Assignment 4

Charvi Upreti

charvi.upreti2021@vitstudent.ac.in (mailto:charvi.upreti2021@vitstudent.ac.in)

21BCE1440

ASSIGNMENT 4

Project Title:

Grapes to Greatness: Machine Learning in Wine Quality Prediction

Description:

Predicting wine quality using machine learning is a common and valuable application in the field of data science and analytics. Wine quality prediction involves building a model that can assess and predict the quality of a wine based on various input features, such as chemical composition, sensory characteristics, and environmental factors.

The two datasets are related to red and white variants of the Portuguese "Vinho Verde" wine. For more details, consult the reference [Cortez et al., 2009]. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.).

These datasets can be viewed as classification or regression tasks. The classes are ordered and not balanced (e.g. there are much more normal wines than excellent or poor ones).

Dataset: link

Task:

- Load the Dataset
- Data preprocessing including visualization
- Machine Learning Model building
- Evaluate the model
- Test with random observation

Importing required libraries

In [1]:
```
import pandas as pd
import numpy as np
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, classification_report, confus
from sklearn.linear_model import LinearRegression, Ridge, Lasso, Logistic
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from IPython.display import Image
import pydotplus
from io import StringIO
from sklearn.ensemble import RandomForestClassifier
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import rcParams
```

In [4]:
```
df.head().T
```

Out[4]:

	0	1	2	3	4
fixed acidity	7.4000	7.8000	7.800	11.200	7.4000
volatile acidity	0.7000	0.8800	0.760	0.280	0.7000
citric acid	0.0000	0.0000	0.040	0.560	0.0000
residual sugar	1.9000	2.6000	2.300	1.900	1.9000
chlorides	0.0760	0.0980	0.092	0.075	0.0760
free sulfur dioxide	11.0000	25.0000	15.000	17.000	11.0000
total sulfur dioxide	34.0000	67.0000	54.000	60.000	34.0000
density	0.9978	0.9968	0.997	0.998	0.9978
pH	3.5100	3.2000	3.260	3.160	3.5100
sulphates	0.5600	0.6800	0.650	0.580	0.5600
alcohol	9.4000	9.8000	9.800	9.800	9.4000
quality	5.0000	5.0000	5.000	6.000	5.0000

Loading the data

In [2]:
```
df=pd.read_csv("C:/Users/Charvi Upreti/Desktop/Assignments/Assignment 4/w
```

Data preprocessing and visualization

Data shape

In [3]:
```
df.shape
```

Out[3]: (1599, 12)

In [4]:
```
from matplotlib import rcParams
df.head().T
```

Out[4]:

	0	1	2	3	4
fixed acidity	7.4000	7.8000	7.800	11.200	7.4000
volatile acidity	0.7000	0.8800	0.760	0.280	0.7000
citric acid	0.0000	0.0000	0.040	0.560	0.0000
residual sugar	1.9000	2.6000	2.300	1.900	1.9000
chlorides	0.0760	0.0980	0.092	0.075	0.0760
free sulfur dioxide	11.0000	25.0000	15.000	17.000	11.0000
total sulfur dioxide	34.0000	67.0000	54.000	60.000	34.0000
density	0.9978	0.9968	0.997	0.998	0.9978
pH	3.5100	3.2000	3.260	3.160	3.5100
sulphates	0.5600	0.6800	0.650	0.580	0.5600
alcohol	9.4000	9.8000	9.800	9.800	9.4000
quality	5.0000	5.0000	5.000	6.000	5.0000

Loading the data

In [2]:
```
df=pd.read_csv("C:/Users/Charvi Upreti/Desktop/Assignments/Assignment 4/w
```

Data preprocessing and visualization

Data shape

In [3]:
```
df.shape
```

Out[3]: (1599, 12)

Information

In [5]:
```
df.info()
```

```
<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
```

Statistical description

In [6]:
```
df.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
fixed acidity	1599.0	8.319637	1.741096	4.60000	7.1000	7.90000	9.200000	15.90000
volatile acidity	1599.0	0.527821	0.179060	0.12000	0.3900	0.52000	0.640000	1.58000
citric acid	1599.0	0.270976	0.194801	0.00000	0.0900	0.26000	0.420000	1.00000
residual sugar	1599.0	2.538806	1.409928	0.90000	1.9000	2.20000	2.600000	15.50000
chlorides	1599.0	0.087467	0.047065	0.01200	0.0700	0.07900	0.090000	0.61100
free sulfur dioxide	1599.0	15.874922	10.460157	1.00000	7.0000	14.00000	21.000000	72.00000
total sulfur dioxide	1599.0	46.467792	32.895324	6.00000	22.0000	38.00000	62.000000	289.00000
density	1599.0	0.996747	0.001887	0.99007	0.9956	0.99675	0.997835	1.00369
pH	1599.0	3.311113	0.154386	2.74000	3.2100	3.31000	3.400000	4.01000
sulphates	1599.0	0.658149	0.169507	0.33000	0.5500	0.62000	0.730000	2.00000
alcohol	1599.0	10.422983	1.065668	8.40000	9.5000	10.20000	11.100000	14.90000
quality	1599.0	5.636023	0.807569	3.00000	5.0000	6.00000	6.000000	8.00000

Checking for null values

In [7]:
```
df.isnull().sum()
#no null values
```

Out[7]:
```
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 [8]:
```
df.quality.value_counts()
```

citric acid	1599.0	0.270976	0.194801	0.00000	0.0900	0.26000	0.420000	1.00000
residual sugar	1599.0	2.538806	1.409928	0.90000	1.9000	2.20000	2.600000	15.50000
chlorides	1599.0	0.087467	0.047065	0.01200	0.0700	0.07900	0.090000	0.61100
free sulfur dioxide	1599.0	15.874922	10.460157	1.00000	7.0000	14.00000	21.000000	72.00000
total sulfur dioxide	1599.0	46.467792	32.895324	6.00000	22.0000	38.00000	62.000000	289.00000
density	1599.0	0.996747	0.001887	0.99007	0.9956	0.99675	0.997835	1.00369
pH	1599.0	3.311113	0.154386	2.74000	3.2100	3.31000	3.400000	4.01000
sulphates	1599.0	0.658149	0.169507	0.33000	0.5500	0.62000	0.730000	2.00000
alcohol	1599.0	10.422983	1.065668	8.40000	9.5000	10.20000	11.100000	14.90000
quality	1599.0	5.636023	0.807569	3.00000	5.0000	6.00000	6.000000	8.00000

