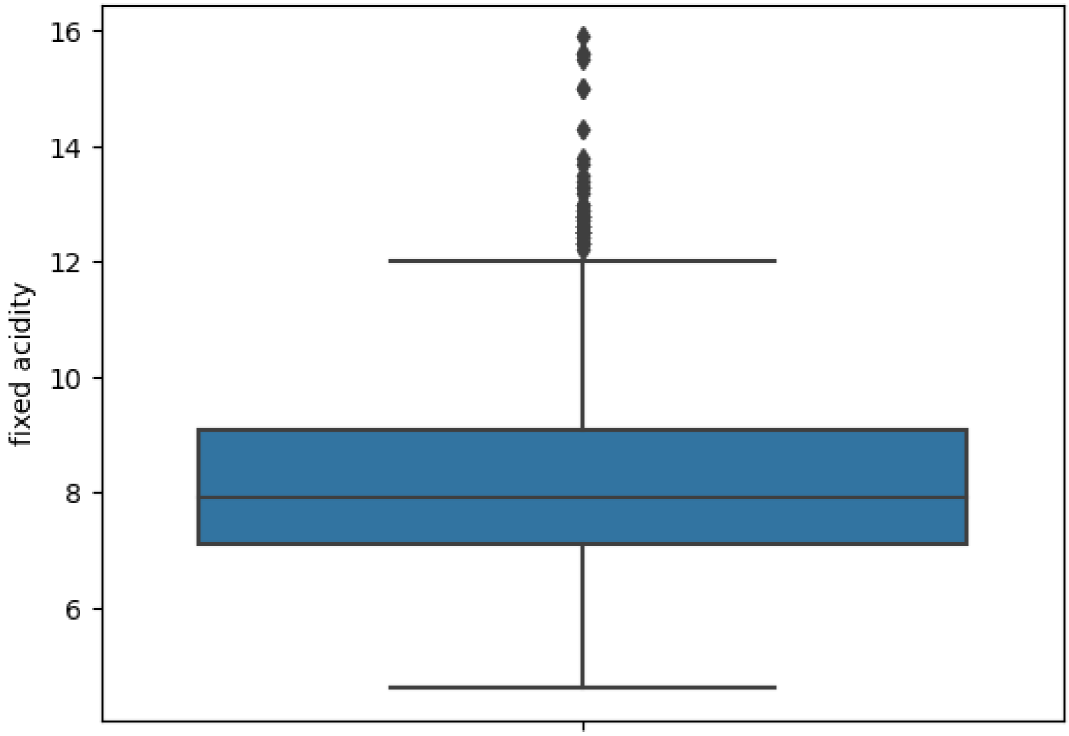



```
#BOXPLOT
sns.boxplot(x=df['fixed acidity'])

<Axes: xlabel='fixed acidity'>
```




```
sns.boxplot(y=df['fixed acidity'])
```

```
<Axes: ylabel='fixed acidity'>
```




```
df['fixed acidity'].quantile(0.25)
```

