```
# 21BRS1617
# Sukanth K

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

# Loading the dataset

df=pd.read_csv('/content/winequality-red.csv')
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
...             ...               ...          ...             ...
...
1594            6.2             0.600         0.08             2.0
0.090
1595            5.9             0.550         0.10             2.2
0.062
1596            6.3             0.510         0.13             2.3
0.076
1597            5.9             0.645         0.12             2.0
0.075
1598            6.0             0.310         0.47             3.6
0.067

      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
```

```
...                   ...                   ...      ...   ...
...
1594                  32.0                  44.0  0.99490  3.45
0.58
1595                  39.0                  51.0  0.99512  3.52
0.76
1596                  29.0                  40.0  0.99574  3.42
0.75
1597                  32.0                  44.0  0.99547  3.57
0.71
1598                  18.0                  42.0  0.99549  3.39
0.66

      alcohol  quality
0         9.4        5
1         9.8        5
2         9.8        5
3         9.8        6
4         9.4        5
...       ...      ...
1594     10.5        5
1595     11.2        6
1596     11.0        6
1597     10.2        5
1598     11.0        6

[1599 rows x 12 columns]
```

```
df.shape
```

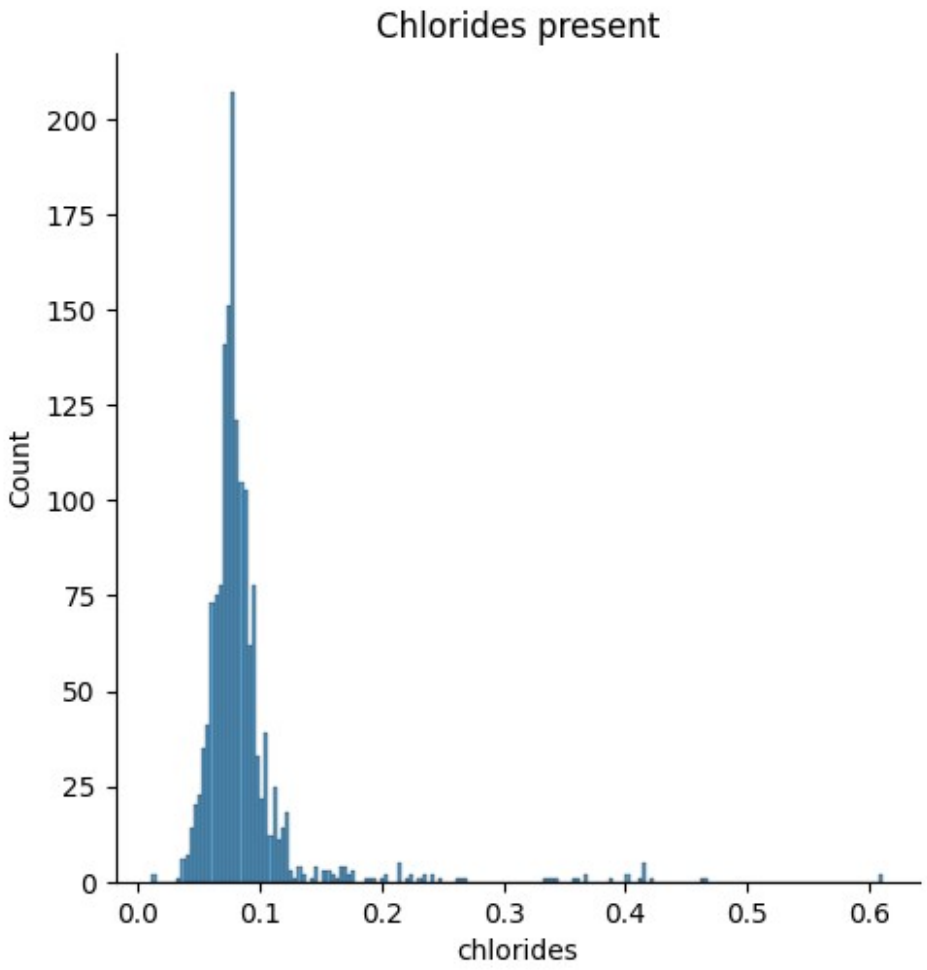
```
# Data Preprocessing and Visualisation

# 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
dtype: int64
```

```
df.isnull().any()
```

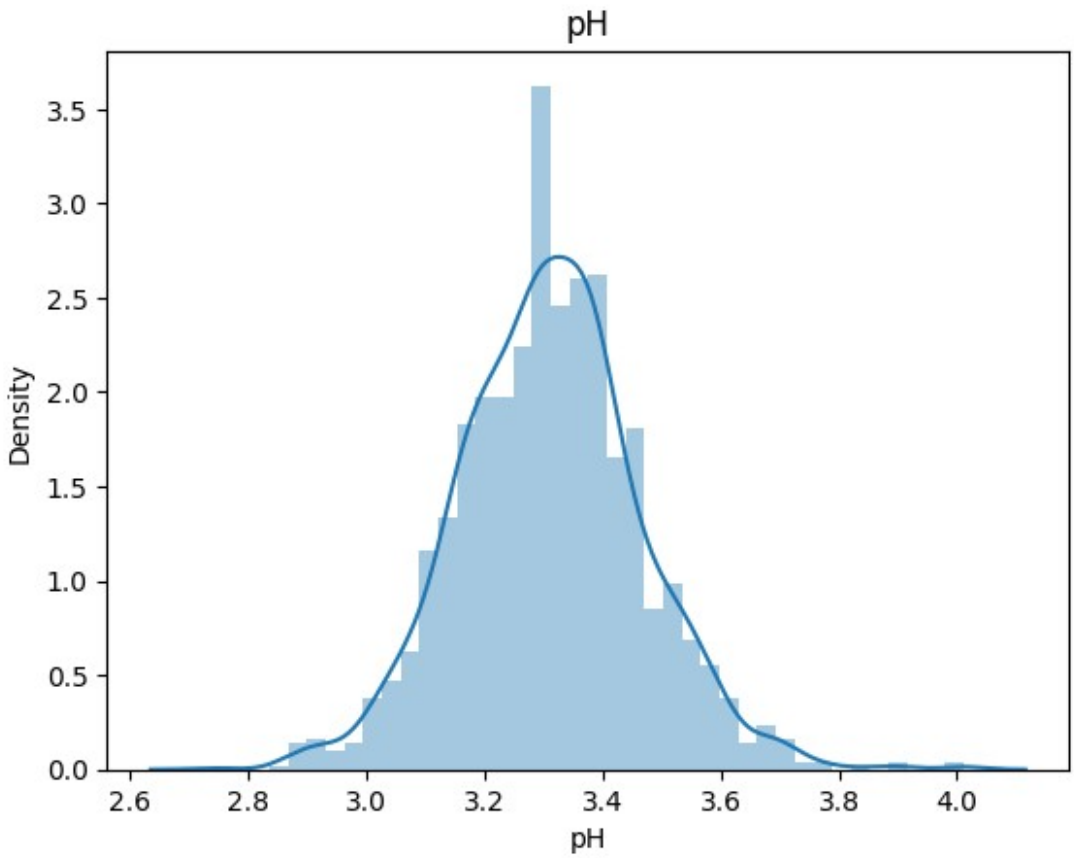
```
fixed acidity           False
volatile acidity        False
citric acid             False
residual sugar          False
chlorides               False
free sulfur dioxide     False
total sulfur dioxide    False
density                 False
pH                      False
sulphates               False
alcohol                 False
quality                 False
dtype: bool
```

```
# Data Visualisation

# Univariate

sns.displot(df['chlorides'])
plt.title('Chlorides present')
plt.show()
```