Checking for null values

In [7]:
```
df.isnull().sum()
#no NULL values
```

Out[7]:
```
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 [8]:
```
df.quality.value_counts()
```

Out[8]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

No categorical column to perform encoding

In [9]:
```
df.dtypes
```

Out[9]:
```
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 [10]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
#no need here
```

Duplicates

In [11]:
```
df.duplicated()
```

Out[11]:
```
0       False
1       False
2       False
3       False
4        True
        ...
1594    False
1595    False
1596     True
1597    False
1598    False
Length: 1599, dtype: bool
```

Index

In [14]:
```
df.index
```

Out[14]:
```
RangeIndex(start=0, stop=1599, step=1)
```

In [12]:
```
df1= df.drop_duplicates()
```

In [13]:
```
df1.shape
## We will not drop the duplicates as its greatly reducing the data set fi
```

Out[13]:
```
(1359, 12)
```

Out[11]:
```
0       False
1       False
2       False
3       False
4        True
        ...
1594    False
1595    False
1596     True
1597    False
1598    False
Length: 1599, dtype: bool
```

Index

In [14]:
```
df.index
```

Out[14]:
```
RangeIndex(start=0, stop=1599, step=1)
```

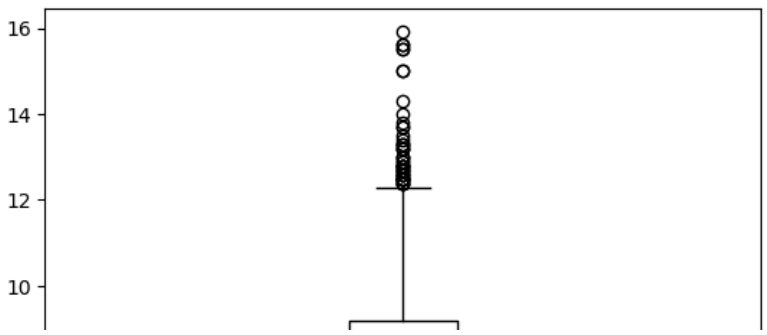
In [12]:
```
df1= df.drop_duplicates()
```

In [13]:
```
df1.shape
## We will not drop the duplicates as its greatly reducing the data set fi
```

Out[13]:
```
(1359, 12)
```

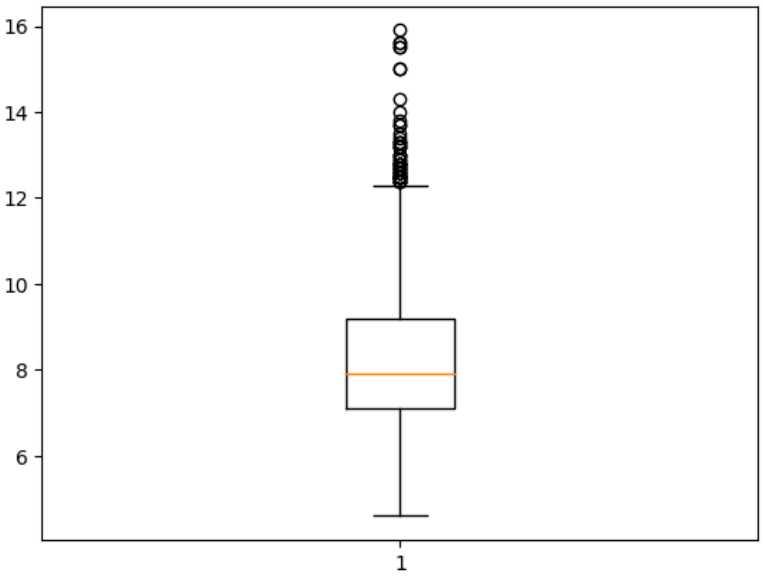
Outlier Detection and replacement

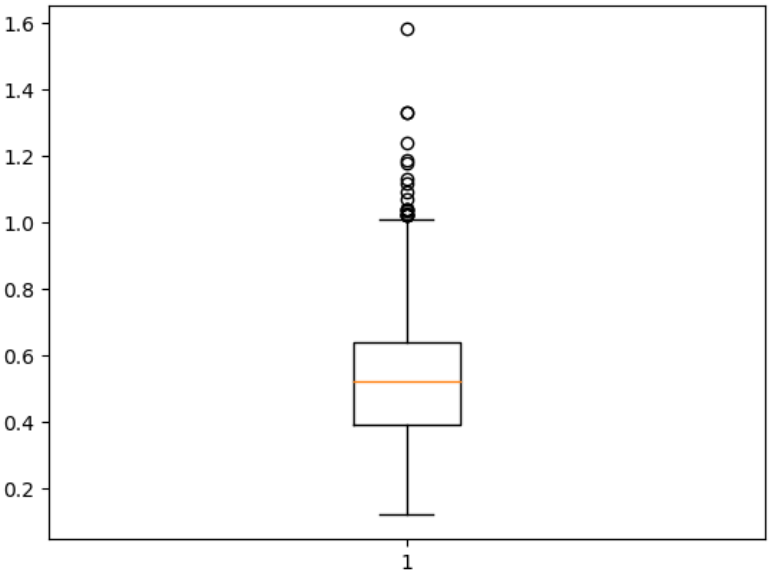
In [15]:
```
columns = df.columns
for col in columns:
    if col != 'quality':
        plt.figure()
        plt.boxplot(df[col])
```

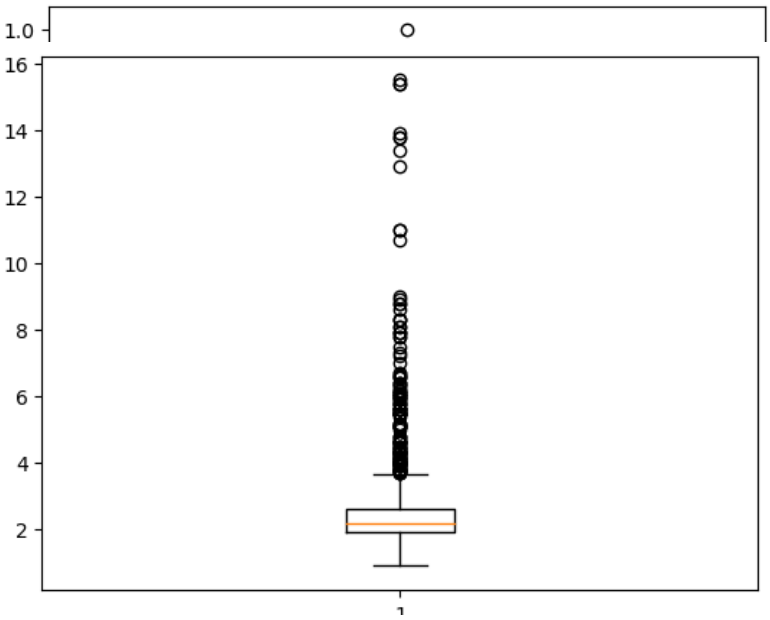


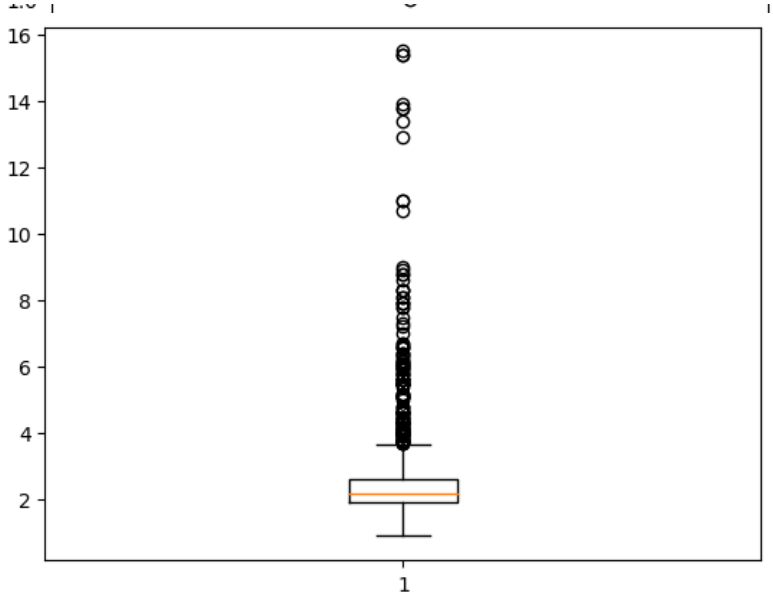
Outlier Detection and replacement

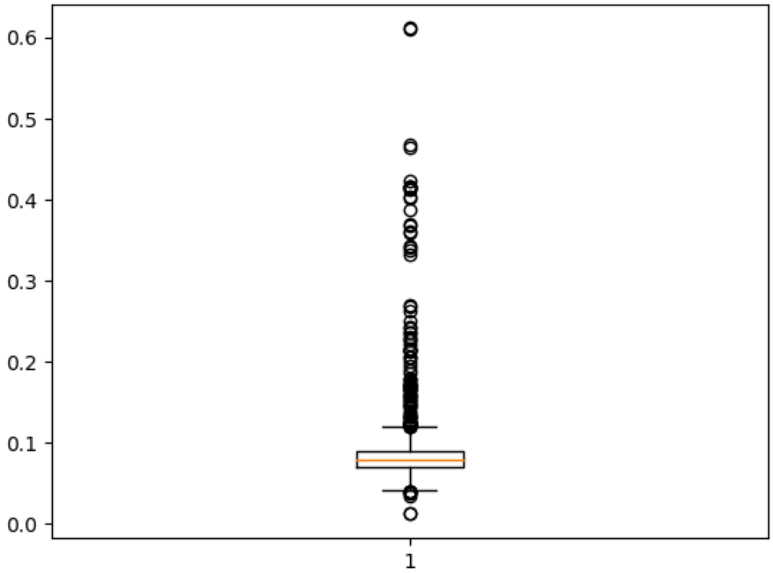
In [15]:
```
columns = df.columns
for col in columns:
    if col != 'quality':
        plt.figure()
        plt.boxplot(df[col])
```

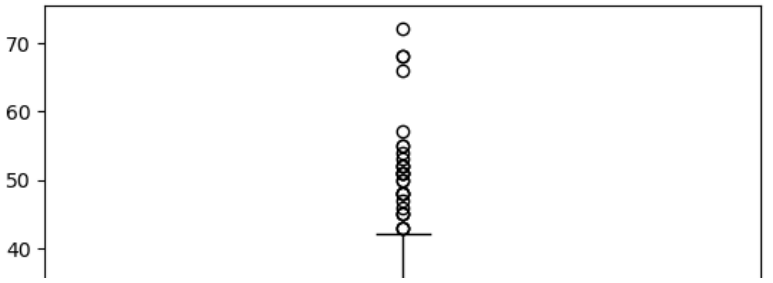


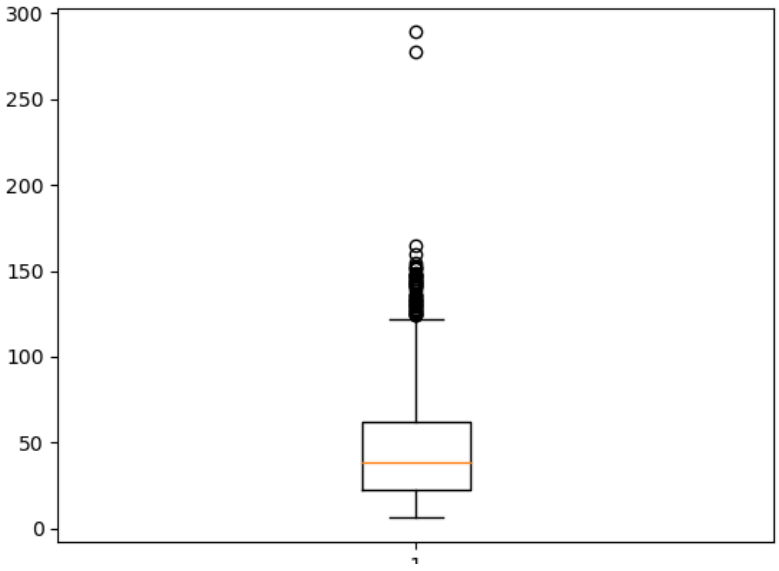


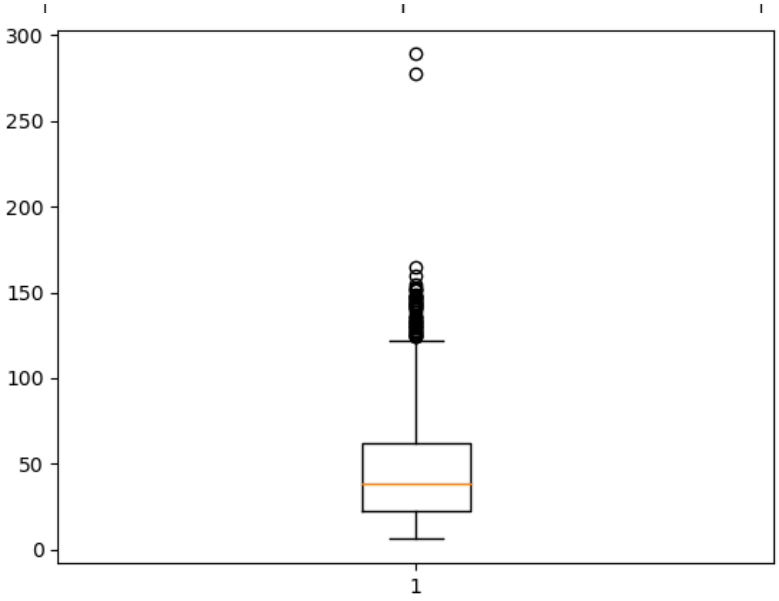


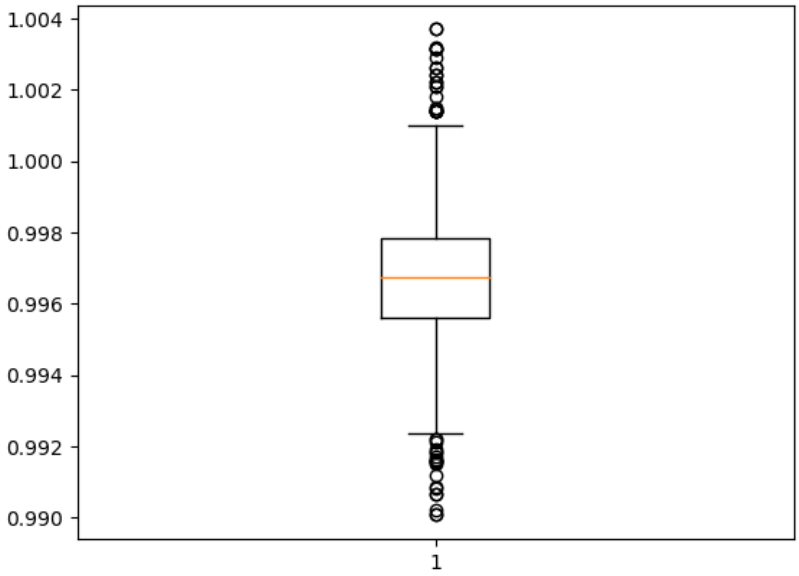

16
14
12
10
8
6
4
2
1

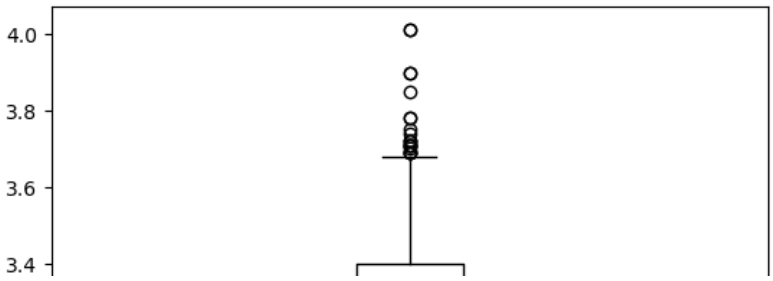

0.6
0.5
0.4
0.3
0.2
0.1
0.0
1

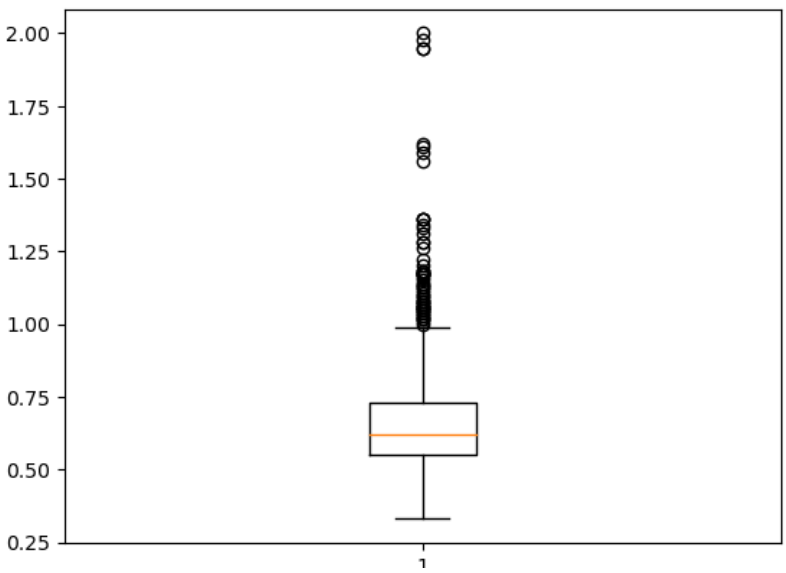

70
60
50
40

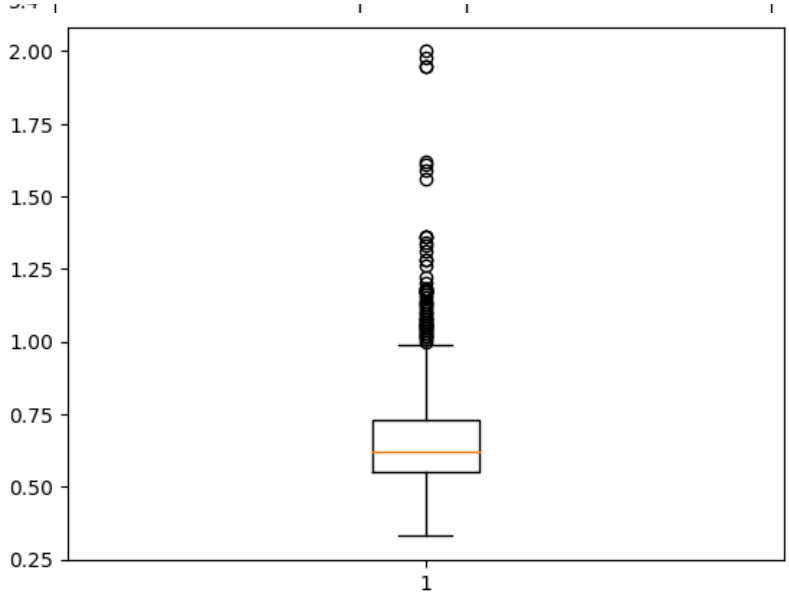

300
250
200
150
100
50
0

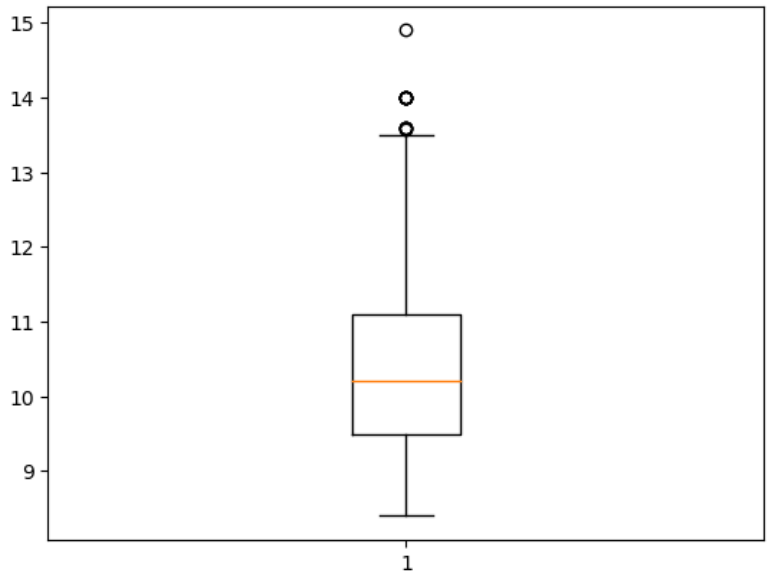

300
250
200
150
100
50
0
1


1.004
1.002
1.000
0.998
0.996
0.994
0.992
0.990
1


4.0
3.8
3.6
3.4


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
1





In [16]:
```
#Removing Outliers
for col in columns:
    if col != 'quality':
        q1 = df[col].quantile(0.25)
        q3 = df[col].quantile(0.75)
        IQR = q3 - q1
        upper_limit = q3 + 1.5 * IQR
        lower_limit = q1 - 1.5 * IQR  # Corrected this line
        median=df[col].median()
        df[col] = np.where(df[col] > upper_limit,median , df[col])
        df[col] = np.where(df[col] < lower_limit, median, df[col])

```

Correlation

In [17]:
```
df.corr().T
```

Out[17]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density
fixed acidity	1.000000	-0.258476	0.608682	0.225436	0.227814	-0.145813	-0.113789	0.557213
volatile acidity	-0.258476	1.000000	-0.564257	0.018550	0.138565	0.004853	0.090253	0.000445
citric acid	0.608682	-0.564257	1.000000	0.156708	0.091785	-0.060658	-0.010996	0.338521
residual sugar	0.225436	0.018550	0.156708	1.000000	0.229396	0.040756	0.126300	0.372047
chlorides	0.227814	0.138565	0.091785	0.229396	1.000000	-0.006299	0.099765	0.375573
free sulfur dioxide	-0.145813	0.004853	-0.060658	0.040756	-0.006299	1.000000	0.596640	-0.018912