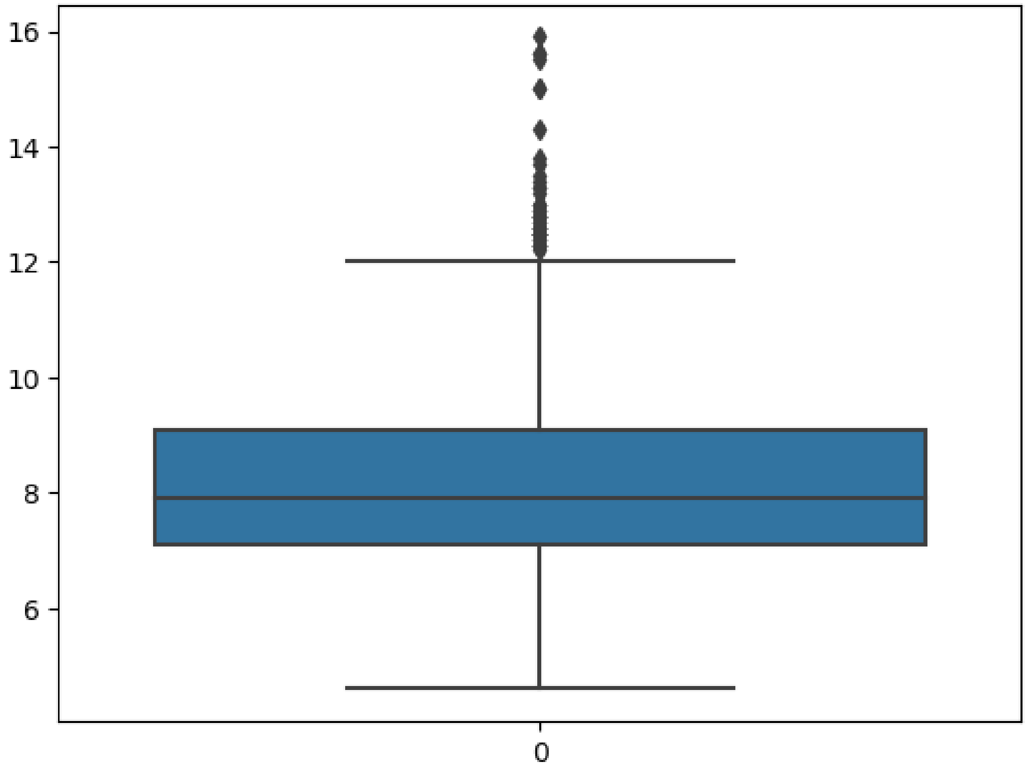
```
7.1
```

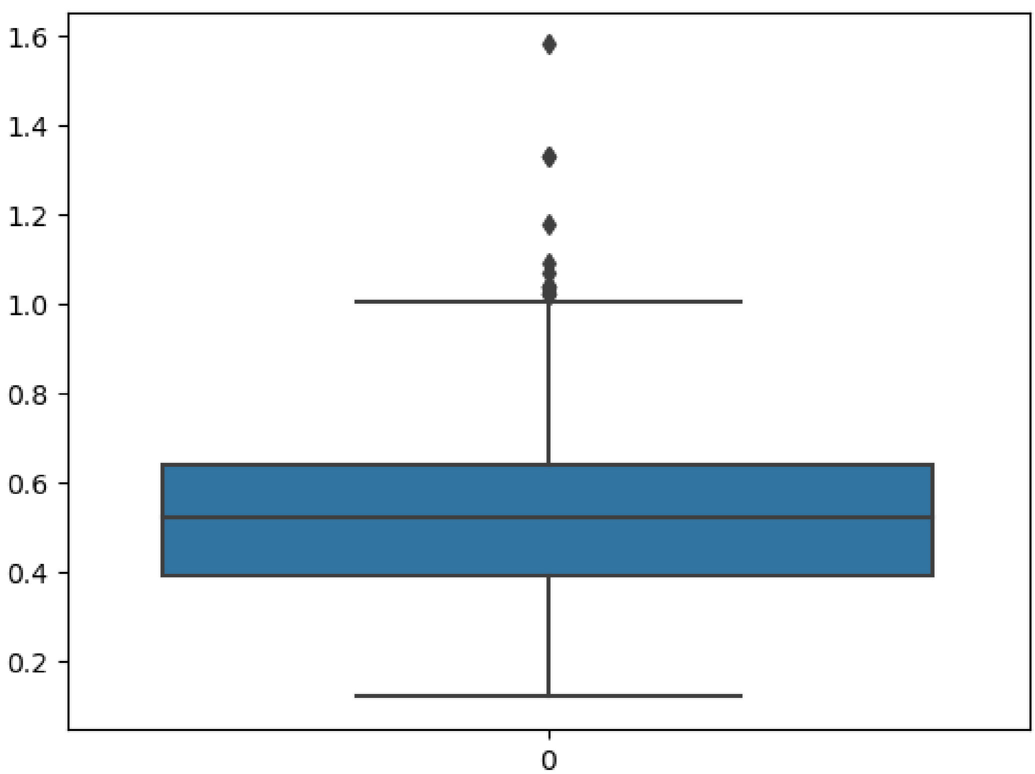
```
df['volatile acidity'].quantile(0.25)
```