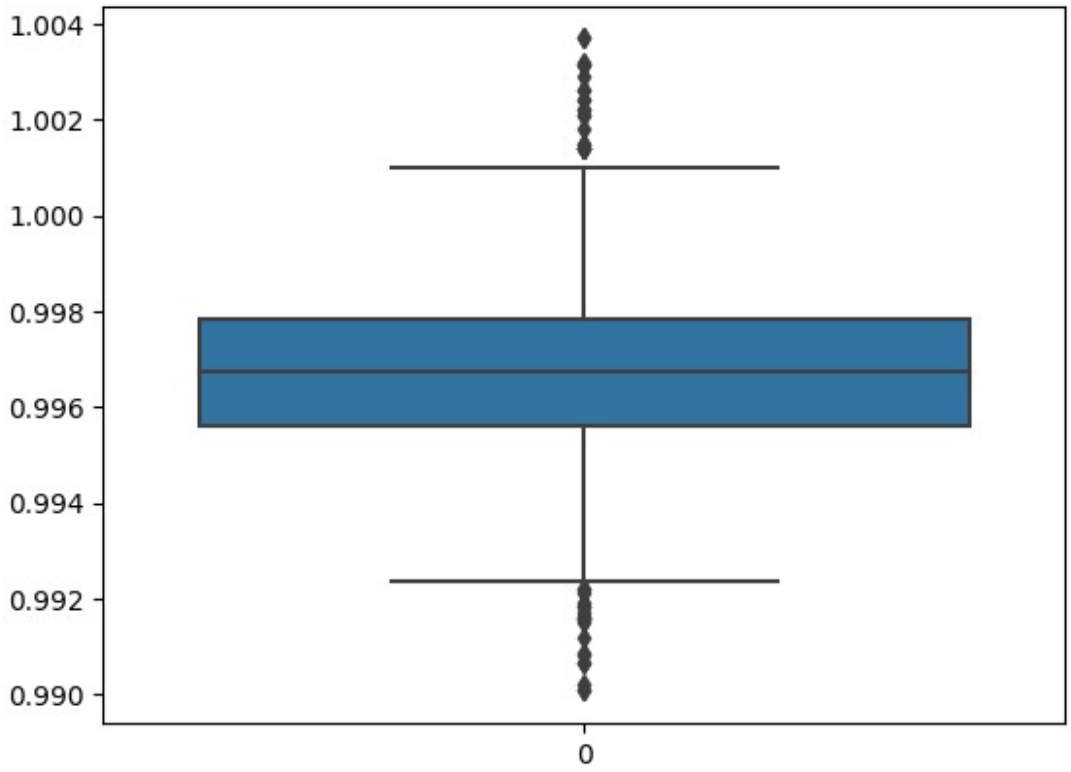

```
sns.distplot(df['pH'])
plt.title('pH')
plt.show()
```

```
<ipython-input-9-7af5d83ca87c>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['pH'])
```