```
df[col] = np.where(df[col] < lower_limit, median, df[col])
```

Correlation

In [17]:
```
df.corr().T
```

Out[17]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density
fixed acidity	1.000000	-0.258476	0.608682	0.225436	0.227814	-0.145813	-0.113789	0.557213
volatile acidity	-0.258476	1.000000	-0.564257	0.018550	0.138565	0.004853	0.090253	0.000445
citric acid	0.608682	-0.564257	1.000000	0.156708	0.091785	-0.060658	-0.010996	0.338521
residual sugar	0.225436	0.018550	0.156708	1.000000	0.229396	0.040756	0.126300	0.372047
chlorides	0.227814	0.138565	0.091785	0.229396	1.000000	-0.006299	0.099765	0.375573
free sulfur dioxide	-0.145813	0.004853	-0.060658	0.040756	-0.006299	1.000000	0.596640	-0.018912
total sulfur dioxide	-0.113789	0.090253	-0.010996	0.126300	0.099765	0.596640	1.000000	0.112545
density	0.557213	0.000445	0.338521	0.372047	0.375573	-0.018912	0.112545	1.000000
pH	-0.596539	0.225681	-0.527095	-0.064925	-0.184304	0.087431	0.001748	-0.263005
sulphates	0.176282	-0.315552	0.277967	0.063026	-0.059988	0.025085	-0.035280	0.120295
alcohol	-0.048917	-0.214283	0.129530	0.082817	-0.260942	-0.074064	-0.234335	-0.472356
quality	0.109637	-0.356247	0.232209	0.035506	-0.162122	-0.047187	-0.168010	-0.164179

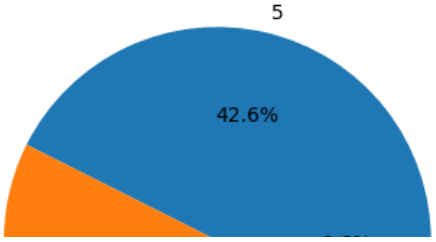
In [18]:
```
df.corr().quality.sort_values(ascending=True)
```

Out[18]:
```
volatile acidity       -0.356247
total sulfur dioxide   -0.168010
density                -0.164179
chlorides              -0.162122
pH                     -0.075837
free sulfur dioxide    -0.047187
residual sugar          0.035506
fixed acidity           0.109637
citric acid             0.232209
sulphates               0.385729
alcohol                 0.464152
quality                 1.000000
Name: quality, dtype: float64
```

Visualizations

Univariate

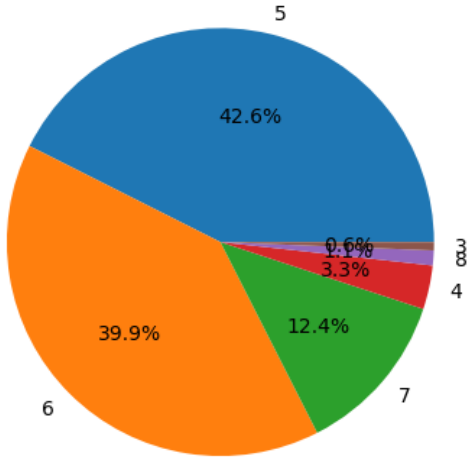
In [19]:
```
counts = df['quality'].value_counts()
plt.figure()
plt.pie(counts, labels=counts.index, autopct='%1.1f%%')
plt.show()
```



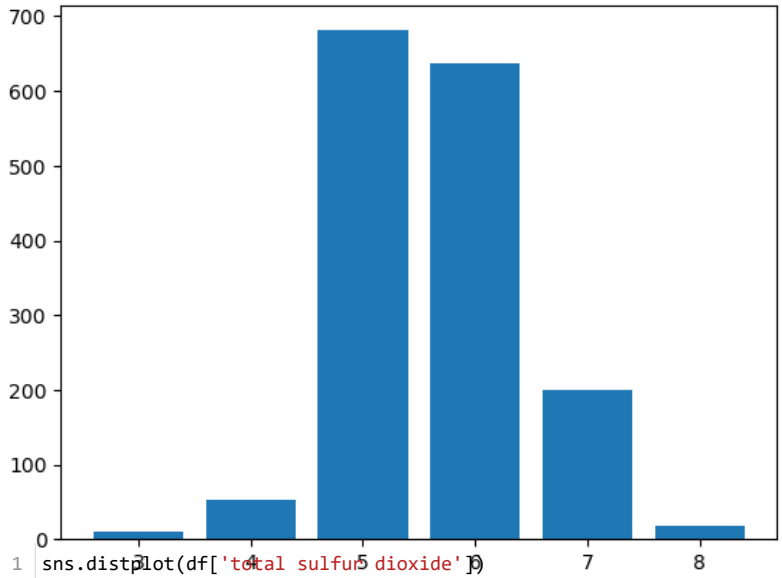
Visualizations

Univariate

In [19]:
```
counts = df['quality'].value_counts()
plt.figure()
plt.pie(counts, labels=counts.index, autopct='%1.1f%%')
plt.show()
```



In [20]:
```
plt.bar(counts.index, counts)
plt.show()
```



In [21]:
```
sns.distplot(df['total sulfur dioxide'])
```

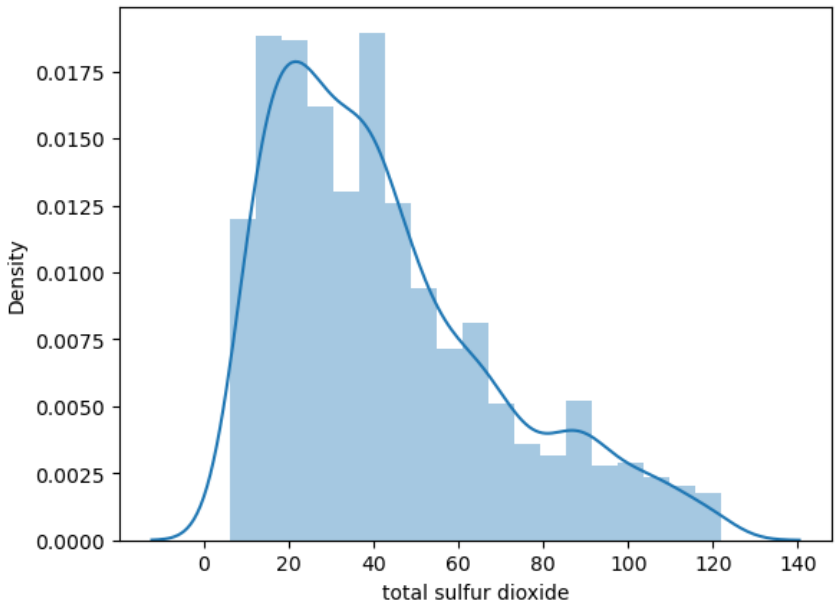
```
C:\Users\Charvi Upreti\AppData\Local\Temp\ipykernel_10484\2700524578.py:1: U
serWarning:

`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 (https://gi
st.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(df['total sulfur dioxide'])
```

Out[21]: <Axes: xlabel='total sulfur dioxide', ylabel='Density'>

In [21]:
```
1 sns.distplot(df['total sulfur dioxide'])
```

```
C:\Users\Charvi Upreti\AppData\Local\Temp\ipykernel_10484\2700524578.py:1: U
serWarning:

`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 (https://gi
st.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(df['total sulfur dioxide'])
```

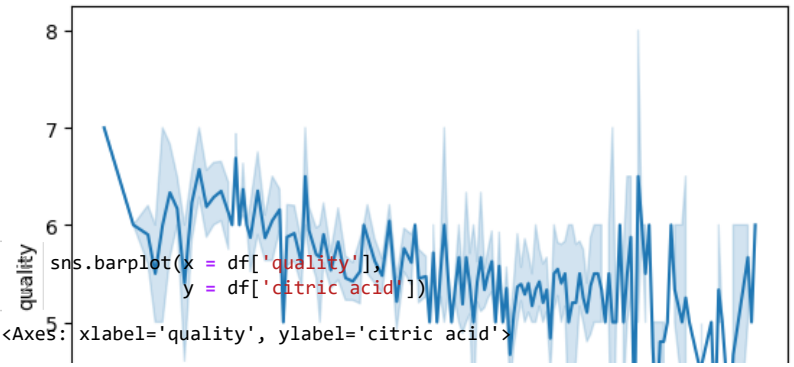
Out[21]: <Axes: xlabel='total sulfur dioxide', ylabel='Density'>



Bivariate

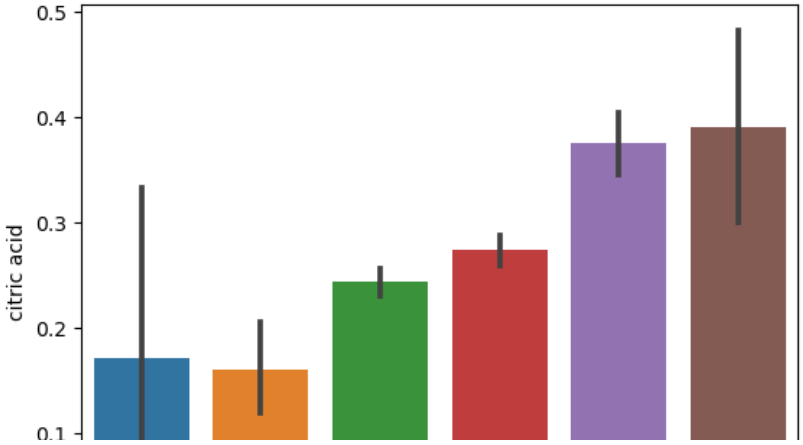
In [22]:
```
1 sns.lineplot(x = df['volatile acidity'],y=df['quality'])
```

Out[22]: <Axes: xlabel='volatile acidity', ylabel='quality'>



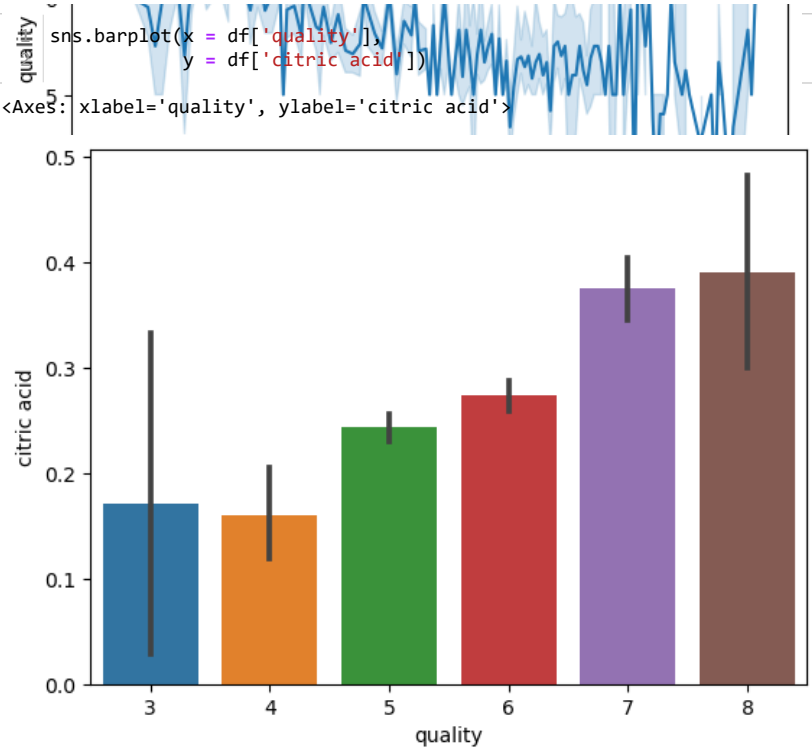
In [23]:
```
sns.barplot(x = df['quality'],
            y = df['citric acid'])
```

Out[23]: <Axes: xlabel='quality', ylabel='citric acid'>



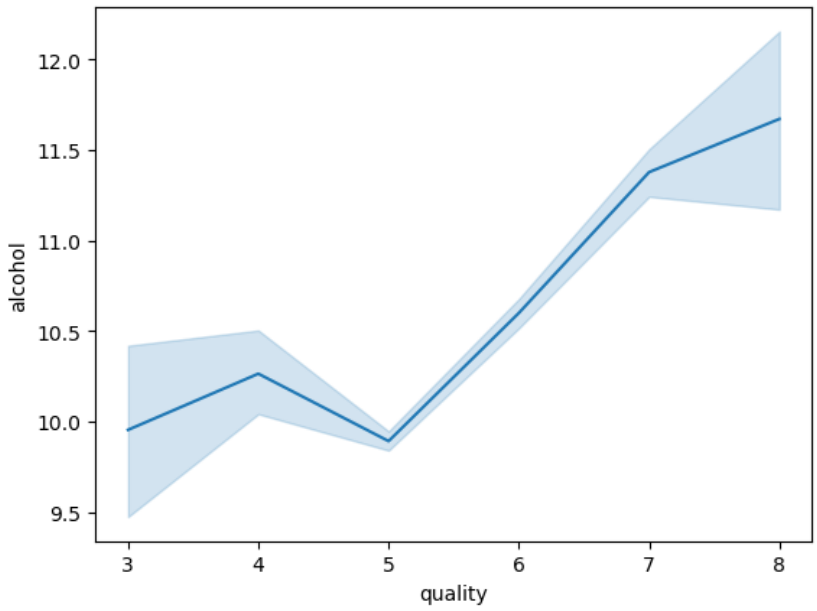
In [23]:
```
sns.barplot(x = df['quality'],
            y = df['citric acid'])
```

Out[23]: <Axes: xlabel='quality', ylabel='citric acid'>



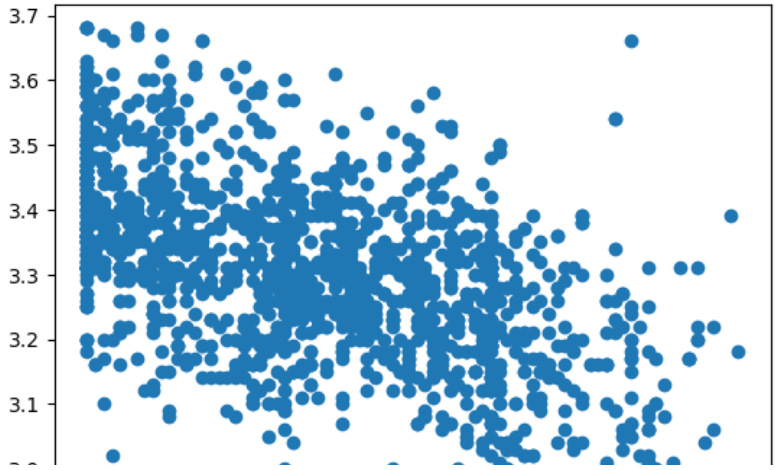
In [24]:
```
sns.lineplot(x=df['quality'],y=df['alcohol'])
```

Out[24]: <Axes: xlabel='quality', ylabel='alcohol'>



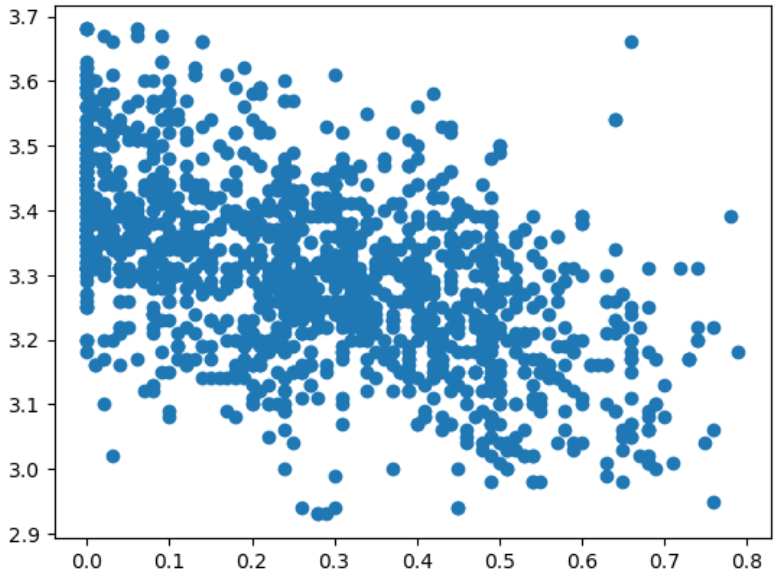
In [25]:
```
plt.scatter(df['citric acid'],df['pH'])
```

Out[25]: <matplotlib.collections.PathCollection at 0x17dfdef5bb0>



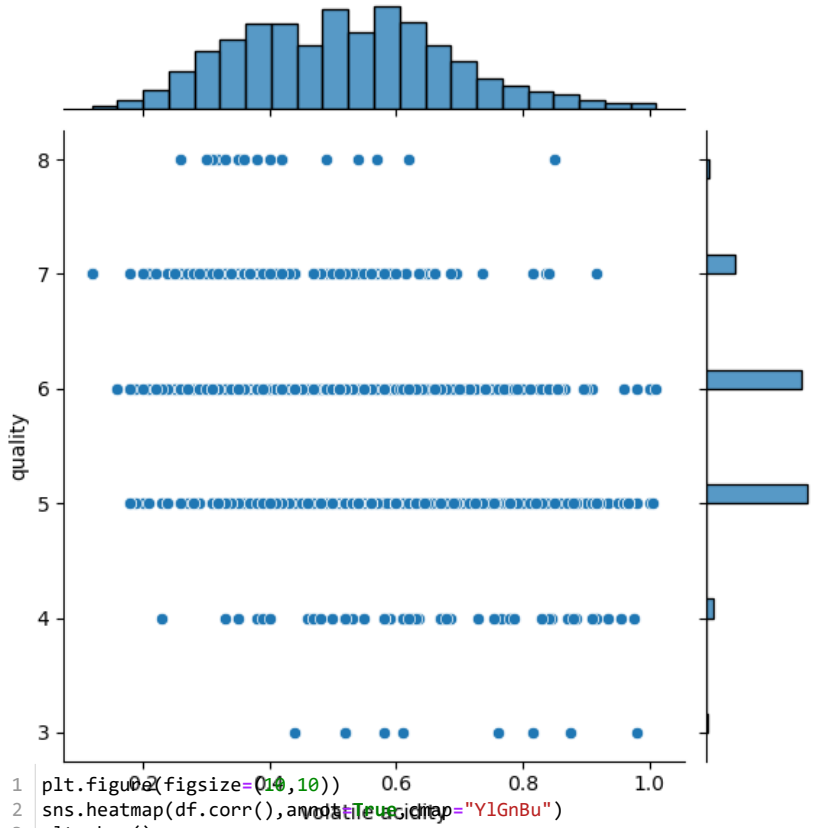
In [25]:
```
1 plt.scatter(df['citric acid'],df['pH'])
```

Out[25]: <matplotlib.collections.PathCollection at 0x17dfdef5bb0>

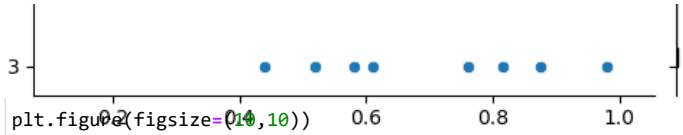


In [26]:
```
1 sns.jointplot(x = df['volatile acidity'],y=df['quality'])
```

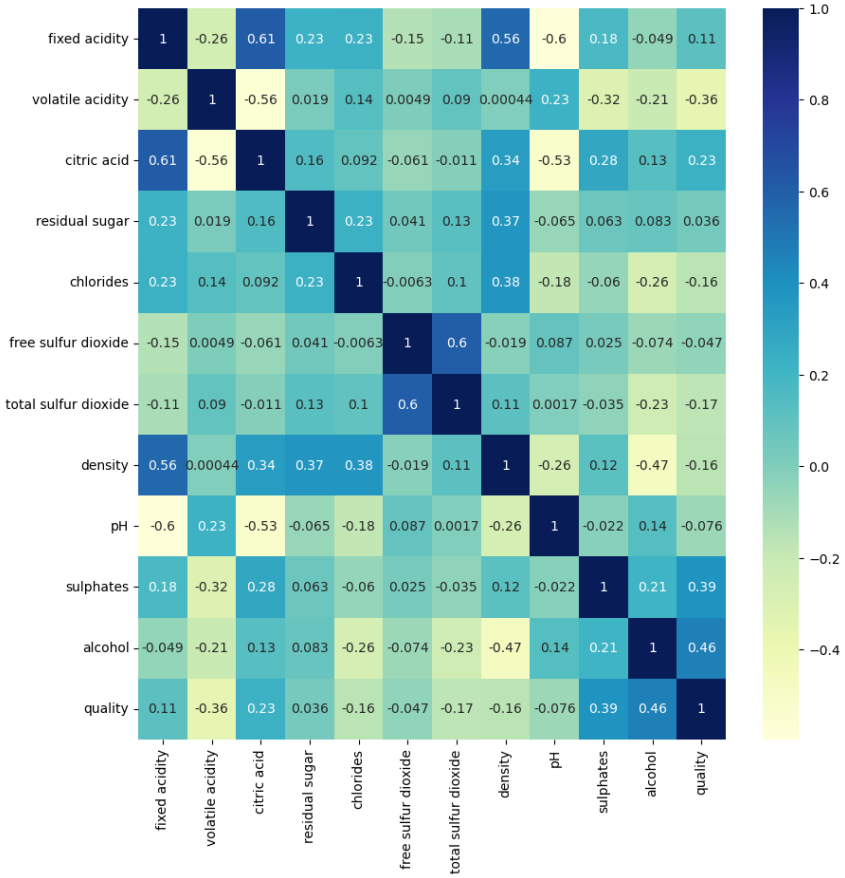
Out[26]: <seaborn.axisgrid.JointGrid at 0x17d81b56d30>



In [27]:
```
1 plt.figure(figsize=(10,10))
2 sns.heatmap(df.corr(),annot=True,cmap="YlGnBu")
3 plt.show()
```

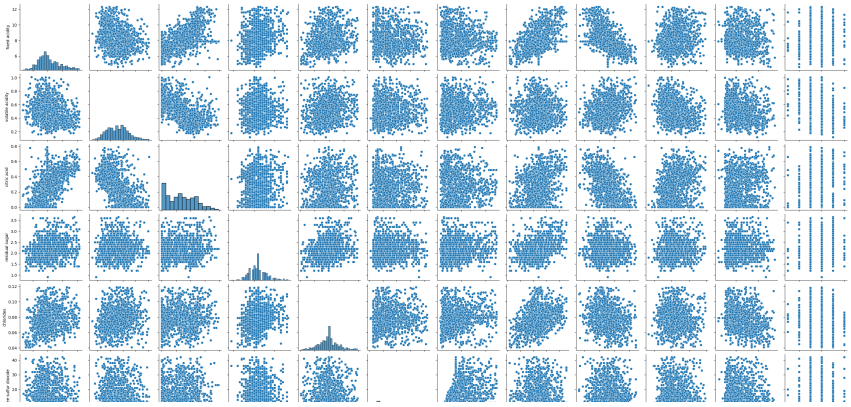


In [27]:
```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(),annot=True,cmap="YlGnBu")
plt.show()
```



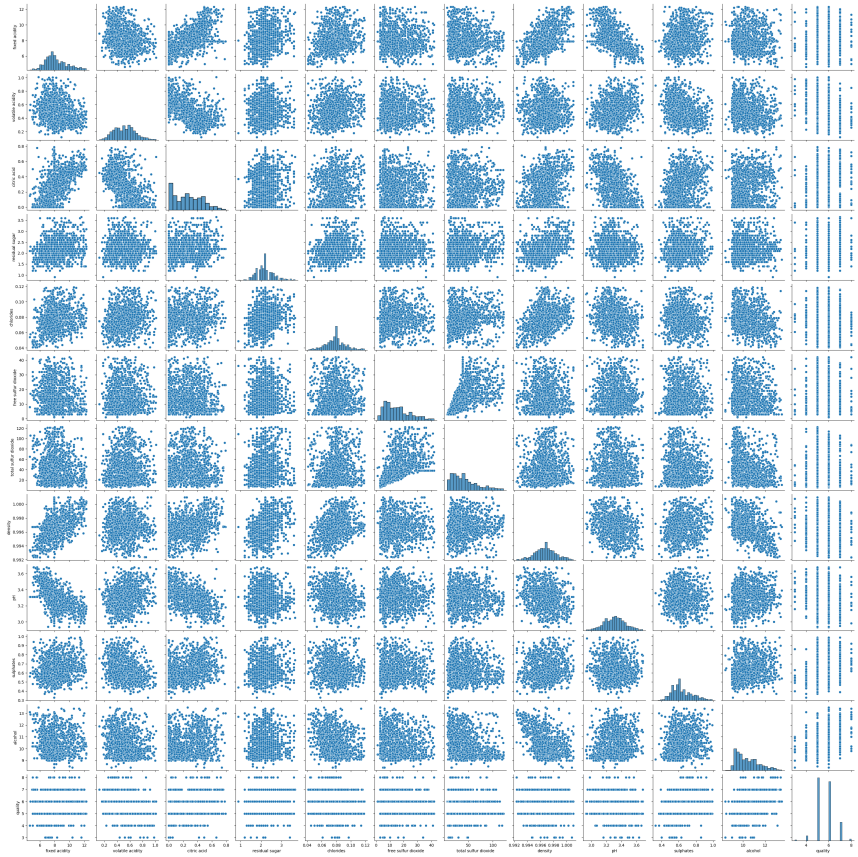
In [28]:
```
sns.pairplot(df)
plt.show()
```