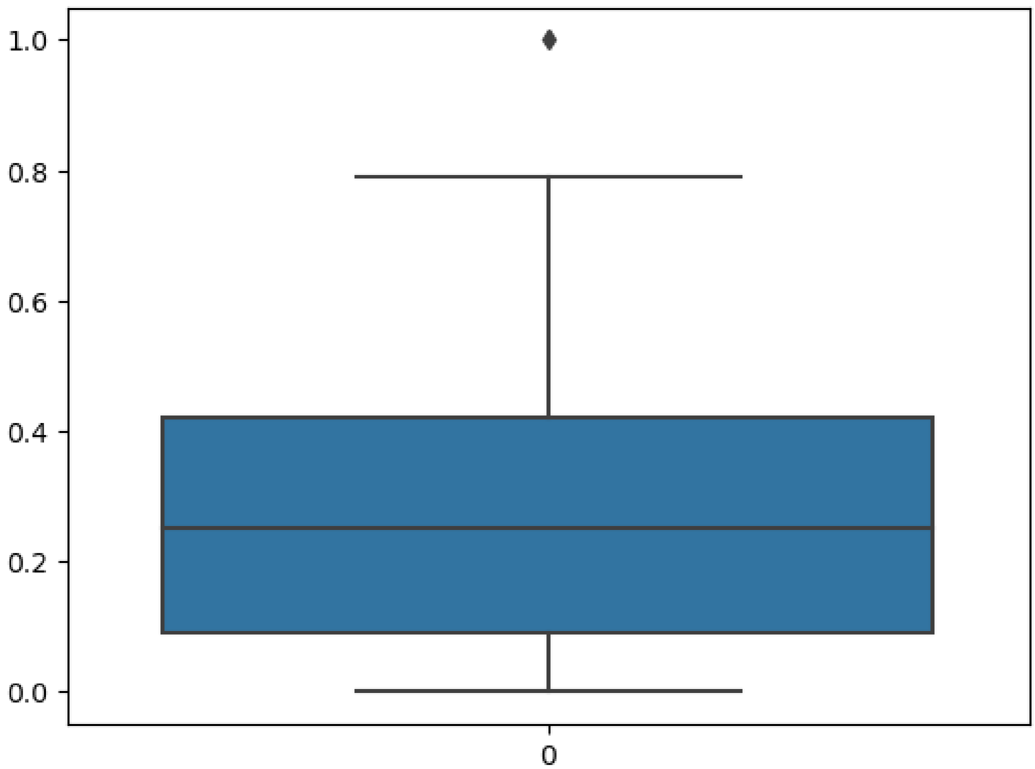
```
0.3925
```

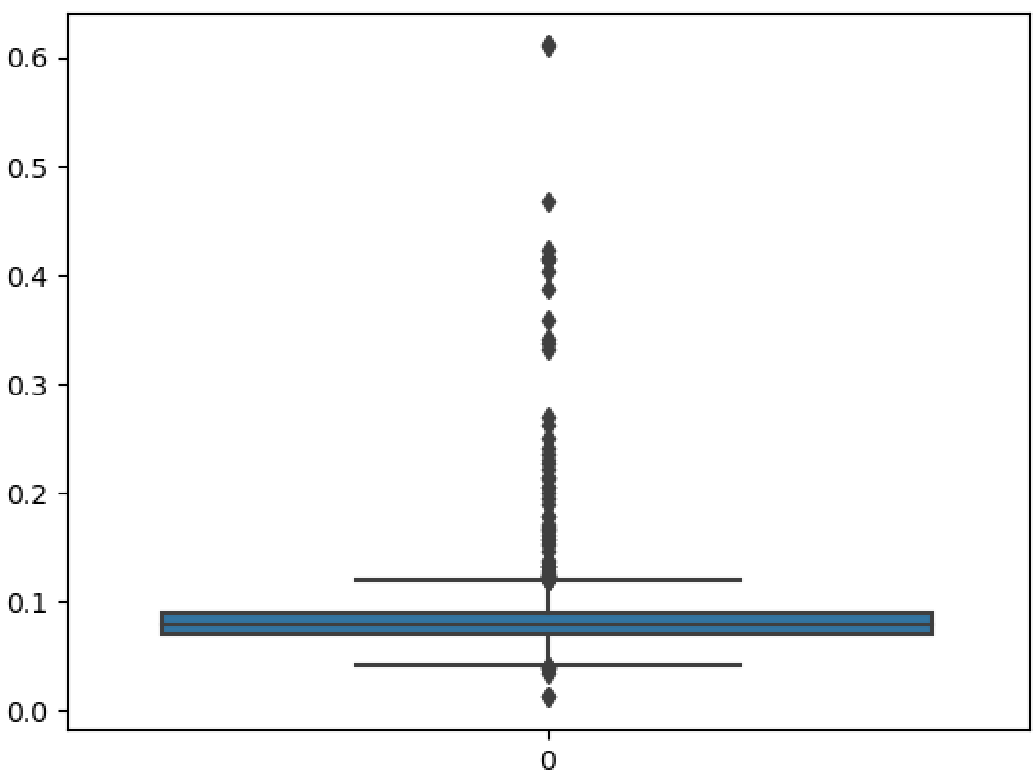
```
cal=['fixed acidity','volatile acidity','citric acid','chlorides']
for i in cal:
    print(sns.boxplot(df[i]))
    plt.figure()
```

```
Axes(0.125,0.11;0.775x0.77)
Axes(0.125,0.11;0.775x0.77)
Axes(0.125,0.11;0.775x0.77)
Axes(0.125,0.11;0.775x0.77)
```


16
14
12
10
8
6
0


1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0

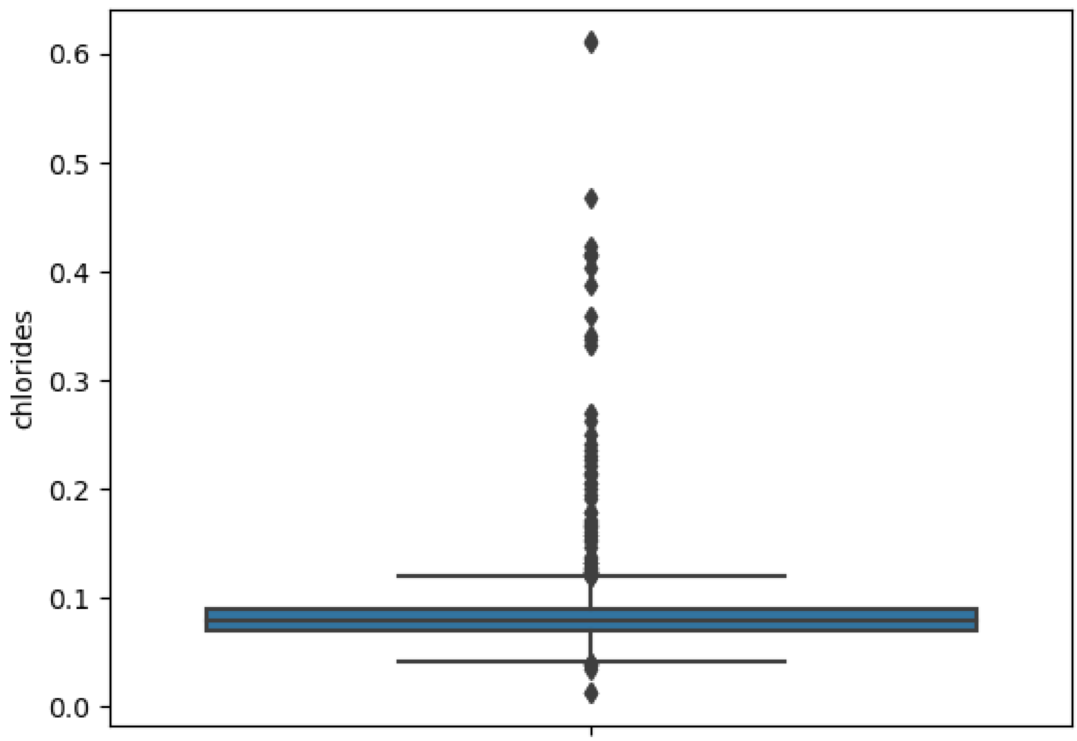





```
&lt;Figure size 640x480 with 0 Axes&gt;
```

```python
import seaborn as sns
import matplotlib.pyplot as plt
col=['chlorides']
for i in col:
    print(sns.boxplot(y=df[i]))
    plt.figure()
```

```
Axes(0.125,0.11;0.775x0.77)
```




```
<Figure size 640x480 with 0 Axes>
```

```python
#OUTLIERS ANALYSIS
q1=df['fixed acidity'].quantile(0.5)
print(q1)
q2=df['volatile acidity'].quantile(0.5)
print(q2)
q3=df['citric acid'].quantile(0.5)
print(q3)
IQR=q3-q1
upper=q3+(1.5*IQR)
print(upper)
lower=q1+(1.5*IQR)
print(lower)
```

```
7.9
0.52
```

```
0.25
-11.225000000000001
-3.575000000000001
```

```
df.columns
```