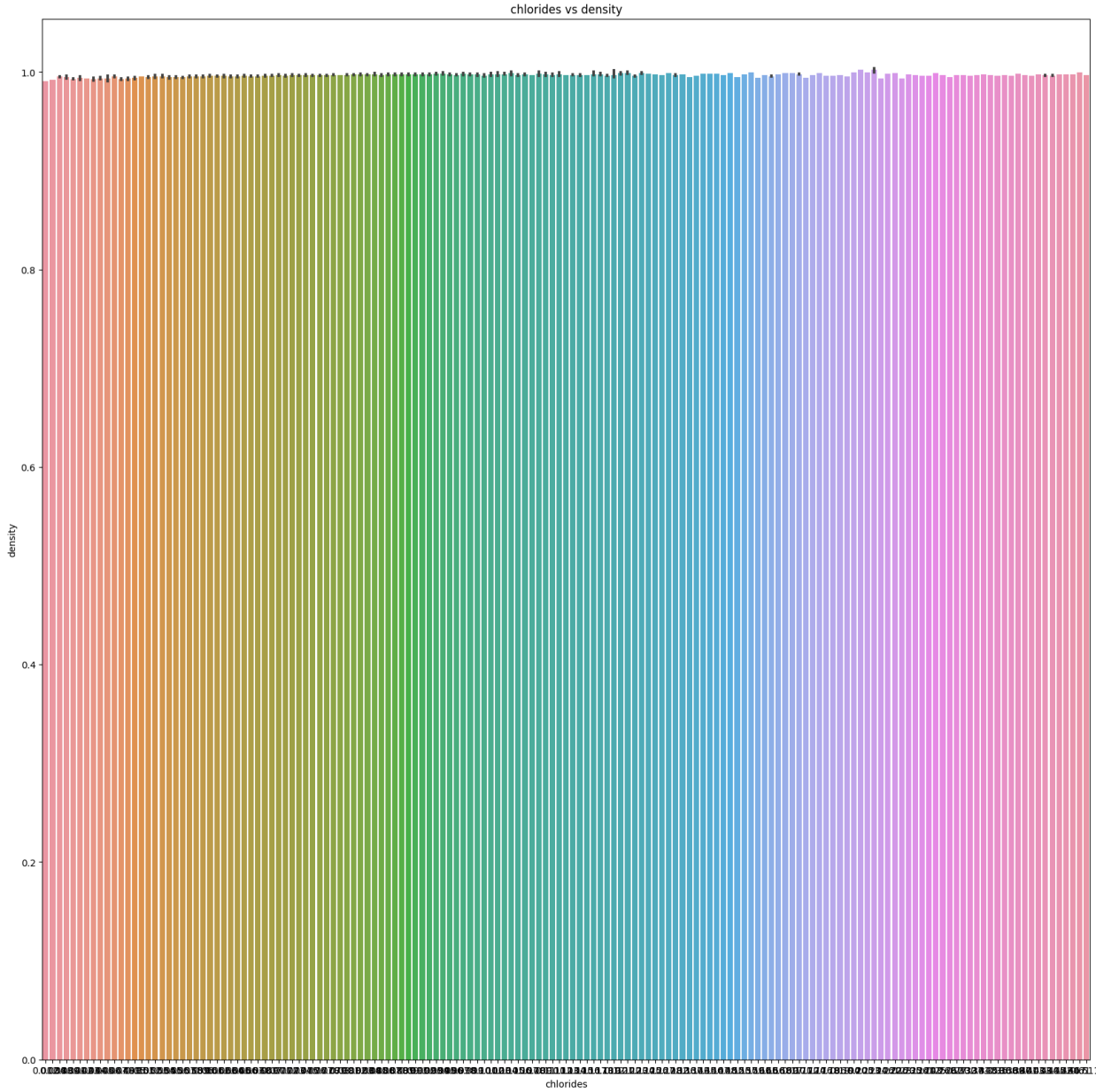



```
sns.boxplot(df['density'])
```

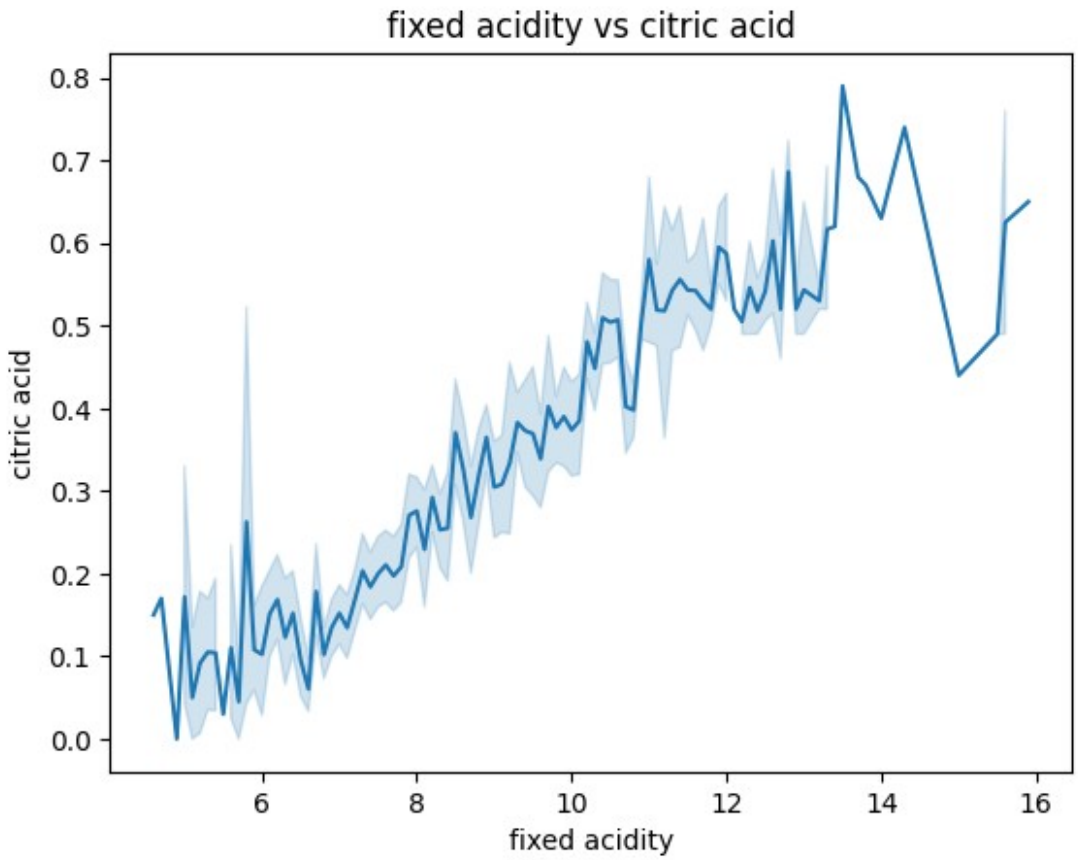
```
<Axes: >
```



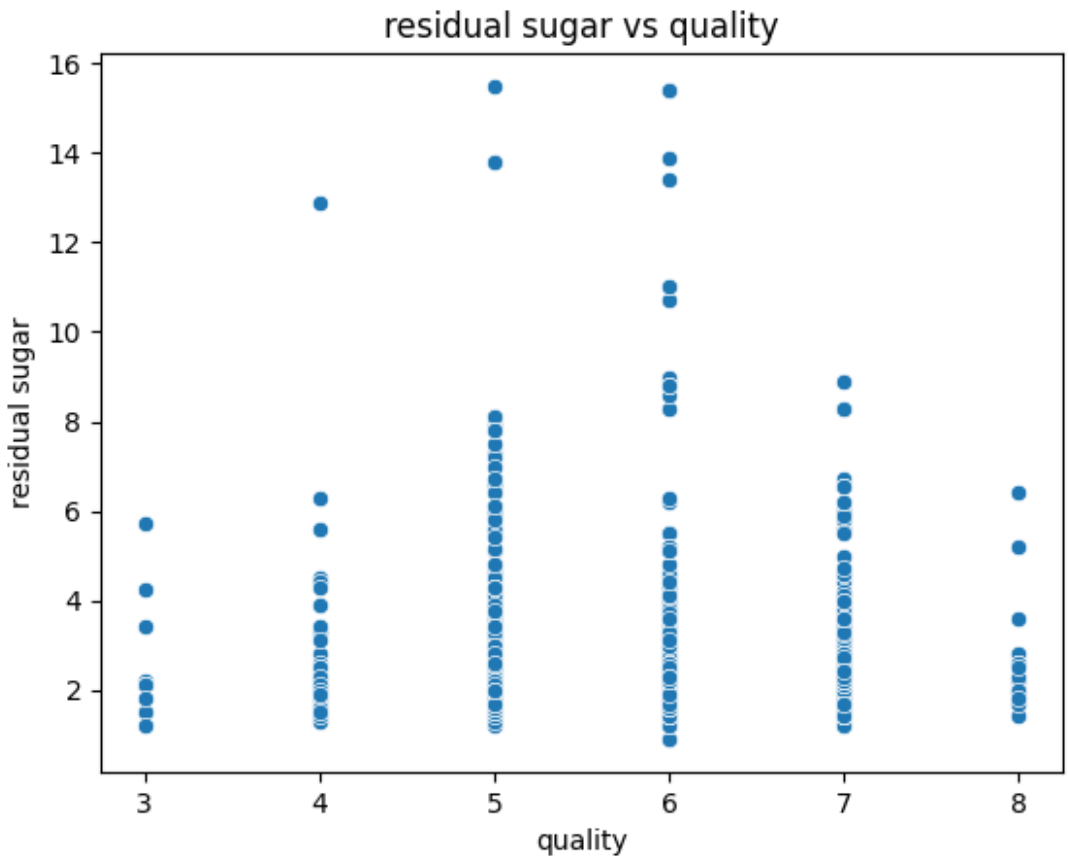

```
# Bivariate

plt.figure(figsize=(20,20))
sns.barplot(x=df['chlorides'],y=df['density'])
plt.title('chlorides vs density')
plt.show()
```




```
sns.lineplot(x=df['fixed acidity'],y=df['citric acid'])
plt.title('fixed acidity vs citric acid')
plt.show()
```

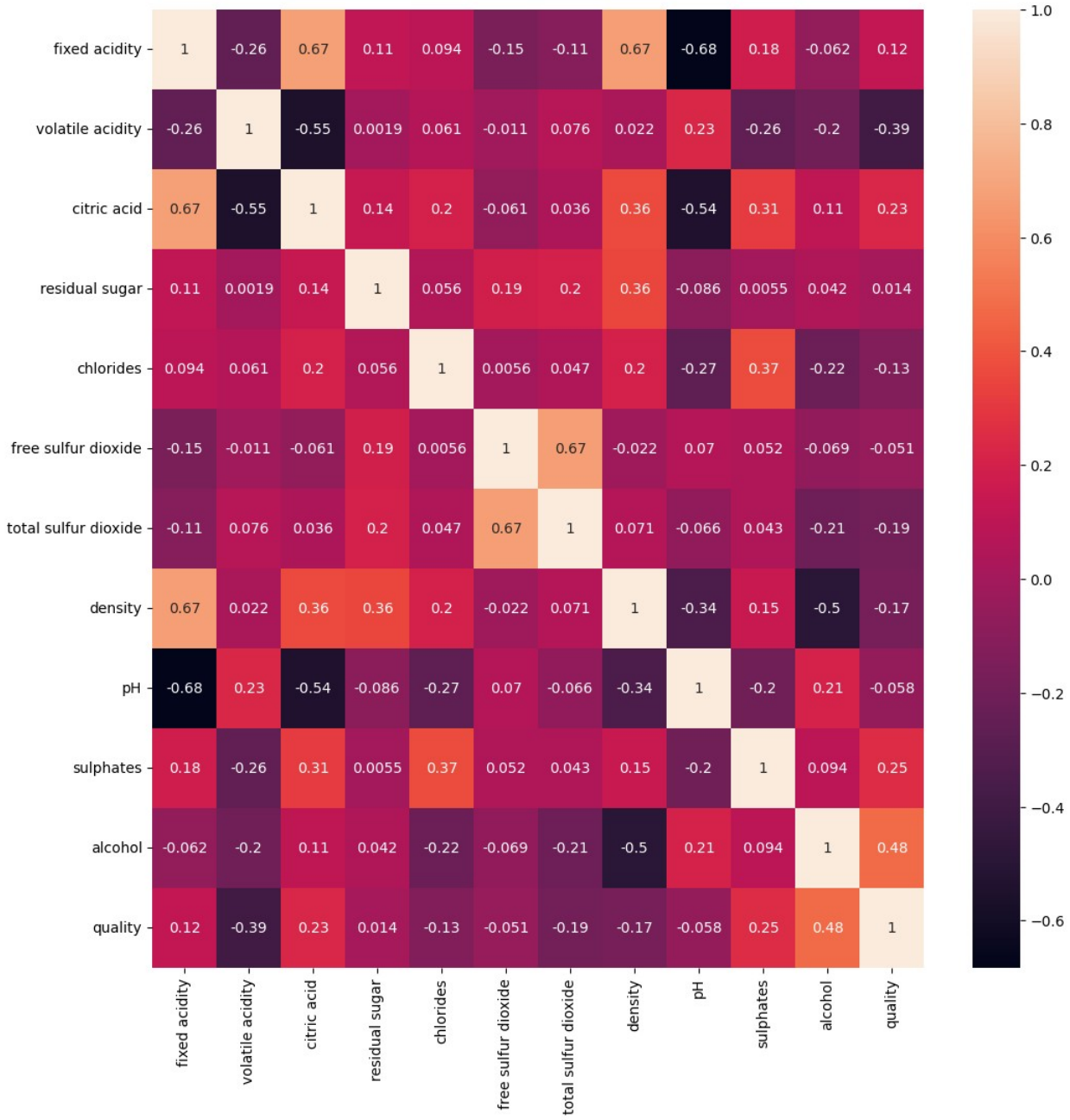



```
sns.scatterplot(x=df['quality'],y=df['residual sugar'])
plt.title('residual sugar vs quality')
plt.show()
```




```
# Multivariate

plt.figure(figsize=(12,12))
sns.heatmap(df.corr(),annot=True)
```

```
<Axes: >
```