```
C:\Users\Charvi Upreti\anaconda3\lib\site-packages\seaborn\axisgrid.py:118:
UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



In [28]:
```
sns.pairplot(df)
plt.show()
```

```
C:\Users\Charvi Upreti\anaconda3\lib\site-packages\seaborn\axisgrid.py:118:
UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



Spliting

In [29]:
```
#x and y split
x=df.drop(columns=['quality'],axis=1)
print(x.head().T)
y=df['quality']
y.head()
```

	0	1	2	3	4
fixed acidity	7.4000	7.8000	7.800	11.200	7.4000
volatile acidity	0.7000	0.8800	0.760	0.280	0.7000
citric acid	0.0000	0.0000	0.040	0.560	0.0000
residual sugar	1.9000	2.6000	2.300	1.900	1.9000
chlorides	0.0760	0.0980	0.092	0.075	0.0760
free sulfur dioxide	11.0000	25.0000	15.000	17.000	11.0000
total sulfur dioxide	34.0000	67.0000	54.000	60.000	34.0000
density	0.9978	0.9968	0.997	0.998	0.9978
pH	3.5100	3.2000	3.260	3.160	3.5100
sulphates	0.5600	0.6800	0.650	0.580	0.5600
alcohol	9.4000	9.8000	9.800	9.800	9.4000

Out[30]:
```
0    5
1    5
2    5
3    6
4    5
Name: quality, dtype: int64
```

Scaling

In [30]:
```
scale=MinMaxScaler()
x_scaled=pd.DataFrame(scale.fit_transform(x),columns=x.columns)
x_scaled.head().T
```

	0	1	2	3	4
fixed acidity	0.363636	0.415584	0.415584	0.857143	0.363636
volatile acidity	0.651685	0.853933	0.719101	0.179775	0.651685
citric acid	0.000000	0.000000	0.050633	0.708861	0.000000
residual sugar	0.363636	0.618182	0.509091	0.363636	0.363636
chlorides	0.448718	0.730769	0.653846	0.435897	0.448718
free sulfur dioxide	0.243902	0.585366	0.341463	0.390244	0.243902
total sulfur dioxide	0.241379	0.525862	0.413793	0.465517	0.241379
density	0.630058	0.514451	0.537572	0.653179	0.630058
pH	0.773333	0.360000	0.440000	0.306667	0.773333

pH	3.5100	3.2000	3.260	3.160	3.5100
sulphates	0.5600	0.6800	0.650	0.580	0.5600
alcohol	9.4000	9.8000	9.800	9.800	9.4000

Scaling

In [30]:
```
scale=MinMaxScaler()
x_scaled=pd.DataFrame(scale.fit_transform(x),columns=x.columns)
x_scaled.head().T
```

Out[30]:

	0	1	2	3	4
fixed acidity	0.363636	0.415584	0.415584	0.857143	0.363636
volatile acidity	0.651685	0.853933	0.719101	0.179775	0.651685
citric acid	0.000000	0.000000	0.050633	0.708861	0.000000
residual sugar	0.363636	0.618182	0.509091	0.363636	0.363636
chlorides	0.448718	0.730769	0.653846	0.435897	0.448718
free sulfur dioxide	0.243902	0.585366	0.341463	0.390244	0.243902
total sulfur dioxide	0.241379	0.525862	0.413793	0.465517	0.241379
density	0.630058	0.514451	0.537572	0.653179	0.630058
pH	0.773333	0.360000	0.440000	0.306667	0.773333
sulphates	0.348485	0.530303	0.484848	0.378788	0.348485
alcohol	0.196078	0.274510	0.274510	0.274510	0.196078

In [31]:
```
x_train,x_test,y_train,y_test=train_test_split(x_scaled,y,test_size=0.2,r
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(1279, 11)
(1279,)
(320, 11)
(320,)
```

Machine Learning Model Building, Evaluation of the model and Test with random observation

Linear Regression

In [32]:
```
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[32]:
```
LinearRegression
LinearRegression()
```