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

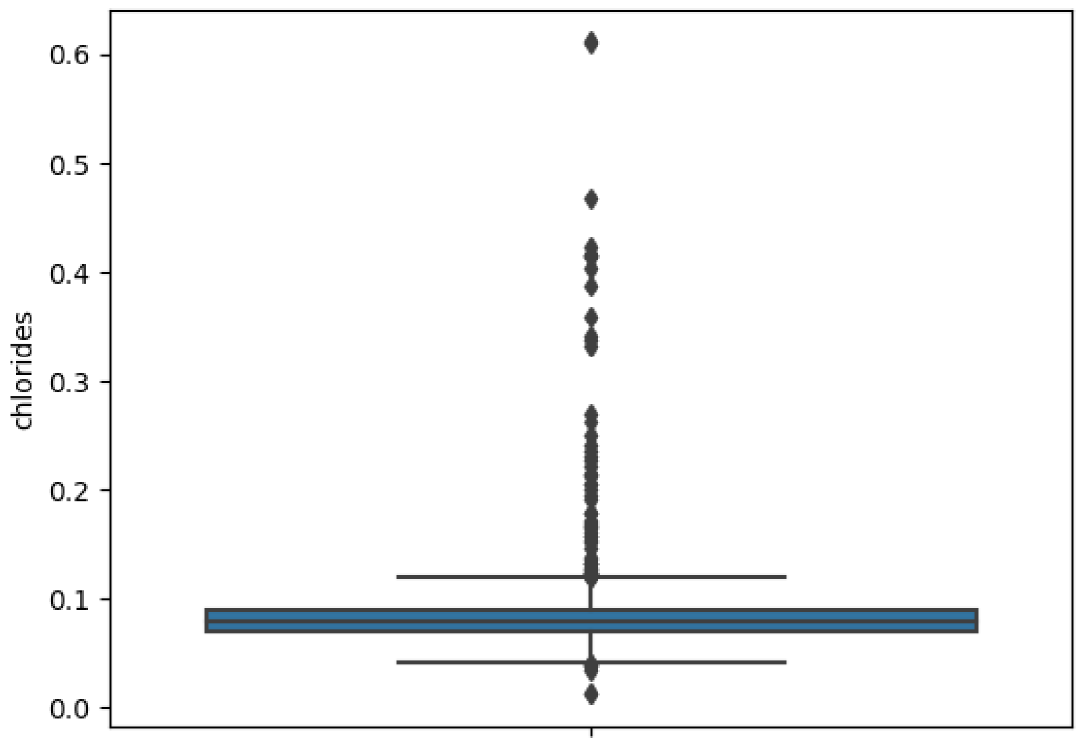
```
col=['fixed acidity', 'volatile acidity', 'citric acid', 'residual
sugar',
       'chlorides']
for i in col:
    print(df[i].value_counts())
```

```
7.2     43
7.1     41
7.0     40
7.8     40
7.5     37
        ..
4.6      1
13.7     1
13.4     1
13.5     1
12.2     1
Name: fixed acidity, Length: 91, dtype: int64
0.600    32
0.500    32
0.430    31
0.390    29
0.580    28
         ..
1.035     1
0.565     1
0.865     1
0.965     1
0.160     1
Name: volatile acidity, Length: 135, dtype: int64
0.00    99
0.49    47
0.24    42
0.02    35
0.01    26
        ..
0.61     1
```

```
0.72     1
1.00     1
0.75     1
0.62     1
Name: citric acid, Length: 77, dtype: int64
2.00    107
2.10    103
1.80     92
2.20     88
1.90     80
       ...
7.30      1
7.20      1
2.95      1
3.65      1
4.40      1
Name: residual sugar, Length: 80, dtype: int64
0.080    48
0.077    41
0.074    38
0.084    38
0.078    36
         ..
0.222     1
0.422     1
0.034     1
0.387     1
0.230     1
Name: chlorides, Length: 131, dtype: int64
```

```python
import seaborn as sns
import matplotlib.pyplot as plt
col=['chlorides']
for i in col:
    print(sns.boxplot(y=df[i]))
    plt.figure()
```

```
Axes(0.125,0.11;0.775x0.77)
```




```
<Figure size 640x480 with 0 Axes>

df.chlorides.value_counts()

0.080    48
0.077    41
0.074    38
0.084    38
0.078    36
         ..
0.222     1
0.422     1
0.034     1
0.387     1
0.230     1
Name: chlorides, Length: 131, dtype: int64

#ENCODING
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()

df

      fixed acidity  volatile acidity  citric acid  residual sugar
chlorides  \
0               7.4             0.700         0.00             1.9
0.076
```

```
1               7.8              0.880         0.00               2.6
0.098
2               7.8              0.760         0.04               2.3
0.092
3              11.2              0.280         0.56               1.9
0.075
4               7.4              0.700         0.00               1.9
0.076
...             ...                ...          ...               ...
...
1138            6.3              0.510         0.13               2.3
0.076
1139            6.8              0.620         0.08               1.9
0.068
1140            6.2              0.600         0.08               2.0
0.090
1141            5.9              0.550         0.10               2.2
0.062
1142            5.9              0.645         0.12               2.0
0.075

      free sulfur dioxide  total sulfur dioxide  density    pH
sulphates  \
0                    11.0                  34.0  0.99780  3.51
0.56
1                    25.0                  67.0  0.99680  3.20
0.68
2                    15.0                  54.0  0.99700  3.26
0.65
3                    17.0                  60.0  0.99800  3.16
0.58
4                    11.0                  34.0  0.99780  3.51
0.56
...                   ...                   ...      ...   ...
...
1138                 29.0                  40.0  0.99574  3.42
0.75
1139                 28.0                  38.0  0.99651  3.42
0.82
1140                 32.0                  44.0  0.99490  3.45
0.58
1141                 39.0                  51.0  0.99512  3.52
0.76
1142                 32.0                  44.0  0.99547  3.57
0.71

      alcohol  quality    Id
0         9.4        5     0
1         9.8        5     1
```

```
2          9.8        5     2
3          9.8        6     3
4          9.4        5     4
...        ...      ...   ...
1138      11.0        6  1592
1139       9.5        6  1593
1140      10.5        5  1594
1141      11.2        6  1595
1142      10.2        5  1597

[1143 rows x 13 columns]
```

```
#LINEAR REGRESSION
X=df.iloc[:,:-1].values
Y=df.iloc[:,-1].values

X.shape
```

```
(1143, 12)
```

```
X
```

```
array([[ 7.4  ,  0.7  ,  0.   , ...,  0.56 ,  9.4  ,  5.   ],
       [ 7.8  ,  0.88 ,  0.   , ...,  0.68 ,  9.8  ,  5.   ],
       [ 7.8  ,  0.76 ,  0.04 , ...,  0.65 ,  9.8  ,  5.   ],
       ...,
       [ 6.2  ,  0.6  ,  0.08 , ...,  0.58 , 10.5  ,  5.   ],
       [ 5.9  ,  0.55 ,  0.1  , ...,  0.76 , 11.2  ,  6.   ],
       [ 5.9  ,  0.645,  0.12 , ...,  0.71 , 10.2  ,  5.   ]])
```

```
Y
```

```
array([   0,    1,    2, ..., 1594, 1595, 1597], dtype=int64)
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train,Y_test=
train_test_split(X,Y,test_size=.25,random_state=67)

X_train
```

```
array([[ 7.  ,  0.62,  0.08, ...,  0.53,  9.  ,  5.  ],
       [11.6 ,  0.47,  0.44, ...,  0.86,  9.9 ,  4.  ],
       [ 8.3 ,  0.78,  0.1 , ...,  0.53, 10.  ,  5.  ],
       ...,
       [ 7.3 ,  0.65,  0.  , ...,  0.47, 10.  ,  7.  ],
       [ 5.1 ,  0.51,  0.18, ...,  0.87, 12.9 ,  7.  ],
       [ 9.8 ,  0.39,  0.43, ...,  0.46, 11.4 ,  5.  ]])
```

```
Y_train
```

```
array([  89,  833,  678, 1011, 1246,  106, 1340,  289,  988,  110,
1013,
        897,  720,  530,  448, 1552, 1420,  313,  231, 1168, 1522,
588,
        183,  671, 1335, 1503, 1057,  197,  466,  497,  637, 1577,
200,
        808,  103,  974,  772, 1274,  318,  131,  644, 1184,  270,
1597,
        533,   91, 1551,  950,  756,  191, 1404, 1052, 1300,  647,
1436,
        960,  325, 1240,  286,  749, 1034,  753,  819, 1279,  757,