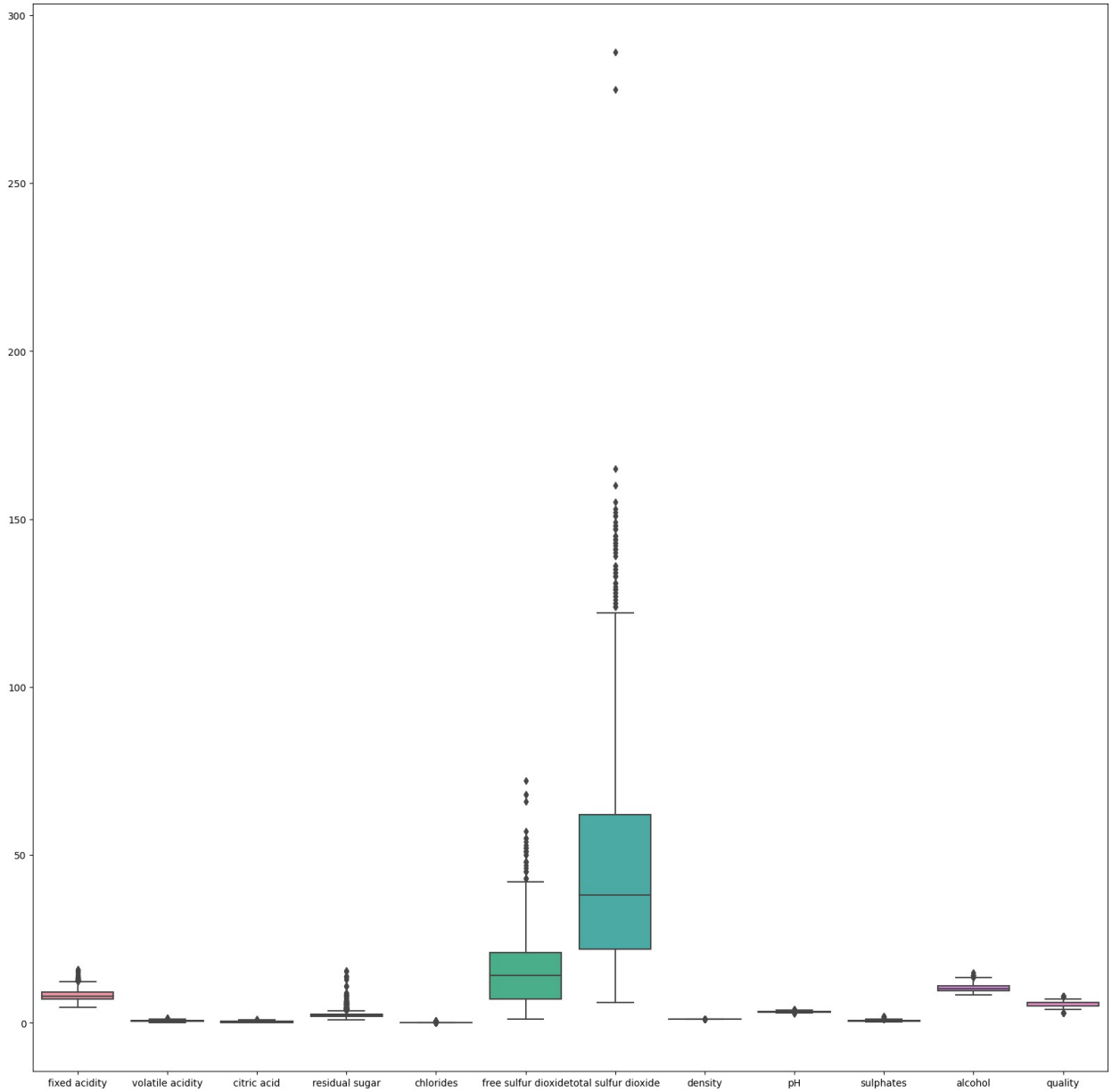

```
# Correlation with respect to target

df.corr().quality.sort_values(ascending=False)
```

```
quality                1.000000
alcohol                0.476166
sulphates              0.251397
citric acid            0.226373
fixed acidity          0.124052
residual sugar         0.013732
free sulfur dioxide   -0.050656
```

```
pH                      -0.057731
chlorides               -0.128907
density                 -0.174919
total sulfur dioxide    -0.185100
volatile acidity        -0.390558
Name: quality, dtype: float64
```

```
# Check for outliers
plt.figure(figsize=(20,20))
sns.boxplot(df)
```

```
<Axes: >
```



```
q1=df.alcohol.quantile(0.25)
q3=df.alcohol.quantile(0.75)

IQR=q3-q1
IQR
```

```
1.5999999999999996
```

```
ul=q3+1.5*IQR
ul
```

```
13.5
```

```
ll=q1-1.5*IQR
ll
```

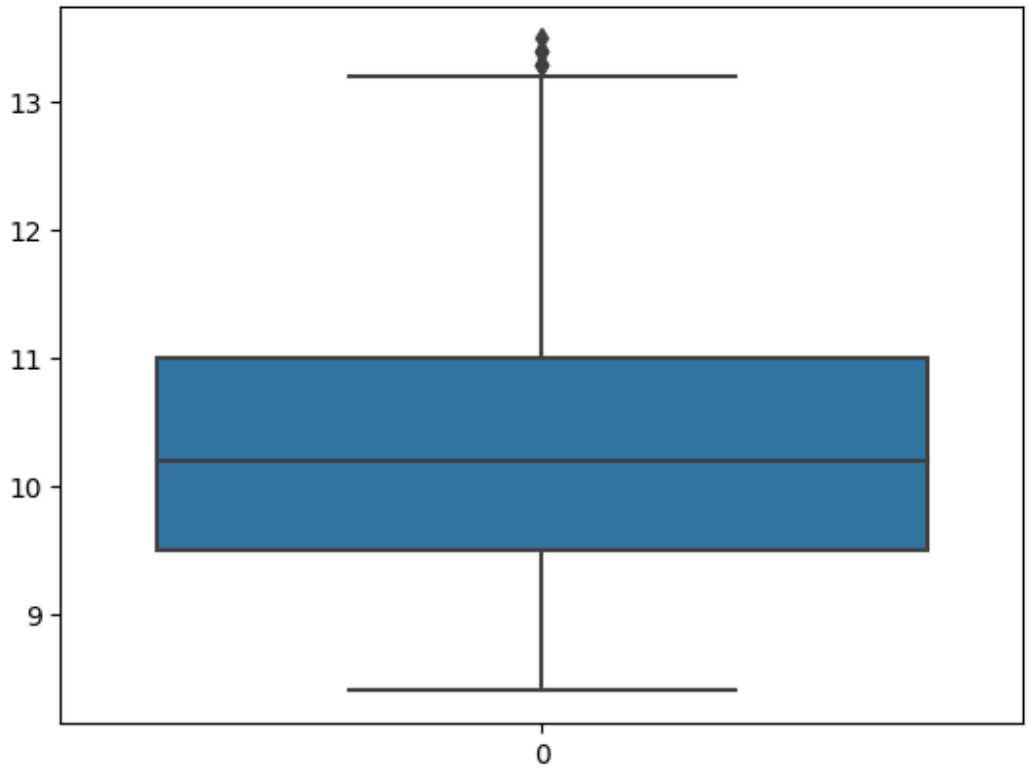
```
7.1000000000000005
```

```
df.alcohol.median()
```

```
10.2
```

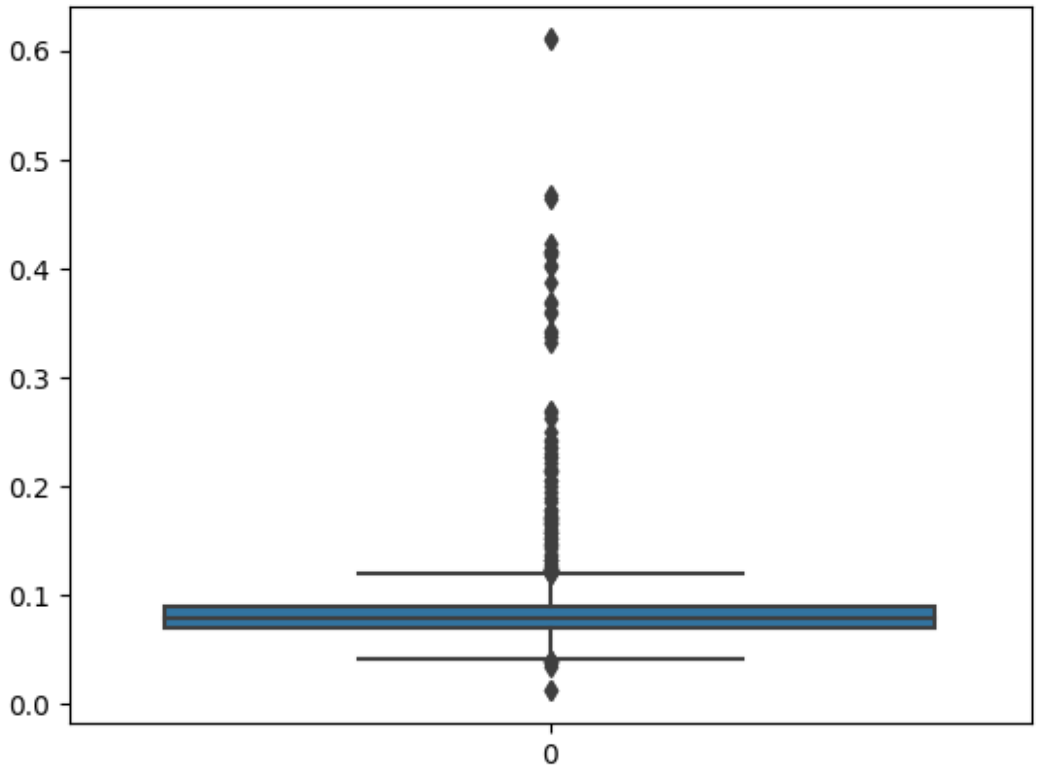
```
df['alcohol']=np.where(df['alcohol']>ul,df.alcohol.median(),df['alcohol'])
sns.boxplot(df['alcohol'])
```

```
<Axes: >
```



```
sns.boxplot(df['chlorides'])
```

```
<Axes: >
```