In [33]:
```
#testing
print("Test result")
y_predict=lr.predict(x_test)
#R2 Score
print("R2 score",metrics.r2_score(y_test,y_predict))
#MSE (Mean Square Error)
print("MSE",metrics.mean_squared_error(y_test,y_predict))
#RMSE (Root Mean Square Error)
print("RMSE",np.sqrt(metrics.mean_squared_error(y_test,y_predict)))

print('\n')

#to check if overfitting
y_predict1=lr.predict(x_train)
#printing to check if overfitting or underfitting
#R2 Score
print("R2 Score of training",metrics.r2_score(y_train,y_predict1))
print("R2 score of training and testing is not much different.")

print('\n')

#Checking a random observation:-
predicted_values = lr.predict(x_test.iloc[[0]])
```

In [33]:

```
#testing
print("Test result")
y_predict=lr.predict(x_test)
#R2 Score
print("R2 score",metrics.r2_score(y_test,y_predict))
#MSE (Mean Square Error)
print("MSE",metrics.mean_squared_error(y_test,y_predict))
#RMSE (Root Mean Square Error)
print("RMSE",np.sqrt(metrics.mean_squared_error(y_test,y_predict)))

print('\n')

#to check if overfitting
y_predict1=lr.predict(x_train)
#printing to check if overfitting or underfitting
#R2 Score
print("R2 Score of training",metrics.r2_score(y_train,y_predict1))
print("R2 score of training and testing is not much different.")

print('\n')

#Checking a random observation:-
predicted_values = lr.predict(x_test.iloc[[0]])
actual_values = y_test.iloc[[0]]
print("Predicted " + str(predicted_values))
print("Actual " + str(actual_values))

column_names = x_test.columns

# Input data for a single observation
input_data = [0.6, 0.5, 0.43, 0.5, 0.5, 0.4, 0.3, 0.3, 0.8, 0.6, 0.4]
input_df = pd.DataFrame([input_data], columns=column_names)

# Make a prediction for the single observation
prediction_single_observation = lr.predict(input_df)

print("Prediction for the single observation:", prediction_single_observa

input_df
```

```
Test result
R2 score 0.3318798167106596
MSE 0.3824727064884877
RMSE 0.6184437779527641


R2 Score of training 0.3515107836557766
R2 score of training and testing is not much different.


Predicted [6.05272976]
Actual 1109    6
Name: quality, dtype: int64
Prediction for the single observation: [5.63253129]
```

Out[33]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	0.6	0.5	0.43	0.5	0.5	0.4	0.3	0.3	0.8	0.6	0.4

In [37]:

```
r_pred=r.predict(x_test)
r_pred_1=r.predict(x_train)
l_pred=l.predict(x_test)
l_pred_1=l.predict(x_train)
```

Lasso and Ridge

In [34]:

```
r=Ridge()
```

In [38]:

```
Quality=pd.DataFrame({'Actual_Profit':y_test, 'Ridge_pred':r_pred,'Lasso_
Quality.head(10)
```

In [35]:

```
r.fit(x_train,y_train)
```

Out[38]:

	Actual_Profit	Ridge_pred	Lasso_pred
1109	6	6.043328	5.646599
1032	5	4.950493	5.646599
1002	7	6.681428	5.646599
487	6	5.156834	5.646599
979	5	5.827819	5.646599
1054	6	5.187421	5.646599
542	5	5.426785	5.646599
853	6	5.955455	5.646599

Out[35]:

```
Ridge()
```

In [36]:

```
l.fit(x_train,y_train)
```

Out[36]:

```
Lasso()
```

In [37]:
```
r_pred=r.predict(x_test)
r_pred_1=r.predict(x_train)
l_pred=l.predict(x_test)
l_pred_1=l.predict(x_train)
```

Lasso and Ridge

In [34]:
```
r=Ridge()
l=Lasso()
```

In [38]:
```
Quality=pd.DataFrame({'Actual_Profit':y_test, 'Ridge_pred':r_pred,'Lasso_
Quality.head(10)
```

In [35]:
```
r.fit(x_train,y_train)
```

Out[38]:

	Actual_Profit	Ridge_pred	Lasso_pred
1109	6	6.043328	5.646599
1032	5	4.950493	5.646599
1002	7	6.681428	5.646599
487	6	5.156834	5.646599
979	5	5.827819	5.646599
1054	6	5.187421	5.646599
542	5	5.426785	5.646599
853	6	5.955455	5.646599
1189	4	4.752772	5.646599
412	5	5.121926	5.646599

Out[35]:
```
Ridge()
```

In [36]:
```
l.fit(x_train,y_train)
```

Out[36]:
```
Lasso()
```

In [39]:
```
print("For ridge\n")

#R2 Score
print("R2 score",metrics.r2_score(y_test,r_pred))
#printing to check if overfitting or underfitting
#R2 Score
print("R2 score for training",metrics.r2_score(y_train,r_pred_1))
#MSE (Mean Square Error)
print("MSE",metrics.mean_squared_error(y_test,r_pred))
#RMSE (Root Mean Square Error)
print("RMSE",np.sqrt(metrics.mean_squared_error(y_test,r_pred)))

print('\n')
#Checking a random observation:-
predicted_values = r.predict(x_test.iloc[[0]])
actual_values = y_test.iloc[[0]]
print("Predicted " + str(predicted_values))
print("Actual " + str(actual_values))

column_names = x_test.columns

# Input data for a single observation
input_data = [0.6, 0.5, 0.43, 0.5, 0.5, 0.4, 0.3, 0.3, 0.8, 0.6, 0.4]
```

In [39]:

```
print("For ridge\n")

#R2 Score
print("R2 score",metrics.r2_score(y_test,r_pred))
#printing to check if overfitting or underfitting
#R2 Score
print("R2 score for training",metrics.r2_score(y_train,r_pred_1))
#MSE (Mean Square Error)
print("MSE",metrics.mean_squared_error(y_test,r_pred))
#RMSE (Root Mean Square Error)
print("RMSE",np.sqrt(metrics.mean_squared_error(y_test,r_pred)))

print('\n')
#Checking a random observation:-
predicted_values = r.predict(x_test.iloc[[0]])
actual_values = y_test.iloc[[0]]
print("Predicted " + str(predicted_values))
print("Actual " + str(actual_values))

column_names = x_test.columns

# Input data for a single observation
input_data = [0.6, 0.5, 0.43, 0.5, 0.5, 0.4, 0.3, 0.3, 0.8, 0.6, 0.4]
input_df = pd.DataFrame([input_data], columns=column_names)

# Make a prediction for the single observation
prediction_single_observation = r.predict(input_df)

print("Prediction for the single observation:", prediction_single_observa



print('\n')
print("For Lasso")

#R2 Score
print("R2 score",metrics.r2_score(y_test,l_pred))
#printing to check if overfitting or underfitting
#R2 Score
print("R2 score with training",metrics.r2_score(y_train,l_pred_1))
#MSE (Mean Square Error)
print("MSE",metrics.mean_squared_error(y_test,l_pred))
#RMSE (Root Mean Square Error)
print("RMSE",np.sqrt(metrics.mean_squared_error(y_test,l_pred)))

print('\n')

#Checking a random observation:-
predicted_values = l.predict(x_test.iloc[[0]])
actual_values = y_test.iloc[[0]]
print("Predicted " + str(predicted_values))
print("Actual " + str(actual_values))

column_names = x_test.columns

# Input data for a single observation
input_data = [0.6, 0.5, 0.43, 0.5, 0.5, 0.4, 0.3, 0.3, 0.8, 0.6, 0.4]
input_df = pd.DataFrame([input_data], columns=column_names)

# Make a prediction for the single observation
prediction_single_observation = l.predict(input_df)

print("Prediction for the single observation:", prediction_single_observa
```

```
For ridge
R2 score 0.33460003973813257
R2 score for training 0.35135292178989475
MSE 0.3809154850639714
RMSE 0.6171835100389279


Predicted [6.04332824]
Actual 1109    6
Name: quality, dtype: int64
Prediction for the single observation: [5.65709029]


For Lasso
R2 score -0.004878947397931155
R2 score with training 0.0
MSE 0.5752539443014328
RMSE 0.7584549718351333


Predicted [5.64659891]
Actual 1109    6
```

```
For ridge
print("Prediction for the single observation:", prediction_single_observa
R2 score 0.33460003973813257
input_df
R2 score for training 0.35135292178989475
MSE 0.3809154850639714
RMSE 0.6171835100389279


Predicted [6.04332824]
Actual 1109    6
Name: quality, dtype: int64
Prediction for the single observation: [5.65709029]


For Lasso
R2 score -0.004878947397931155
R2 score with training 0.0
MSE 0.5752539443014328
RMSE 0.7584549718351333


Predicted [5.64659891]
Actual 1109    6
Name: quality, dtype: int64
Prediction for the single observation: [5.64659891]
```

Out[39]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	0.6	0.5	0.43	0.5	0.5	0.4	0.3	0.3	0.8	0.6	0.4

In [40]:

```
#Linear Regression and Ridge Regression both have similar R2 scores
#and perform better than Lasso Regression in terms of R2.
```

Logistic Regression

In [41]:

```
model=LogisticRegression()
model.fit(x_train,y_train)
```

```
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\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 (https://scik
it-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-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
```

Out[41]:

```
▾ LogisticRegression

LogisticRegression()
```

In [42]:

```
pred_log=model.predict(x_test)
pred_log_1=model.predict(x_train)

#Checking a random obseration:-
predicted_values = model.predict(x_test.iloc[[0]])
actual_values = y_test.iloc[[0]]

print("Predicted " + str(predicted_values))
print("Actual " + str(actual_values))

column_names = x_test.columns

# Input data for a single observation
input_data = [0.6, 0.5, 0.43, 0.5, 0.5, 0.4, 0.3, 0.3, 0.8, 0.6, 0.4]
input_df = pd.DataFrame([input_data], columns=column_names)

# Make a prediction for the single observation
prediction_single_observation = model.predict(input_df)

print("Prediction for the single observation:", prediction_single_observa

input_df
```

In [42]:

```
pred_log=model.predict(x_test)
pred_log_1=model.predict(x_train)

#Checking a random obseration:-
predicted_values = model.predict(x_test.iloc[[0]])
actual_values = y_test.iloc[[0]]

print("Predicted " + str(predicted_values))
print("Actual " + str(actual_values))

column_names = x_test.columns

# Input data for a single observation
input_data = [0.6, 0.5, 0.43, 0.5, 0.5, 0.4, 0.3, 0.3, 0.8, 0.6, 0.4]
input_df = pd.DataFrame([input_data], columns=column_names)

# Make a prediction for the single observation
prediction_single_observation = model.predict(input_df)

print("Prediction for the single observation:", prediction_single_observa

input_df
```

```
Predicted [6]
Actual 1109    6
Name: quality, dtype: int64
Prediction for the single observation: [6]
```

Out[42]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	0.6	0.5	0.43	0.5	0.5	0.4	0.3	0.3	0.8	0.6	0.4