1347,
        477,  604, 1399,   42, 1051, 1575, 1283,  577, 1062,  699,
1288,
       1371,  455, 1361,  765, 1366,  360,  427, 1485, 1029,  308,
1088,
        250,  583,  733,  232,  592, 1085,  384,  998, 1594, 1364,
1285,
        178,  453, 1141, 1390, 1047,  524,  830, 1284, 1081,  593,
685,
       1510, 1292,  403, 1572,  406, 1391, 1382,  245,  886,  795,
991,
       1532, 1290,  759,   45,  728, 1114,  348,  838, 1357,  741,
877,
        409,  240,  809, 1275,  944, 1362,  501,  895, 1556,  328,
156,
        537, 1107, 1461,  152, 1449, 1531, 1570,  794,  750,   35,
1415,
        324,  339,  721,  892,  188,  559,  277,  390,  987,  736,
967,
       1439, 1110,  901,  827, 1555, 1511,  884, 1225,  173, 1053,
1027,
       1351,  475,  636,  234, 1155,   54, 1429,  280, 1238, 1450,
292,
         34, 1472, 1195,  199,  893,  999,  633, 1393, 1454,  531,
114,
       1112, 1169, 1583, 1119,  952,  257,  803, 1467,   60, 1419,
729,
        806, 1101, 1299,  655,  983, 1479,  860,  485,  989,  442,
160,
         96, 1509, 1495, 1245,  595,  153,  168,  796,  281,  428,
1254,
        435,  640, 1170,  702,  842,  745,  172, 1341, 1386, 1586,
392,
        388,  351, 1207,  312,  407, 1372,  560, 1540,  296,  480,
1183,
        193,  894,  915, 1149, 1325, 1327, 1162,  440,  888,  791,
1058,
        639,  841,  523,  564, 1191, 1571,  591,  754, 1451, 1567,
1049,
```

```
       1559,  823, 1595,  718, 1421,  539,  155, 1306,  180, 1266,
536,
        491,  872,  396,  768, 1336, 1493, 1546,  771, 1010,   22,
353,
        811, 1487,  652,  216, 1592,  452,  275, 1091,  376, 1298,
579,
        113, 1220,  780,  697, 1226, 1319, 1392,  569, 1416, 1076,
1289,
        278,  137,  661,  261,  495, 1059,  244, 1188,  126,  881,
662,
        946,   78,  707,  283,  402,  337, 1412,  982,  928, 1563,
111,
        632, 1297,  429, 1480,  487,  175,  565, 1311, 1558,  684,
499,
        515, 1203,  752,  760, 1580,  290, 1281, 1529,  964,  343,
865,
        578,  431,  975,  891,  971,  457, 1099,  912,   52, 1236,
1560,
       1271,  848,  128, 1561,  167,   80, 1206,  953, 1127, 1519,
590,
         10,  648,  454, 1108, 1518,  692,   93,  513,  300, 1105,
327,
       1210, 1396, 1447, 1428,  164,   76, 1476, 1046,    6, 1578,
1176,
        450, 1533, 1228,  557,  159, 1333, 1014,  185,  689, 1417,
311,
        298,  941,   98, 1506, 1006, 1569,  359,  708, 1118, 1478,
920,
       1078, 1343,  608,  425,  395, 1332,  414,  529,  793, 1151,
831,
       1227,   23,  898,  446,  694,  785,  596,  801,  955,  412,
58,
        285,  755,  653,  853, 1515, 1355,  770,   64, 1120,  306,
1264,
        459,  961,  502,  567, 1593, 1070,  657,  856, 1256, 1230,
942,
       1197,  962,   16, 1475,  810, 1219,  268, 1100,  676, 1019,
176,
        925,  609,  469,  422,  500,  824,  656,  127,    3, 1431,
1305,
       1378,  927,  122,  182, 1443, 1339, 1268,  673, 1060,   88,
333,
        813,    4,  336,  447, 1069, 1562,  627, 1000,  218,   90,
1026,
       1445,  906, 1468,  737,  658,  902, 1250,  705,  748, 1409,
120,
        703,  710, 1063,  775,  226,   77,  956,  587,  727, 1385,
1239,
        433,  416, 1548,  344, 1134, 1273,  951,  354, 1338, 1021,
542,
```

```
       1395,  511,  535,  730, 1334,  919, 1022,  726,  504,  778,
1189,
       1103, 1018,   67,    2, 1089,  792, 1156, 1380, 1201,  924,
731,
        698,   63,  548,  423,  157,  400,  166,  288, 1212, 1545,
5,
         61,  580, 1258,  184,  821, 1358,  626, 1418,  820, 1221,
620,
       1437, 1486,  444,  706,  617,  165, 1422,  367,  464, 1291,
213,
        314,  804, 1178, 1041,  570,  258,  532,  779, 1145,  739,
439,
       1304,  784, 1411,  102, 1452, 1128, 1346, 1331,  252,  307,
279,
       1237,  287,  985,  527, 1326,  506, 1324, 1344,  667,  851,
1144,
        404,  211, 1143,  124, 1568, 1231, 1321,  562,  917,  862,
419,
       1280,  889,  230,  259,  758,  543, 1111, 1125,  725,  115,
788,
        293, 1353, 1481,  505, 1133,  251,   51, 1153, 1179,  434,
79,
        494,  871,   59,  334,  836, 1261,  352,  295, 1582, 1199,
1492,
       1094, 1092,  610,  204, 1132, 1413,  145, 1023,  885,  269,
1494,
         24, 1278,  461, 1525, 1140,  936, 1173,  410,  650,  170,
781,
       1356,  253,    1,  616, 1423, 1270,  512, 1087,  467,   40,
1477,
       1096, 1471,  612, 1367,  460, 1036,  194, 1376,  361, 1061,
104,
        489,  516,   36, 1460,  140,  802,  854,  479,  934,  682,
214,
        134,  556,  534,  540, 1008,  472, 1229,  355, 1243,  634,
190,
       1267, 1038, 1505, 1142,  931, 1427,  545, 1175,   72,  625,
1501,
       1260,  121, 1172,  364,  858, 1115,  691,  711, 1066, 1122,
837,
        679,  496,   43, 1433,  264,  177, 1379, 1407, 1517,  366,
161,
       1316,  993,  347, 1535,  654,  922,  629, 1309,  437,  949,
777,
        468, 1039, 1590,  947,   32,  572,    8,  926,  144,  135,
399,
        222, 1232,  263, 1024, 1139,  932, 1497,  362,  430,  100,
1565,
        554, 1146,   86,  649, 1370, 1296, 1464,  622, 1368,  267,
1265,
```

```
        787, 1410,  981,  208,  418,  977,  220, 1591,   30,  681,
1402,
       1073, 1135,  769,  538,  260,  790, 1473,  426,  761,  882,
683,
        507, 1147,  666, 1469, 1214, 1277, 1388,    7, 1157, 1181],
      dtype=int64)
```