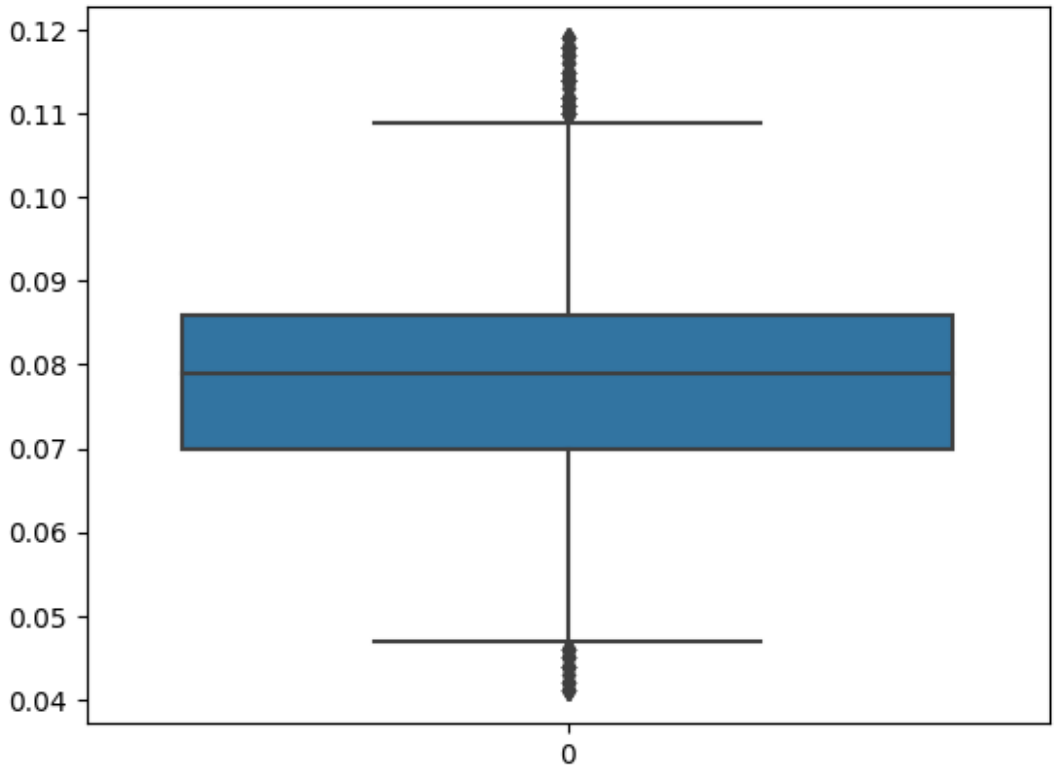



```
q1=df.chlorides.quantile(0.25)
q3=df.chlorides.quantile(0.75)

IQR=q3-q1
IQR
```

```
0.01999999999999999
```

```
ul=q3+1.5*IQR
ll=q1-1.5*IQR
df.chlorides=np.where(df.chlorides>ul,df.chlorides.median(),df.chlorid
es)
df.chlorides=np.where(df.chlorides<ll,df.chlorides.median(),df.chlorid
es)
sns.boxplot(df['chlorides'])
```

```
<Axes: >
```




```
q1=df['total sulfur dioxide'].quantile(0.25)
q3=df['total sulfur dioxide'].quantile(0.75)
IQR=q3-q1
IQR
```

```
40.0
```

```
ul=q3+1.5*IQR
ll=q1-1.5*IQR
df['total sulfur dioxide']=np.where(df['total sulfur
dioxide']>ul,df['total sulfur dioxide'].median(),df['total sulfur
dioxide'])
```

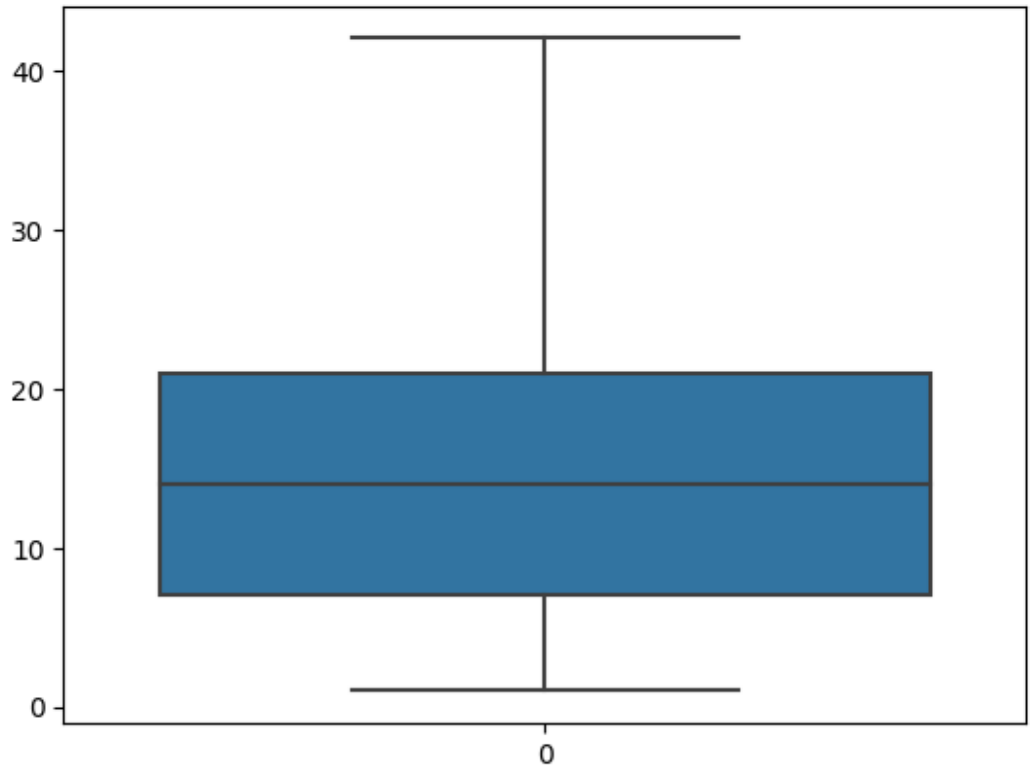
```
q1=df['free sulfur dioxide'].quantile(0.25)
q3=df['free sulfur dioxide'].quantile(0.75)
IQR=q3-q1
IQR
```

```
14.0
```

```
ul=q3+1.5*IQR
ll=q1-1.5*IQR
df['free sulfur dioxide']=np.where(df['free sulfur
dioxide']>ul,df['free sulfur dioxide'].median(),df['free sulfur
dioxide'])
```

```
sns.boxplot(df['free sulfur dioxide'])
```

```
<Axes: >
```