In [43]:

```
print("Accurracy score: ",accuracy_score(y_test,pred_log))
print("Accurracy score with training: ",accuracy_score(y_train,pred_log_1
print("Confusion matrix")
print(confusion_matrix(y_test,pred_log))
print("Crosstab")
pd.crosstab(y_test,pred_log)
```

```
Accurracy score:  0.621875
Accurracy score with training:  0.5957779515246286
Confusion matrix
[[  0   0   2   0   0   0]
 [  0   0   7   4   0   0]
 [  0   0 105  29   1   0]
 [  0   0  46  85  11   0]
 [  0   0   2  16   9   0]
 [  0   0   0   1   2   0]]
Crosstab
```

Out[43]:

col_0	5	6	7
quality			
3	2	0	0
4	7	4	0
5	105	29	1
6	46	85	11
7	2	16	9
8	0	1	2

In [44]:

```
print(classification_report(y_test,pred_log))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00        11
           5       0.65      0.78      0.71       135
           6       0.63      0.60      0.61       142
           7       0.39      0.33      0.36        27
           8       0.00      0.00      0.00         3

    accuracy                           0.62       320
   macro avg       0.28      0.28      0.28       320
weighted avg       0.59      0.62      0.60       320


C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
```

5	105	29	1
6	46	85	11
7	2	16	9
8	0	1	2

In [44]:
```
print(classification_report(y_test,pred_log))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00        11
           5       0.65      0.78      0.71       135
           6       0.63      0.60      0.61       142
           7       0.39      0.33      0.36        27
           8       0.00      0.00      0.00         3

    accuracy                           0.62       320
   macro avg       0.28      0.28      0.28       320
weighted avg       0.59      0.62      0.60       320


C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [45]:
```
print("Here the columns 3,4,8 were lost, this is because there samples we
```

```
Here the columns 3,4,8 were lost, this is because there samples were less.
```

Decision Tree

In [46]:
```
model1=DecisionTreeClassifier()
model1.fit(x_train,y_train)
```

Out[46]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [47]:
```
d_y_predict = model1.predict(x_test)
d_y_predict_train = model1.predict(x_train)

#Checking a random obseration:-
predicted_values = model1.predict(x_test.iloc[[0]])
actual_values = y_test.iloc[[0]]

print("Predicted " + str(predicted_values))
print("Actual " + str(actual_values))

column_names = x_test.columns

# Input data for a single observation
input_data = [0.6, 0.5, 0.43, 0.5, 0.5, 0.4, 0.3, 0.3, 0.8, 0.6, 0.4]
input_df = pd.DataFrame([input_data], columns=column_names)

# Make a prediction for the single observation
prediction_single_observation = model1.predict(input_df)

print("Prediction for the single observation:", prediction_single_observa

input_df
```

In [48]:
```
print("Accurracy score: ",accuracy_score(y_test,d_y_predict ))
print("Accurracy score with training: ",accuracy_score(y_train,d_y_predic
print("Confusion matrix")
print(confusion_matrix(y_test,d_y_predict))
print("Crosstab")
print(classification_report(y_test,d_y_predict))
pd.crosstab(y_test,d_y_predict)
```

```
Predicted [6]
Actual 1109    6
Name: quality, dtype: int64
Prediction for the single observation: [7]
```

Out[47]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	0.6	0.5	0.43	0.5	0.5	0.4	0.3	0.3	0.8	0.6	0.4

```
Accurracy score:  0.621875
Accurracy score with training:  1.0
Confusion matrix
[[ 0  0  1  1  0  0]
 [ 0  2  4  5  0  0]
 [ 0  6 89 33  6  1]
 [ 0  3 25 91 23  0]
 [ 0  0  1  7 17  2]
 [ 0  0  0  1  2  0]]
Crosstab
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.18      0.18      0.18        11
           5       0.74      0.66      0.70       135
```

```
input_data = [0.6, 0.5, 0.43, 0.5, 0.5, 0.4, 0.3, 0.3, 0.8, 0.6, 0.4]
input_df = pd.DataFrame([input_data], columns=column_names)

# Make a prediction for the single observation
prediction_single_observation = model1.predict(input_df)

print("Prediction for the single observation:", prediction_single_observation)

input_df
```

```
Predicted [6]
Actual 1109    6
Name: quality, dtype: int64
Prediction for the single observation: [7]
```

Out[47]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	0.6	0.5	0.43	0.5	0.5	0.4	0.3	0.3	0.8	0.6	0.4

In [48]:

```
print("Accurracy score: ",accuracy_score(y_test,d_y_predict ))
print("Accurracy score with training: ",accuracy_score(y_train,d_y_predic
print("Confusion matrix")
print(confusion_matrix(y_test,d_y_predict))
print("Crosstab")
print(classification_report(y_test,d_y_predict))
pd.crosstab(y_test,d_y_predict)
```

```
Accurracy score:  0.621875
Accurracy score with training:  1.0
Confusion matrix
[[ 0  0  1  1  0  0]
 [ 0  2  4  5  0  0]
 [ 0  6 89 33  6  1]
 [ 0  3 25 91 23  0]
 [ 0  0  1  7 17  2]
 [ 0  0  0  1  2  0]]
Crosstab
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.18      0.18      0.18        11
           5       0.74      0.66      0.70       135
           6       0.66      0.64      0.65       142
           7       0.35      0.63      0.45        27
           8       0.00      0.00      0.00         3

    accuracy                           0.62       320
   macro avg       0.32      0.35      0.33       320
weighted avg       0.64      0.62      0.63       320


C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[48]:

col_0	4	5	6	7	8
quality					
3	0	1	1	0	0
4	2	4	5	0	0
5	6	89	33	6	1
6	3	25	91	23	0
7	0	1	7	17	2
8	0	0	1	2	0

In [49]:

```
print(classification_report(y_test,d_y_predict))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.18      0.18      0.18        11
           5       0.74      0.66      0.70       135
           6       0.66      0.64      0.65       142
           7       0.35      0.63      0.45        27
           8       0.00      0.00      0.00         3

    accuracy                           0.62       320
   macro avg       0.32      0.35      0.33       320
weighted avg       0.64      0.62      0.63       320


C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
```

In [49]:
```
print(classification_report(y_test,d_y_predict))
```

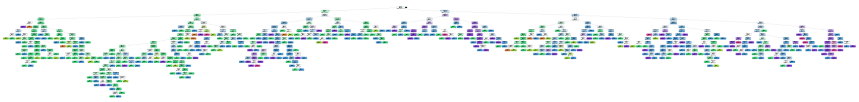
```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.18      0.18      0.18        11
           5       0.74      0.66      0.70       135
           6       0.66      0.64      0.65       142
           7       0.35      0.63      0.45        27
           8       0.00      0.00      0.00         3

    accuracy                           0.62       320
   macro avg       0.32      0.35      0.33       320
weighted avg       0.64      0.62      0.63       320


C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [53]:
```
print("The model seems to be overfit.")
print("Unlike logistic regression, No quality was lost in predictions.")
```

```
The model seems to be overfit.
Unlike logistic regression, No quality was lost in predictions.
```

In [51]:
```
dot_data = StringIO()
export_graphviz(model1, out_file=dot_data, feature_names=x.columns, fille
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.set_dpi(5)
Image(graph.create_png())

```

Out[51]:



Random Forest Classifier

In [52]:
```
model2=RandomForestClassifier()
model2.fit(x_train,y_train)
```

Out[52]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [54]:
```
r_y_predict=model2.predict(x_test)
r_y_predict_train=model2.predict(x_train)

#Checking a random obseration:-
predicted_values = model2.predict(x_test.iloc[[0]])
actual_values = y_test.iloc[[0]]

print("Predicted " + str(predicted_values))
print("Actual " + str(actual_values))

column_names = x_test.columns

# Input data for a single observation
input_data = [0.6, 0.5, 0.43, 0.5, 0.5, 0.4, 0.3, 0.3, 0.8, 0.6, 0.4]
input_df = pd.DataFrame([input_data], columns=column_names)

# Make a prediction for the single observation
prediction_single_observation = model2.predict(input_df)

print("Prediction for the single observation:", prediction_single_observa

input_df
```

In [54]:

```
r_y_predict=model2.predict(x_test)
r_y_predict_train=model2.predict(x_train)

#Checking a random observation:-
predicted_values = model2.predict(x_test.iloc[[0]])
actual_values = y_test.iloc[[0]]

print("Predicted " + str(predicted_values))
print("Actual " + str(actual_values))

column_names = x_test.columns

# Input data for a single observation
input_data = [0.6, 0.5, 0.43, 0.5, 0.5, 0.4, 0.3, 0.3, 0.8, 0.6, 0.4]
input_df = pd.DataFrame([input_data], columns=column_names)

# Make a prediction for the single observation
prediction_single_observation = model2.predict(input_df)

print("Prediction for the single observation:", prediction_single_observa

input_df
```

```
Predicted [6]
Actual 1109    6
Name: quality, dtype: int64
Prediction for the single observation: [6]
```

Out[54]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	0.6	0.5	0.43	0.5	0.5	0.4	0.3	0.3	0.8	0.6	0.4

In [55]:

```
print("Accurracy score: ",accuracy_score(y_test,r_y_predict ))
print("Accurracy score with training: ",accuracy_score(y_train,r_y_predic
print("Confusion matrix")
print(confusion_matrix(y_test,r_y_predict))
print("Crosstab")
print(classification_report(y_test,r_y_predict))
pd.crosstab(y_test,r_y_predict)
```

```
Accurracy score:  0.728125
Accurracy score with training:  1.0
Confusion matrix
[[  0   0   0   2   0   0]
 [  0   0   6   5   0   0]
 [  0   0 112  21   2   0]
 [  0   0  25 106  11   0]
 [  0   0   1  11  15   0]
 [  0   0   0   1   2   0]]
Crosstab
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00        11
           5       0.78      0.83      0.80       135
```

In [55]:

```
print("Accurracy score: ",accuracy_score(y_test,r_y_predict ))
print("Accurracy score with training: ",accuracy_score(y_train,r_y_predic
print("Confusion matrix")
print(confusion_matrix(y_test,r_y_predict))
print("Crosstab")
print(classification_report(y_test,r_y_predict))
pd.crosstab(y_test,r_y_predict)
```

```
Accurracy score:  0.728125
Accurracy score with training:  1.0
Confusion matrix
[[  0   0   0   2   0   0]
 [  0   0   6   5   0   0]
 [  0   0 112  21   2   0]
 [  0   0  25 106  11   0]
 [  0   0   1  11  15   0]
 [  0   0   0   1   2   0]]
Crosstab
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00        11
           5       0.78      0.83      0.80       135
           6       0.73      0.75      0.74       142
           7       0.50      0.56      0.53        27
           8       0.00      0.00      0.00         3

    accuracy                           0.73       320
   macro avg       0.33      0.36      0.34       320
weighted avg       0.69      0.73      0.71       320


C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[55]:

col_0	5	6	7
quality			
3	0	2	0
4	6	5	0
5	112	21	2
6	25	106	11
7	1	11	15
8	0	1	2

In [56]:

```
print(classification_report(y_test,r_y_predict))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00        11
           5       0.78      0.83      0.80       135
           6       0.73      0.75      0.74       142
           7       0.50      0.56      0.53        27
           8       0.00      0.00      0.00         3

    accuracy                           0.73       320
   macro avg       0.33      0.36      0.34       320
weighted avg       0.69      0.73      0.71       320


C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
```