X_test

```
array([[12.   ,  0.38 ,  0.56 , ...,  0.71 , 10.9  ,  6.   ],
       [11.3  ,  0.34 ,  0.45 , ...,  0.66 ,  9.2  ,  6.   ],
       [ 7.5  ,  0.63 ,  0.27 , ...,  0.58 ,  9.8  ,  6.   ],
       ...,
       [ 8.   ,  0.3  ,  0.63 , ...,  0.78 , 10.8  ,  6.   ],
       [ 9.1  ,  0.37 ,  0.32 , ...,  0.8  , 11.2  ,  6.   ],
       [ 8.   ,  0.705,  0.05 , ...,  0.95 , 10.5  ,  6.   ]])
```

Y_test

```
array([ 241,  669, 1315,  907, 1499, 1337,  262,  150, 1192,  321,
717,
       1566,   56, 1584,  284, 1384, 1312,  493, 1330,  417,   65,
846,
       1012, 1438,   25, 1048, 1345,  326, 1020,  130, 1557,  995,
1198,
        990,  958, 1272,  547, 1459, 1056,  693, 1148,  517,   99,
1190,
       1194,   46,  221,  520, 1458, 1234,  714,   82, 1035, 1408,
1490,
        378,  762,  911,  476, 1397,  598,  369,  319, 1152, 1253,
158,
       1414, 1093,  916, 1400,   37,  297, 1204,  151, 1007, 1457,
552,
       1045,  910,  553,  875, 1504,   28,   84, 1573,  576,  483,
1257,
         53,  148,  619,   29, 1174, 1090,  799,  704,  107,  215,
642,
        843,  116,   73, 1301,   21, 1104,  602,  630,  470,   19,
421,
        563,  939,  611, 1549,  734,  929, 1328, 1262, 1079,  896,
568,
       1044,  380,  857,  900, 1216, 1209,  773,  621,  732,  544,
235,
        863,  301,   12,  550,  302,   26,  358, 1553,  397, 1067,
1130,
        142,  146,   85,  904, 1587,  276,  179, 1523, 1514,  304,
1117,
        205, 1534, 1463, 1544,  246, 1086, 1071,  143, 1466, 1109,
551,
          0, 1224,  363,  514,  471,  980, 1202,  764,  217,  449,
```

```
1252,
        356, 1200,  665,  456,  391,  635, 1430,  210,  413,  908,
1083,
        607,  847, 1138,  850,  255,  415,  436,  335,  225,  744,
94,
         87, 1507,  372,  481,  814, 1032,  695, 1065,  723,  256,
1004,
        294, 1538, 1483, 1055, 1474,  688,  498, 1164,  581, 1434,
1508,
       1528,  105,  660,  628,  631, 1276,  746, 1406,  163, 1005,
826,
        223, 1403, 1082, 1116,  342,  701, 1208,  541,  546,  242,
1016,
        789, 1282,  700,   50,  243,  377,  196,  181, 1167, 1488,
492,
       1455,  266, 1136,  509,  677,   41, 1383,   13, 1530,  968,
1526,
        978,  206, 1359, 1251,  162,  371, 1322,  797, 1329,  719,
249,
       1064,  918,  766,  186, 1025, 1470,  740,  357, 1576, 1302,
69],
      dtype=int64)
```

```
X_train.shape
```

```
(857, 12)
```

```
#FEATURE SCALING
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

```
from sklearn.linear_model import LinearRegression
reg=LinearRegression()
```

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

```
LinearRegression()
```

```
reg.coef_
```

```
array([-8.49521000e+01,  1.15202170e+02,  3.14545000e+02,
4.15854633e+00,
       -3.02028117e+02,  1.24269361e+01, -4.61333273e+00, -
3.73576471e+04,
       -4.35625267e+02, -1.25777042e+02,  5.60787306e+01, -
4.46523564e+01])
```