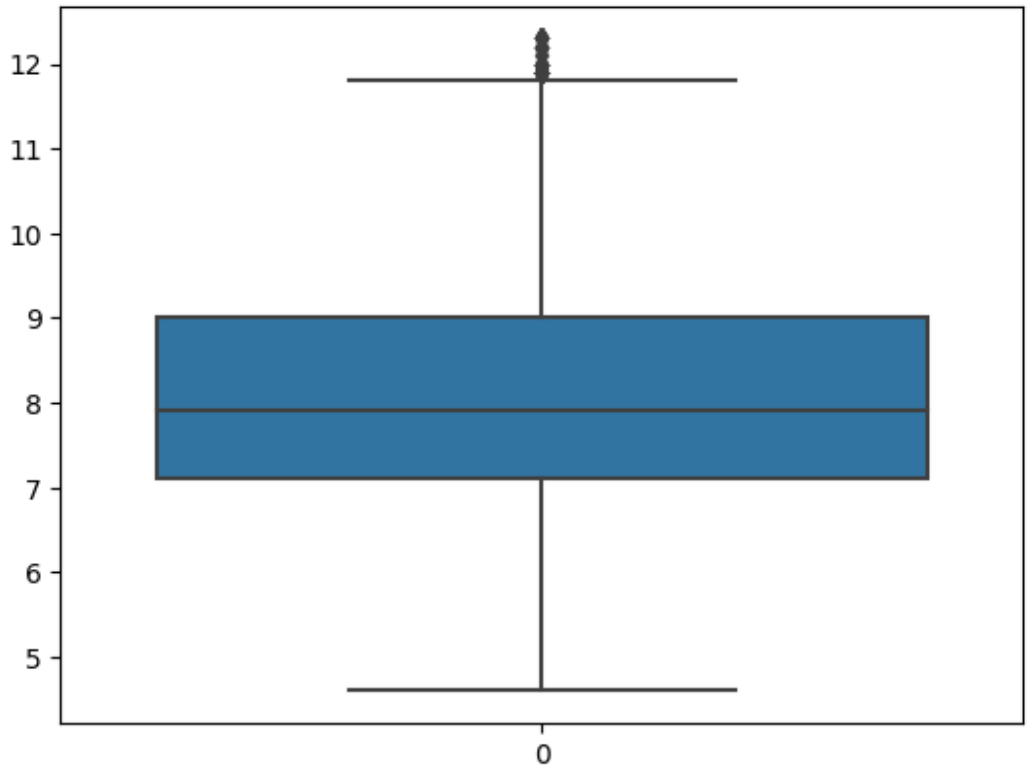



```
q1=df['fixed acidity'].quantile(0.25)
q3=df['fixed acidity'].quantile(0.75)
IQR=q3-q1
IQR
```

```
2.0999999999999996
```

```
ul=q3+1.5*IQR
ll=q1-1.5*IQR
df['fixed acidity']=np.where(df['fixed acidity']>ul,df['fixed
acidity'].median(),df['fixed acidity'])
sns.boxplot(df['fixed acidity'])
```

```
<Axes: >
```

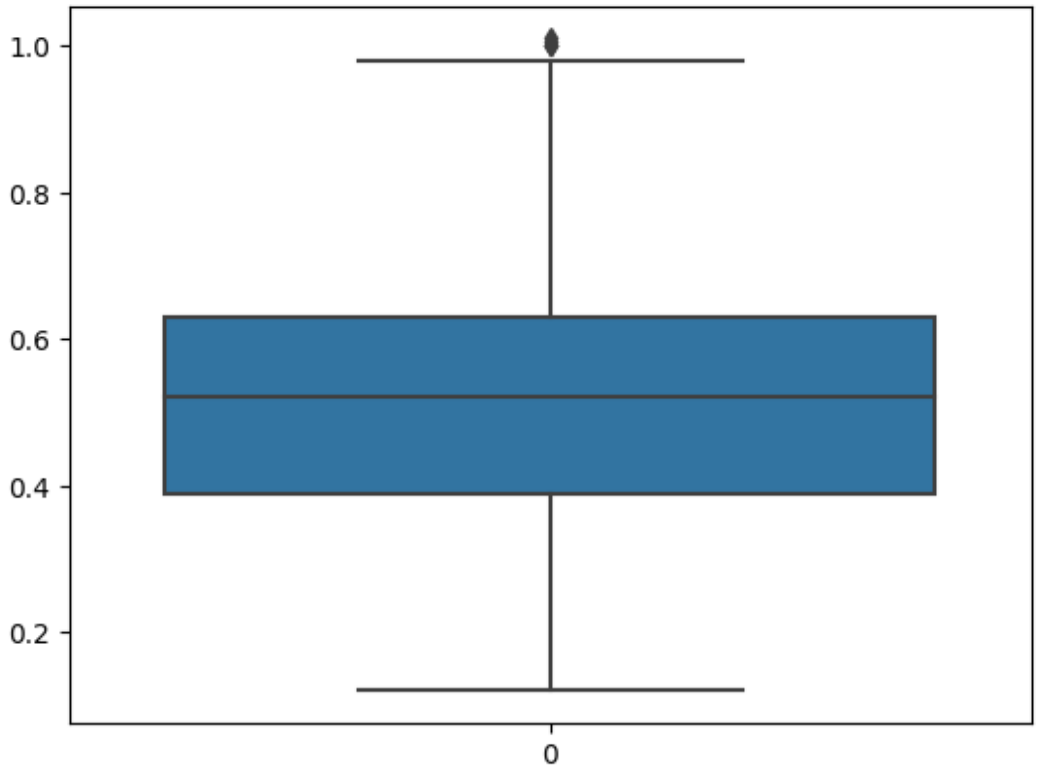



```
q1=df['volatile acidity'].quantile(0.25)
q3=df['volatile acidity'].quantile(0.75)
IQR=q3-q1
IQR
```

```
0.25
```

```
ul=q3+1.5*IQR
ll=q1-1.5*IQR
df['volatile acidity']=np.where(df['volatile acidity']>ul,df['volatile
acidity'].median(),df['volatile acidity'])

sns.boxplot(df['volatile acidity'])
```

```
<Axes: >
```




```
q1=df['citric acid'].quantile(0.25)
q3=df['citric acid'].quantile(0.75)
IQR=q3-q1
IQR
```

```
0.32999999999999996
```

```
ul=q3+1.5*IQR
ll=q1-1.5*IQR
df['citric acid']=np.where(df['citric acid']>ul,df['citric
acid'].median(),df['citric acid'])
```

```
q1=df['residual sugar'].quantile(0.25)
q3=df['residual sugar'].quantile(0.75)
IQR=q3-q1
IQR
```

```
0.7000000000000002
```

```
ul=q3+1.5*IQR
ll=q1-1.5*IQR
df['residual sugar']=np.where(df['residual sugar']>ul,df['residual
sugar'].median(),df['residual sugar'])
df['residual sugar']=np.where(df['residual sugar']<ll,df['residual
sugar'].median(),df['residual sugar'])
q1=df['pH'].quantile(0.25)
```

```
q3=df['pH'].quantile(0.75)
IQR=q3-q1
IQR
```

```
0.18999999999999995
```

```
ul=q3+1.5*IQR
ll=q1-1.5*IQR
df['pH']=np.where(df['pH']>ul,df['pH'].median(),df['pH'])
df['pH']=np.where(df['pH']<ll,df['pH'].median(),df['pH'])
q1=df['sulphates'].quantile(0.25)
q3=df['sulphates'].quantile(0.75)
IQR=q3-q1
IQR
```

```
0.17999999999999994
```

```
ul=q3+1.5*IQR
ll=q1-1.5*IQR
df['sulphates']=np.where(df['sulphates']>ul,df['sulphates'].median(),d
f['sulphates'])
df['sulphates']=np.where(df['sulphates']<ll,df['sulphates'].median(),d
f['sulphates'])
q1=df['density'].quantile(0.25)
q3=df['density'].quantile(0.75)
IQR=q3-q1
IQR
```