In [56]:
```python
print(classification_report(y_test,r_y_predict))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00        11
           5       0.78      0.83      0.80       135
           6       0.73      0.75      0.74       142
           7       0.50      0.56      0.53        27
           8       0.00      0.00      0.00         3

    accuracy                           0.73       320
   macro avg       0.33      0.36      0.34       320
weighted avg       0.69      0.73      0.71       320


C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [57]:
```python
#Random forest based on entropy
model3 = RandomForestClassifier(criterion='entropy')
model3.fit(x_train, y_train)
```

Out[57]:
```
RandomForestClassifier
RandomForestClassifier(criterion='entropy')
```

In [58]:
```python
r_y_predict_1 = model3.predict(x_test)
r_y_predict_train_1 = model3.predict(x_train)

#Checking a random obseration:-
predicted_values = model3.predict(x_test.iloc[[0]])
actual_values = y_test.iloc[[0]]

print("Predicted " + str(predicted_values))
print("Actual " + str(actual_values))

column_names = x_test.columns

# Input data for a single observation
input_data = [0.6, 0.5, 0.43, 0.5, 0.5, 0.4, 0.3, 0.3, 0.8, 0.6, 0.4]
input_df = pd.DataFrame([input_data], columns=column_names)

# Make a prediction for the single observation
prediction_single_observation = model3.predict(input_df)

print("Prediction for the single observation:", prediction_single_observa
input_df
```

```
Predicted [6]
Actual 1109    6
Name: quality, dtype: int64
Prediction for the single observation: [6]
```

Out[58]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	0.6	0.5	0.43	0.5	0.5	0.4	0.3	0.3	0.8	0.6	0.4

In [59]:
```python
print("Accurracy score: ",accuracy_score(y_test,r_y_predict_1))
print("Accurracy score with training: ",accuracy_score(y_train,r_y_predic
print("Confusion matrix")
print(confusion_matrix(y_test,r_y_predict_1))
print("Crosstab")
print(classification_report(y_test,r_y_predict_1))
pd.crosstab(y_test,r_y_predict_1)
```

```
Accurracy score:  0.734375
Accurracy score with training: [illegible]
Confusion matrix
[[  0   0   1   1   0   0]
 [  0   1   7   3   0   0]
 [  0   0 112  21   2   0]
 [  0   0  27 105  10   0]
 [  0   0   1   9  17   0]
 [  0   0   0   1   2   0]]
Crosstab
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       1.00      0.09      0.17        11
           5       0.76      0.83      0.79       135
```

```
21
22 input_df
```

Predicted [6]
Actual 1109   6
Name: quality, dtype: int64
Prediction for the single observation: [6]

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	0.6	0.5	0.43	0.5	0.5	0.4	0.3	0.3	0.8	0.6	0.4

In [59]:

```
print("Accurracy score: ",accuracy_score(y_test,r_y_predict_1))
print("Accurracy score with training: ",accuracy_score(y_train,r_y_predic
print("Confusion matrix")
print(confusion_matrix(y_test,r_y_predict_1))
print("Crosstab")
print(classification_report(y_test,r_y_predict_1))
pd.crosstab(y_test,r_y_predict_1)
```

Out[58]:

```
Accurracy score:  0.734375
Accurracy score with training:  1.0
Confusion matrix
[[  0   0   1   1   0   0]
 [  0   1   7   3   0   0]
 [  0   0 112  21   2   0]
 [  0   0  27 105  10   0]
 [  0   0   1   9  17   0]
 [  0   0   0   1   2   0]]
Crosstab
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       1.00      0.09      0.17        11
           5       0.76      0.83      0.79       135
           6       0.75      0.74      0.74       142
           7       0.55      0.63      0.59        27
           8       0.00      0.00      0.00         3

    accuracy                           0.73       320
   macro avg       0.51      0.38      0.38       320
weighted avg       0.73      0.73      0.72       320


C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[59]:

col_0	4	5	6	7
quality				
3	0	1	1	0
4	1	7	3	0
5	0	112	21	2
6	0	27	105	10
7	0	1	9	17
8	0	0	1	2

In [60]:

```
print(classification_report(y_test,r_y_predict_1))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       1.00      0.09      0.17        11
           5       0.76      0.83      0.79       135
           6       0.75      0.74      0.74       142
           7       0.55      0.63      0.59        27
           8       0.00      0.00      0.00         3

    accuracy                           0.73       320
   macro avg       0.51      0.38      0.38       320
weighted avg       0.73      0.73      0.72       320


C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
```

In [60]:
```
print(classification_report(y_test,r_y_predict_1))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       1.00      0.09      0.17        11
           5       0.76      0.83      0.79       135
           6       0.75      0.74      0.74       142
           7       0.55      0.63      0.59        27
           8       0.00      0.00      0.00         3

    accuracy                           0.73       320
   macro avg       0.51      0.38      0.38       320
weighted avg       0.73      0.73      0.72       320
```

```
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Charvi Upreti\AppData\Roaming\Python\Python39\site-packages\sklearn
\metrics\_classification.py:1469: UndefinedMetricWarning: Precision and F-sc
ore are ill-defined and being set to 0.0 in labels with no predicted sample
s. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [61]:
```
print("Maximum Accuracy reached was with Random forest classifier and is
```

```
Maximum Accuracy reached was with Random forest classifier and is  0.734375
```

Trying binary classification to get a better accuracy (Good: 1 vs Bad: 0 Wine)

In [62]:
```
df['quality'].value_counts().sort_index(ascending=True)
```

Out[62]:
```
3     10
4     53
5    681
6    638
7    199
8     18
Name: quality, dtype: int64
```

In [63]:
```
y_train = y_train.replace([3, 4, 5,6], 0)
y_train = y_train.replace([7, 8], 1)
y_train
```

Out[63]:
```
642     0
679     0
473     0
390     1
1096    0
       ..
763     0
835     0
1216    0
559     0
684     0
Name: quality, Length: 1279, dtype: int64
```

In [65]:
```
y_test = y_test.replace([3, 4, 5,6], 0)
y_test = y_test.replace([7, 8], 1)
y_test
```

Out[65]:
```
1109    0
1032    0
1002    1
487     0
979     0
       ..
794     0
813     0
1322    0
704     0
1023    0
Name: quality, Length: 320, dtype: int64
```

In [64]:
```
y_train.value_counts()
```

Out[64]:
```
0    1092
1     187
Name: quality, dtype: int64
```

In [66]:
```
y_test.value_counts()
```

Out[66]:
```
0    290
1     30
Name: quality, dtype: int64
```

In [65]:
```
y_test = y_test.replace([3, 4, 5,6], 0)
y_test = y_test.replace([7, 8], 1)
y_test
```

Out[65]:
```
763     0
835     0
1216    0
559     0
684     0
1109    0
1032    0
1002    1
487     0
979     0
..
0       1092
1       189
Name: quality, dtype: int64
1322    0
704     0
1023    0
Name: quality, Length: 320, dtype: int64
```

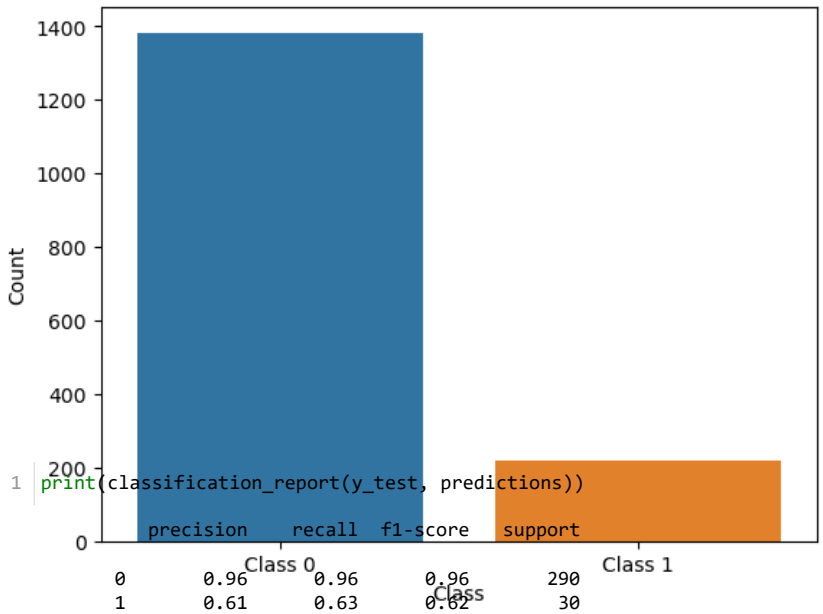
In [64]:
```
y_train.value_counts()
```

In [66]:
```
y_test.value_counts()
```

Out[66]:
```
0    290
1     30
Name: quality, dtype: int64
```

In [67]:
```
import seaborn as sns
import matplotlib.pyplot as plt

combined_y = np.concatenate((y_test, y_train))
```

In [68]:
```
# Create a countplot using Seaborn
sns.countplot(x=combined_y)

# Set labels for x and y axes
plt.xlabel("Class")
plt.ylabel("Count")

# Set class labels for the x-axis ticks (0 and 1)
plt.xticks([0, 1], ["Class 0", "Class 1"])

# Show the plot
plt.show()
```



In [71]:
```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       290
           1       0.61      0.63      0.62        30

    accuracy                           0.93       320
   macro avg       0.79      0.80      0.79       320
weighted avg       0.93      0.93      0.93       320
```

In [69]:
```
model_random = RandomForestClassifier()
# Train the model on the training data
model_random.fit(x_train, y_train)
predictions = model_random.predict(x_test)
#  Accuracy Score.
print(accuracy_score(y_test, predictions))
```

```
0.928125
```

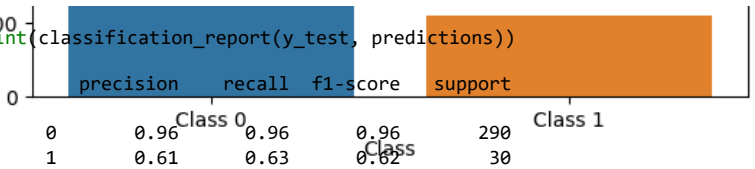
In [72]:
```
column_names = x_test.columns

# Input data for a single observation
input_data = [0.6, 0.5, 0.43, 0.5, 0.5, 0.4, 0.3, 0.3, 0.8, 0.6, 0.4]
input_df = pd.DataFrame([input_data], columns=column_names)

# Make a prediction for the single observation
prediction_single_observation = model_random.predict(input_df)

print("Prediction for the single observation:", prediction_single_observa
```

In [70]:
```
print( "Having only 2 classes increased the accuracy to",accuracy_score(y_
```

```
Having only 2 classes increased the accuracy to 0.928125
```

In [71]:
```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       290
           1       0.61      0.63      0.62        30

    accuracy                           0.93       320
   macro avg       0.79      0.80      0.79       320
weighted avg       0.93      0.93      0.93       320
```



In [69]:
```
model_random = RandomForestClassifier()
# Train the model on the training data
model_random.fit(x_train, y_train)
predictions = model_random.predict(x_test)
#  Accuracy Score.
print(accuracy_score(y_test, predictions))
```

```
0.928125
```

In [72]:
```
# Input data for a single observation
input_data = [0.6, 0.5, 0.43, 0.5, 0.5, 0.4, 0.3, 0.3, 0.8, 0.6, 0.4]
input_df = pd.DataFrame([input_data], columns=column_names)

# Make a prediction for the single observation
prediction_single_observation = model_random.predict(input_df)

print("Prediction for the single observation:", prediction_single_observa

input_df
```

```
Prediction for the single observation: [0]
```

Out[72]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	0.6	0.5	0.43	0.5	0.5	0.4	0.3	0.3	0.8	0.6	0.4

In [70]:
```
print( "Having only 2 classes increased the accuracy to ",accuracy_score(y_
```

```
Having only 2 classes increased the accuracy to 0.928125
```

In [ ]:
```

```