```
reg.intercept_
```

```
39834.55848194451
```

```
Y_pred=reg.predict(X_test)
```

```
Y_pred
```

```
array([ 536.0913878 ,  615.6672747 ,  703.9902699 ,  980.98370256,
        970.50239001, 1075.26341935,  784.98055005,  988.7836496 ,
        907.72029104,  575.81553786,  815.91026766, 1181.64715411,
        854.53788546, 1212.83022811,  568.80533489,  853.0966624 ,
        993.24603163,  579.54402124,  695.74660228,  788.70322128,
        952.24922106,  882.57117678,  970.75508097,  901.1365322 ,
        990.17484252,  743.91060984,  858.33589974,  741.52577317,
        811.73077381,  715.54870895,  963.28468054,  799.11951446,
        862.82430917, 1011.37564234, 1054.43534766, 1164.23808401,
        586.48754514,  957.82087026,  968.20164831,  392.09942001,
        988.51973688,  658.16206774,  721.64917511,  969.01162764,
        811.8481123 ,  685.07496782,  856.19162417,  603.20123337,
        878.16171381, 1214.26810206,  610.22876275,  823.97191208,
        674.0860328 , 1138.6003416 , 1190.62826881,  794.552941  ,
        703.95723816,  799.12462201,  707.02810413,  715.20870572,
        738.30317187,  821.53555098,  654.21402256,  860.49348393,
        776.59104376,  822.03427362,  649.21174908,  768.67731172,
        893.30212261,  689.46170884,  784.65288034,  592.77978253,
       1023.41221484,  897.35247361,  866.70500588,  753.31491109,
        668.14479652, 1210.57294573,  995.66961157, 1154.77004106,
        834.84121132, 1033.23029238,  779.47710416,  922.37852343,
       1028.36802245,  701.38044166,  672.36250569,  813.96309036,
        666.68441988,  871.04944699,  568.67125587,  767.26644282,
        762.38449685, 1122.77565925,  712.74253586,  711.71811884,
        799.38830711,  774.99186897,  624.80708439,  667.179074  ,
        680.46512042,  771.42947002,  869.57613924,  689.57412313,
       1184.8226665 ,  803.99400788,  650.20941656,  546.88879154,
        764.15637997, 1143.08318542,  924.30061105, 1067.83692107,
        462.78618539, 1069.43216197,  620.66751341,  956.42372733,
        936.8960328 ,  903.21628399,  514.50902385,  986.85929374,
        589.48457886, 1124.70184197,  787.53863915,  781.11788161,
        842.21625717,  578.74263595, 1097.38960148,  525.01716037,
        488.63808508,  886.17877575,  348.77617439,  732.30755234,
        586.01424853,  625.28554849,  898.36019159,  844.69626463,
        634.08338232,  884.66921266,  512.77338418,  851.65309837,
        600.28334569,  757.13833323,  716.05289231, 1240.12796492,
        816.54973196,  968.19557768,  921.91721578, 1078.12604684,
        503.05742918,  725.33834271, 1102.58453502,  627.8183631 ,
        669.87450392, 1033.976805  ,  446.28631342, 1044.19235028,
        942.20368855,  994.01288611,  812.25251295,  798.89726582,
        809.90949864,  891.73980663,  969.34989675,  665.418789  ,
        639.75046508,  680.28272012,  498.02992458,  640.02606935,
        848.9346132 ,  912.13940037,  734.83332421,  857.11237951,
        529.62702469,  753.92990474,  693.67722716,  826.80385915,
        664.76607531,  987.72594584,  815.93033158,  718.63487863,
        380.97661933,  965.66963823,  943.69101355,  910.44423173,
        702.37269494, 1022.24778959, 1042.09803471,  862.31187277,
        690.51647875,  771.07484227,  787.91377269,  516.68725976,
```

```
        329.60562447, 1115.65339593,  593.23909402,  933.19038266,
        404.37768643, 1239.70761363,  803.85864553, 1033.23029238,
        857.9386383 ,  818.13948575,  545.7183075 ,  708.44400492,
       1088.02566911,  850.51762486,  814.18622427,  709.25278128,
       1050.13312001,  283.44519702, 1092.48985001,  989.78016508,
        520.92148345,  773.34791134,  820.05399399,  663.21905496,
        824.27702043,  486.18100628,  650.42129786, 1008.92327205,
        841.39225549,  677.56325802,  723.6714779 ,  650.20941656,
        608.81060753, 1086.14382338,  633.70004882,  834.83311833,
        738.97484773, 1182.7529067 ,  910.81364462,  710.42361199,
        780.60080462,  869.59916278, 1033.976805  ,  542.52453136,
        709.8451694 , 1023.41221484,  771.17026914,  685.80610331,
        783.63803724,  966.20973092,  528.0109678 ,  836.73141583,
        680.24692555,  751.17368323,  123.73586086,  821.53555098,
        825.05683989,  565.78890032, 1070.15905946, 1009.63551781,
       1025.63495585,  991.30097398,  618.25309137,  751.11081148,
        659.28815674,  732.25903828,  826.37624439,  723.75755608,
        693.88860853, 1011.00845157, 1191.00601906,  934.85482478,
       1052.39496642,  446.28631342,  581.92820149,  956.71787463,
        846.57735247,  757.90852674, 1082.59961257,  887.11751279,
        695.74660228,  767.64823784,  541.61629465, 1008.63797428,
        878.6506882 ,  612.80855049,  765.7090696 ,  693.47989026,
        875.26212753,  940.01181041,  783.33692083, 1033.23248759,
        797.416425  ,  779.9476916 ])
```

```python
#EVALUATION
from sklearn import metrics

metrics.mean_squared_error(Y_test,Y_pred)
```

```
192915.74724032867
```

```python
import numpy as np
np.sqrt(metrics.mean_squared_error(Y_test,Y_pred))
```