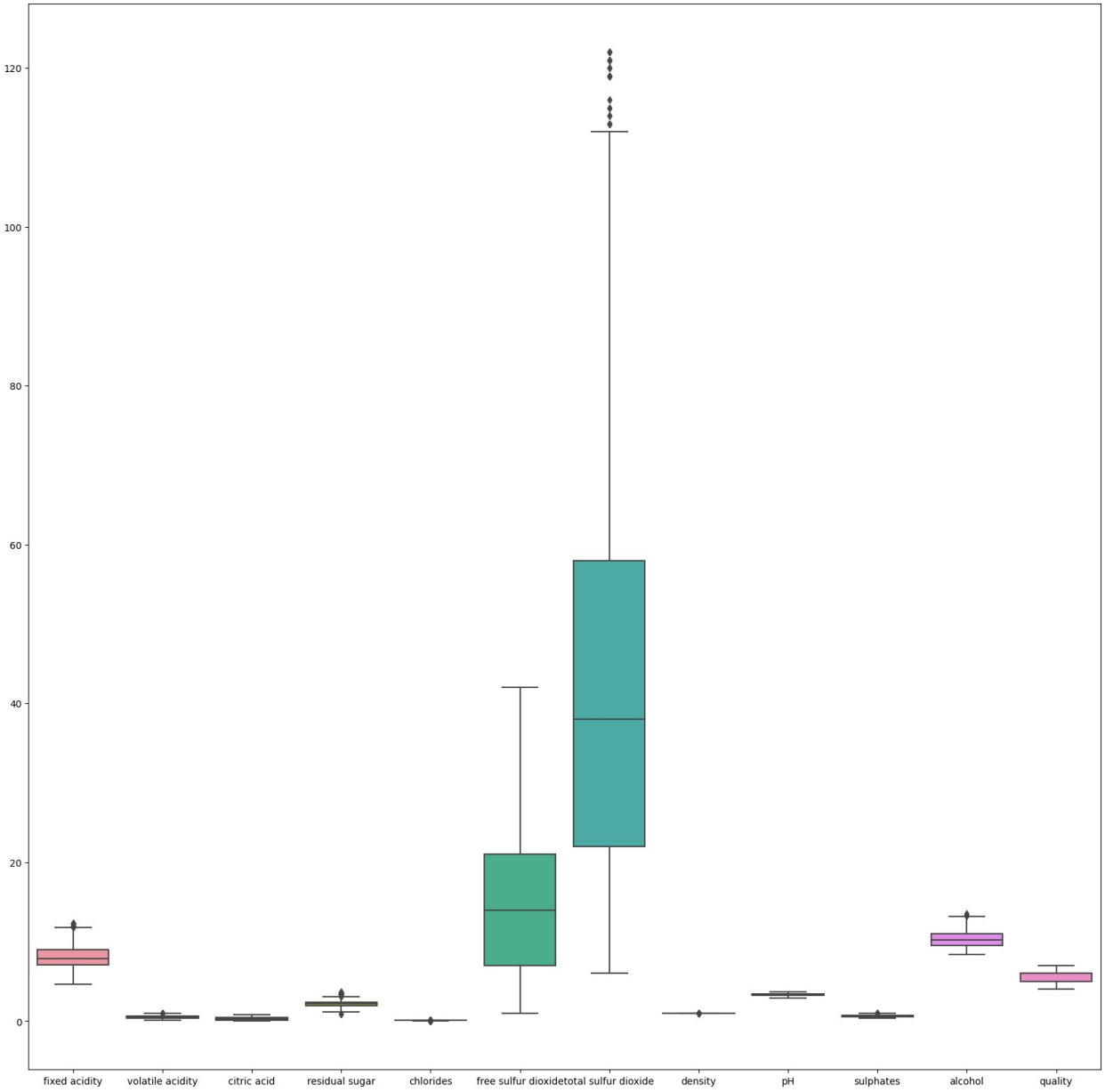
```
0.002234999999999987
```

```
ul=q3+1.5*IQR
ll=q1-1.5*IQR
df['density']=np.where(df['density']>ul,df['density'].median(),df['den
sity'])
df['density']=np.where(df['density']<ll,df['density'].median(),df['den
sity'])
q1=df['quality'].quantile(0.25)
q3=df['quality'].quantile(0.75)
IQR=q3-q1
IQR
```

```
1.0
```

```
ul=q3+1.5*IQR
ll=q1-1.5*IQR
df['quality']=np.where(df['quality']>ul,df['quality'].median(),df['qua
lity'])
df['quality']=np.where(df['quality']<ll,df['quality'].median(),df['qua
lity'])
plt.figure(figsize=(20,20))
sns.boxplot(df)
```

```
<Axes: >
```




```
# X and Y split

x=df.drop(columns=['quality'],axis=1)
y=df.quality
x.head()
```

```
   fixed acidity  volatile acidity  citric acid  residual sugar
chlorides  \
0            7.4              0.70         0.00             1.9
0.076
```

```
1            7.8              0.88         0.00             2.6
0.098
2            7.8              0.76         0.04             2.3
0.092
3           11.2              0.28         0.56             1.9
0.075
4            7.4              0.70         0.00             1.9
0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates
\
0                 11.0                  34.0   0.9978  3.51       0.56

1                 25.0                  67.0   0.9968  3.20       0.68

2                 15.0                  54.0   0.9970  3.26       0.65

3                 17.0                  60.0   0.9980  3.16       0.58

4                 11.0                  34.0   0.9978  3.51       0.56


   alcohol
0      9.4
1      9.8
2      9.8
3      9.8
4      9.4

y.head()

0    5.0
1    5.0
2    5.0
3    6.0
4    5.0
Name: quality, dtype: float64

# Scaling Independent Variables

from sklearn.preprocessing import MinMaxScaler
scale=MinMaxScaler()

x_scaled=pd.DataFrame(scale.fit_transform(x),columns=x.columns)

x_scaled

      fixed acidity  volatile acidity  citric acid  residual sugar
chlorides  \
0          0.363636          0.651685     0.000000        0.363636
0.448718
```

```
1          0.415584          0.853933     0.000000         0.618182
0.730769
2          0.415584          0.719101     0.050633         0.509091
0.653846
3          0.857143          0.179775     0.708861         0.363636
0.435897
4          0.363636          0.651685     0.000000         0.363636
0.448718
...             ...               ...          ...              ...
...
1594       0.207792          0.539326     0.101266         0.400000
0.628205
1595       0.168831          0.483146     0.126582         0.472727
0.269231
1596       0.220779          0.438202     0.164557         0.509091
0.448718
1597       0.168831          0.589888     0.151899         0.400000
0.435897
1598       0.181818          0.213483     0.594937         0.981818
0.333333

      free sulfur dioxide  total sulfur dioxide   density        pH  \
0                0.243902              0.241379  0.630058  0.773333
1                0.585366              0.525862  0.514451  0.360000
2                0.341463              0.413793  0.537572  0.440000
3                0.390244              0.465517  0.653179  0.306667
4                0.243902              0.241379  0.630058  0.773333
...                   ...                   ...       ...       ...
1594             0.756098              0.327586  0.294798  0.693333
1595             0.926829              0.387931  0.320231  0.786667
1596             0.682927              0.293103  0.391908  0.653333
1597             0.756098              0.327586  0.360694  0.853333
1598             0.414634              0.310345  0.363006  0.613333

      sulphates   alcohol
0      0.348485  0.196078
1      0.530303  0.274510
2      0.484848  0.274510
3      0.378788  0.274510
4      0.348485  0.196078
...         ...       ...
1594   0.378788  0.411765
1595   0.651515  0.549020
1596   0.636364  0.509804
1597   0.575758  0.352941
1598   0.500000  0.509804

[1599 rows x 11 columns]
```

```
# Train-test split

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_scaled,y,test_size=0.
3,random_state=10)
x_train.shape, x_test.shape
```

```
((1119, 11), (480, 11))
```

```
# Machine Learning Model Building

# Logistic Regression

from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr.fit(x_train,y_train)
```

```
LogisticRegression()
```

```
y_pred_train=lr.predict(x_train)
y_pred=lr.predict(x_test)
pd.DataFrame({'y_actual':y_test,'y_pred':y_pred})
```

```
      y_actual  y_pred
1518       5.0     6.0
1246       5.0     5.0
544        6.0     5.0
1343       6.0     6.0
428        5.0     5.0
...        ...     ...
174        5.0     5.0
387        6.0     5.0
1560       5.0     5.0
846        5.0     5.0
1567       5.0     5.0

[480 rows x 2 columns]
```

```
# Evaluate the model

from sklearn.metrics import
accuracy_score,classification_report,confusion_matrix
accuracy_score(y_train,y_pred_train)
accuracy_score(y_test,y_pred)
print(classification_report(y_test,y_pred))
pd.crosstab(y_test,y_pred)
```

```
              precision    recall  f1-score   support

         4.0       0.00      0.00      0.00        16
         5.0       0.66      0.67      0.67       210
```

```
         6.0       0.52      0.64      0.57       196
         7.0       0.48      0.22      0.31        58

    accuracy                           0.58       480
   macro avg       0.42      0.38      0.39       480
weighted avg       0.56      0.58      0.56       480


/usr/local/lib/python3.10/dist-packages/sklearn/metrics/
_classification.py:1344: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

col_0	5.0	6.0	7.0
quality			
4.0	12	4	0
5.0	140	70	0
6.0	57	125	14
7.0	2	43	13

```
# Decision Tree Algorithm

from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dt.fit(x_train,y_train)
dty_pred=dt.predict(x_test)
dty_pred
```

```
array([7., 5., 6., 6., 6., 6., 7., 5., 6., 6., 7., 7., 5., 5., 6., 4.,
6.,
       6., 6., 5., 5., 6., 6., 5., 5., 6., 5., 6., 6., 5., 5., 5., 6.,
7.,
       5., 5., 7., 5., 6., 5., 6., 6., 5., 6., 5., 7., 6., 7., 6., 5.,
6.,
       7., 6., 7., 5., 5., 7., 5., 5., 6., 5., 6., 7., 7., 5., 6., 6.,
6.,
       6., 5., 6., 5., 6., 6., 5., 5., 6., 7., 5., 6., 7., 5., 7., 5.,
6.,
       6., 5., 5., 6., 5., 7., 7., 7., 6., 5., 6., 5., 5., 6., 6., 7.,
```