```
439.22175178414
```

```python
r2=metrics.r2_score(Y_test,Y_pred)
n=df.shape[0] #sample size
p=df.shape[1] # columns/independent variables

df.shape
```

```
(1143, 13)
```

```python
#PERCENTAGE ACCURATE
r2=metrics.r2_score(Y_test,Y_pred)
r2
```

```
0.14708477569668355
```

```python
Adj_r2 = 1-(1-r2)*(n-1)/(n-p-1)
```

```
Adj_r2

0.13726378551427154

#LOGISTIC REGRESSION
x=df.iloc[:,:-2].values
y=df.iloc[:,:-2].values

from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=.25,rando
m_state=67)

from sklearn.preprocessing import StandardScaler
sc=StandardScaler()

X_train=sc.fit_transform(X_train)

X_test=sc.transform(X_test)

#MODELLING
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()

lr.fit(X_train,Y_train)

C:\Users\Lenovo\anaconda3\Lib\site-packages\sklearn\linear_model\
_logistic.py:460: 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()

Y_pred=lr.predict(X_test)

Y_pred

array([ 418,  667,  311, 1300, 1497, 1336, 1420, 1356,  949,  285,
23,
       1472,  654, 1206,  285, 1304,  422,  499,  311, 1419,   64,
848,
        736, 1347,   34, 1049, 1332,  513, 1021,  145,  756, 1013,
1230,
        988, 1452, 1122,  971,  806, 1059,  694, 1091,  459,  102,
```

```
1078,
        120,   61,  439,  269,  955, 1237,  296,  703,  495, 1156,
1026,
        946,    8, 1078, 1416,  188, 1332,  504,  285, 1155,  222,
122,
       1416,  950, 1191, 1419, 1449,  298, 1206,  106, 1006, 1533,
348,
       1122, 1021, 1178, 1451,  956,    4, 1472, 1300, 1238,  689,
1388,
        523,   51, 1416,    8, 1173, 1156, 1147, 1189,  848,  156,
644,
        253,  500,  927,  355,  733, 1100,  170,  172,  446,  754,
425,
        610,  813,  435, 1449,  298,  806, 1336,  545, 1081, 1132,
308,
       1168,  200,  949, 1509,  311, 1206,  689,  620,  170,  464,
8,
        860, 1060, 1480,   52,  298,   80,  440, 1551,  429,  440,
1110,
        144,  153,  872, 1133, 1591,  268,  529,  872, 1515,  165,
1115,
        407, 1066, 1351, 1449,  182,  974,  595,   80, 1468,  842,
348,
          4,  244,  264,  513, 1058,  983, 1061,  765,  170,  450,
1334,
        446, 1197,  655,  626,  359,  469,  872,  467,  278, 1175,
982,
        884,  245, 1057,  851,  188,  515, 1529,  455, 1460,  523,
1178,
         80,  956,  639,  495,  472, 1108, 1321, 1277,   88, 1416,
872,
        244, 1052, 1509, 1275, 1476,  170,  495,  981,  580, 1476,
1472,
        790,  103,  658,  172, 1416, 1078,   59, 1412,  164, 1010,
952,
        427,  128, 1140, 1115,  343,  697, 1206, 1078,  268,   54,
491,
        967,  134,  710,  542,  244,  504,  733, 1260, 1132, 1480,
491,
        756,  419, 1172,  657,  170,  927, 1419,   43, 1142, 1100,
1494,
       1201,  407, 1362, 1085,    7, 1143, 1181, 1070,  311,  625,
245,
       1237, 1156,   23,  610,  351, 1376, 1577,  513, 1472,  971,
128],
      dtype=int64)
```

```
Y_test
```

```
array([ 241,  669, 1315,  907, 1499, 1337,  262,  150, 1192,  321,
717,
       1566,   56, 1584,  284, 1384, 1312,  493, 1330,  417,   65,
846,
       1012, 1438,   25, 1048, 1345,  326, 1020,  130, 1557,  995,
1198,
        990,  958, 1272,  547, 1459, 1056,  693, 1148,  517,   99,
1190,
       1194,   46,  221,  520, 1458, 1234,  714,   82, 1035, 1408,
1490,
        378,  762,  911,  476, 1397,  598,  369,  319, 1152, 1253,
158,
       1414, 1093,  916, 1400,   37,  297, 1204,  151, 1007, 1457,
552,
       1045,  910,  553,  875, 1504,   28,   84, 1573,  576,  483,
1257,
         53,  148,  619,   29, 1174, 1090,  799,  704,  107,  215,
642,
        843,  116,   73, 1301,   21, 1104,  602,  630,  470,   19,
421,
        563,  939,  611, 1549,  734,  929, 1328, 1262, 1079,  896,
568,
       1044,  380,  857,  900, 1216, 1209,  773,  621,  732,  544,
235,
        863,  301,   12,  550,  302,   26,  358, 1553,  397, 1067,
1130,
        142,  146,   85,  904, 1587,  276,  179, 1523, 1514,  304,
1117,
        205, 1534, 1463, 1544,  246, 1086, 1071,  143, 1466, 1109,
551,
          0, 1224,  363,  514,  471,  980, 1202,  764,  217,  449,
1252,
        356, 1200,  665,  456,  391,  635, 1430,  210,  413,  908,
1083,
        607,  847, 1138,  850,  255,  415,  436,  335,  225,  744,
94,
         87, 1507,  372,  481,  814, 1032,  695, 1065,  723,  256,
1004,
        294, 1538, 1483, 1055, 1474,  688,  498, 1164,  581, 1434,
1508,
       1528,  105,  660,  628,  631, 1276,  746, 1406,  163, 1005,
826,
        223, 1403, 1082, 1116,  342,  701, 1208,  541,  546,  242,
1016,
        789, 1282,  700,   50,  243,  377,  196,  181, 1167, 1488,
492,
       1455,  266, 1136,  509,  677,   41, 1383,   13, 1530,  968,
1526,
        978,  206, 1359, 1251,  162,  371, 1322,  797, 1329,  719,
249,
```

```
       1064,  918,  766,  186, 1025, 1470,  740,  357, 1576, 1302,
69],
      dtype=int64)
```

```python
from sklearn.metrics import confusion_matrix
```

```python
confusion_matrix(Y_test,Y_pred)
```

```
array([[0, 1, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 1],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```