```
6.,
       5., 6., 6., 6., 5., 6., 6., 5., 5., 6., 7., 5., 5., 5., 5., 6.,
7.,
       6., 6., 5., 6., 4., 6., 5., 5., 5., 5., 6., 6., 5., 5., 4., 6.,
7.,
       6., 5., 5., 7., 6., 6., 6., 6., 6., 6., 6., 5., 5., 6., 6., 5.,
5.,
       7., 6., 6., 5., 4., 5., 7., 7., 7., 7., 5., 5., 4., 6., 7., 7.,
5.,
       5., 7., 6., 6., 5., 6., 6., 5., 4., 6., 5., 6., 5., 5., 5., 5.,
4.,
       6., 5., 6., 6., 5., 6., 5., 6., 6., 4., 5., 6., 6., 6., 6., 7.,
6.,
       5., 5., 5., 5., 6., 7., 5., 6., 6., 5., 7., 6., 6., 5., 5., 6.,
5.,
       7., 5., 5., 6., 5., 6., 6., 5., 5., 6., 6., 6., 6., 5., 6., 5.,
6.,
       5., 5., 6., 5., 6., 7., 6., 5., 5., 5., 6., 6., 5., 6., 6., 6.,
6.,
       6., 7., 7., 6., 6., 5., 5., 7., 5., 5., 4., 4., 7., 7., 6., 5.,
7.,
       6., 6., 5., 6., 5., 5., 6., 5., 7., 6., 7., 7., 5., 5., 7., 5.,
4.,
       6., 6., 5., 7., 6., 5., 7., 4., 5., 6., 6., 6., 6., 6., 6., 6.,
6.,
       6., 6., 6., 5., 6., 5., 6., 6., 6., 6., 7., 7., 5., 6., 6., 6.,
6.,
       7., 6., 7., 5., 5., 6., 6., 6., 6., 4., 7., 5., 5., 6., 5., 5.,
5.,
       6., 5., 6., 5., 6., 5., 6., 5., 4., 7., 5., 5., 7., 5., 5., 5.,
6.,
       5., 6., 6., 5., 6., 7., 5., 5., 5., 6., 5., 5., 6., 6., 5., 5.,
6.,
       6., 7., 6., 5., 5., 7., 6., 7., 4., 5., 5., 6., 4., 6., 7., 5.,
6.,
       7., 7., 6., 5., 5., 5., 6., 5., 7., 5., 7., 5., 5., 5., 5., 6.,
5.,
       6., 6., 6., 5., 6., 5., 5., 6., 5., 6., 6., 7., 5., 4., 5., 5.,
5.,
       7., 6., 6., 5., 7., 6., 6., 6., 6., 5., 7., 6., 6., 6., 5., 5.,
6.,
       6., 6., 6., 6., 5., 7., 5., 5., 6., 5., 5., 5., 6., 7., 6., 6.,
7.,
       5., 6., 5., 6., 7., 6., 6., 7., 6., 6., 6., 4., 5., 5., 5., 5.,
5.,
       5., 6., 5., 5.])

# Evaluating Decision Tree Algorithm

accuracy_score(y_test,dty_pred)
```

```
0.5791666666666667
```

```
print(classification_report(y_test,dty_pred))
```

```
              precision    recall  f1-score   support

         4.0       0.06      0.06      0.06        16
         5.0       0.70      0.62      0.66       210
         6.0       0.58      0.60      0.59       196
         7.0       0.41      0.52      0.45        58

    accuracy                           0.58       480
   macro avg       0.44      0.45      0.44       480
weighted avg       0.59      0.58      0.58       480
```

```
pd.crosstab(y_test,dty_pred)
```

```
col_0    4.0  5.0  6.0  7.0
quality
4.0        1   10    4    1
5.0        7  130   62   11
6.0        8   39  117   32
7.0        2    6   20   30
```

```
# Using random tree classifier

from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier()

rf.fit(x_train,y_train)

rfy_pred=rf.predict(x_test)

rfy_pred
```

```
array([6., 5., 6., 6., 6., 6., 6., 6., 6., 5., 7., 6., 5., 5., 6., 5.,
5.,
       6., 6., 5., 5., 6., 6., 5., 5., 6., 5., 6., 6., 5., 5., 5., 6.,
7.,
       5., 5., 7., 5., 6., 5., 6., 6., 5., 6., 5., 6., 6., 7., 6., 6.,
6.,
       7., 6., 7., 6., 6., 6., 5., 5., 5., 6., 5., 7., 6., 5., 6., 6.,
6.,
       5., 7., 5., 5., 5., 5., 6., 5., 5., 7., 5., 6., 7., 5., 6., 5.,
6.,
       6., 5., 5., 6., 5., 7., 6., 7., 6., 5., 6., 5., 5., 6., 6., 7.,
5.,
       5., 6., 6., 5., 5., 6., 6., 5., 5., 6., 7., 5., 5., 5., 5., 5.,
6.,
```

```
       6., 5., 5., 6., 5., 5., 5., 5., 5., 5., 6., 6., 5., 6., 5., 6.,
6.,
       6., 5., 5., 7., 6., 6., 5., 6., 6., 6., 5., 5., 6., 6., 5., 7.,
6.,
       7., 6., 6., 5., 5., 6., 6., 6., 6., 7., 5., 5., 6., 6., 7., 6.,
5.,
       5., 7., 6., 6., 5., 6., 5., 5., 5., 6., 6., 6., 6., 6., 5., 5.,
5.,
       6., 5., 5., 5., 5., 6., 6., 6., 6., 5., 5., 7., 6., 6., 6., 6.,
6.,
       5., 6., 5., 5., 5., 7., 6., 6., 6., 5., 5., 6., 6., 5., 5., 5.,
5.,
       6., 5., 6., 7., 5., 6., 5., 5., 5., 5., 6., 6., 5., 5., 6., 5.,
5.,
       5., 5., 6., 5., 5., 6., 6., 5., 5., 5., 5., 6., 5., 6., 5., 5.,
6.,
       6., 7., 7., 7., 6., 5., 5., 6., 5., 5., 5., 5., 6., 6., 6., 5.,
7.,
       6., 6., 5., 5., 5., 5., 6., 5., 6., 6., 6., 7., 4., 5., 6., 6.,
5.,
       6., 6., 5., 6., 6., 5., 7., 5., 6., 5., 6., 6., 6., 6., 6., 6.,
6.,
       5., 7., 6., 5., 6., 5., 6., 6., 6., 6., 7., 7., 5., 5., 6., 6.,
6.,
       7., 5., 6., 5., 5., 5., 5., 5., 7., 6., 6., 5., 5., 5., 5., 5.,
5.,
       6., 5., 6., 6., 6., 5., 6., 5., 5., 6., 5., 5., 7., 5., 5., 5.,
6.,
       5., 6., 5., 5., 6., 6., 5., 5., 5., 6., 5., 5., 6., 5., 5., 5.,
6.,
       6., 7., 6., 5., 5., 7., 6., 5., 6., 5., 5., 5., 5., 6., 6., 5.,
6.,
       6., 7., 6., 5., 5., 6., 6., 5., 6., 5., 6., 5., 5., 5., 5., 6.,
6.,
       6., 7., 6., 5., 6., 5., 5., 5., 5., 6., 6., 6., 5., 5., 6., 5.,
5.,
       6., 5., 6., 5., 7., 6., 6., 6., 6., 5., 6., 6., 6., 6., 5., 5.,
6.,
       6., 6., 6., 6., 5., 7., 5., 6., 6., 5., 6., 5., 5., 6., 5., 6.,
6.,
       5., 6., 5., 6., 7., 6., 6., 7., 6., 7., 5., 5., 5., 5., 6., 5.,
5.,
       5., 5., 5., 5.])
rfy_pred_train=rf.predict(x_train)
# Evaluation for random tree classifier
accuracy_score(y_train,rfy_pred_train)
```

```
1.0
```

```
accuracy_score(y_test,rfy_pred)
```

```
0.68125
```

```
accuracy_score(y_test,rfy_pred)
```

```
0.68125
```

```
pd.crosstab(y_test,rfy_pred)
```

```
col_0    4.0  5.0  6.0  7.0
quality
4.0        0   11    5    0
5.0        0  163   45    2
6.0        1   45  136   14
7.0        0    2   28   28
```

```
print(classification_report(y_test,rfy_pred))
```

```
              precision    recall  f1-score   support

         4.0       0.00      0.00      0.00        16
         5.0       0.74      0.78      0.76       210
         6.0       0.64      0.69      0.66       196
         7.0       0.64      0.48      0.55        58

    accuracy                           0.68       480
   macro avg       0.50      0.49      0.49       480
weighted avg       0.66      0.68      0.67       480
```

```
# Test with random observation
```

```
rf.predict([[7.4,0.28,0.04,2.6,0.076,25,38,0.9978,3.51,0.67,9.4]])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
```

```
array([5.])
```