SYAM KRISHNA REDDY PULAGAM

REG NO: 21BAI1725 VIT CHENNAI CAMPUS

AI&ML ASSIGNMENT-4

1. LOADING ALL THE NECCESARY LIBRARIES

In [2]:

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

In [3]:

```
df=pd.read_csv("winequality-red.csv")
df
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56
...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66

1599 rows × 12 columns

In [4]:

```
df.head()
```

Out[4]:

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

2. DATA PREPROCESSING INCLUDING VISUALIZATION

In [5]:

```
df.shape
```

Out[5]:

```
(1599, 12)
```

In [6]:

```
df.columns
```

Out[6]:

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual suga
r',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'densit
y',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [7]:

```
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
```

In [8]:

```
df.describe()
```

Out[8]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total d
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.0
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.4
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.8
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.0
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.0
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.0
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.0
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.0

In [9]:

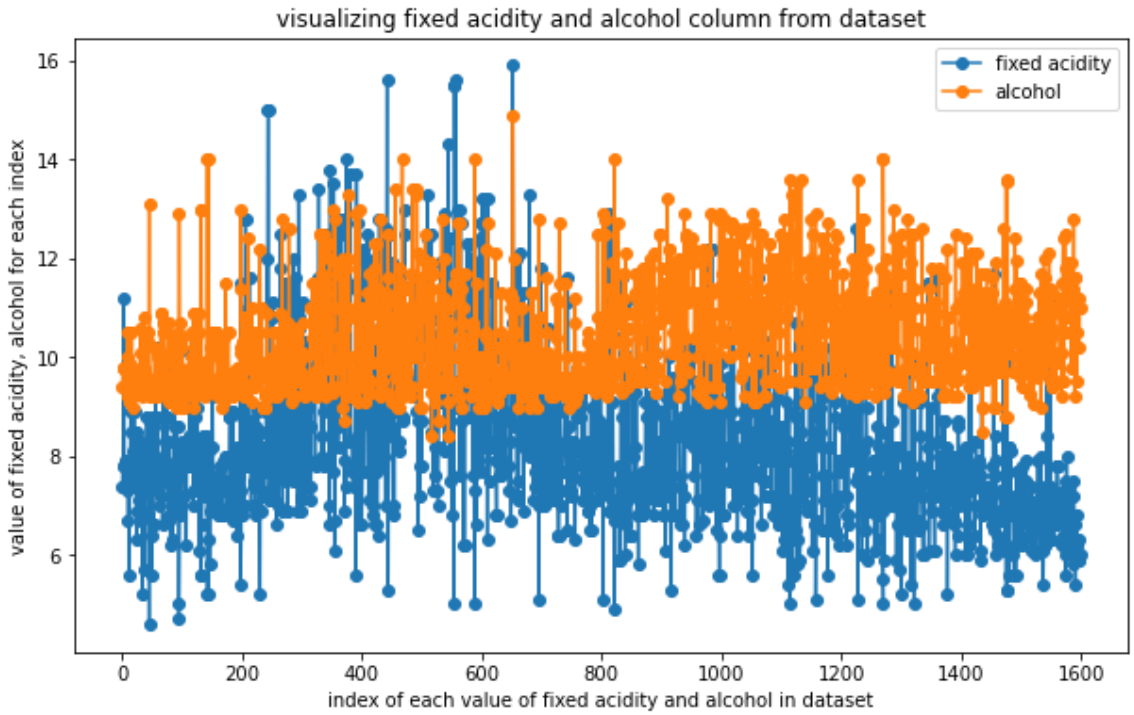
```
df.isnull().any() #no null values
```

Out[9]:

```
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
```

In [10]:

```
plt.figure(figsize=(10,6))
a=['fixed acidity','alcohol']
plt.plot(df['fixed acidity'],"o-")
plt.plot(df['alcohol'],"o-")
plt.xlabel("index of each value of fixed acidity and alcohol in dataset")
plt.ylabel("value of fixed acidity, alcohol for each index")
plt.title("visualizing fixed acidity and alcohol column from dataset")
plt.legend(labels=a)
plt.show()
```



In [11]:

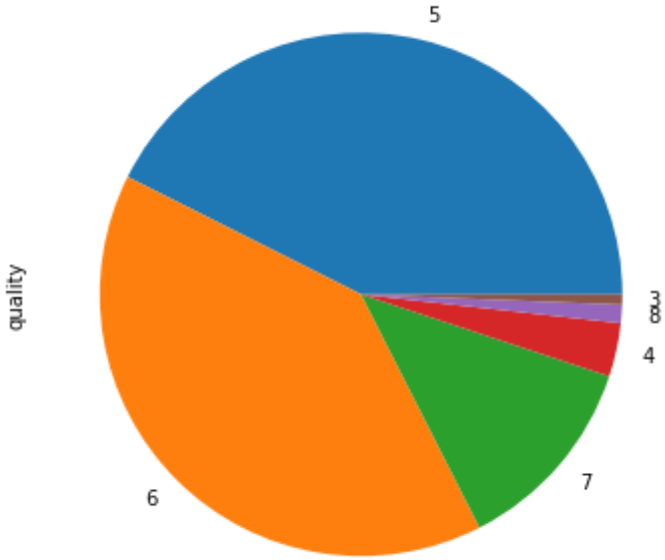
```
df.quality.value_counts()
```

Out[11]:

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

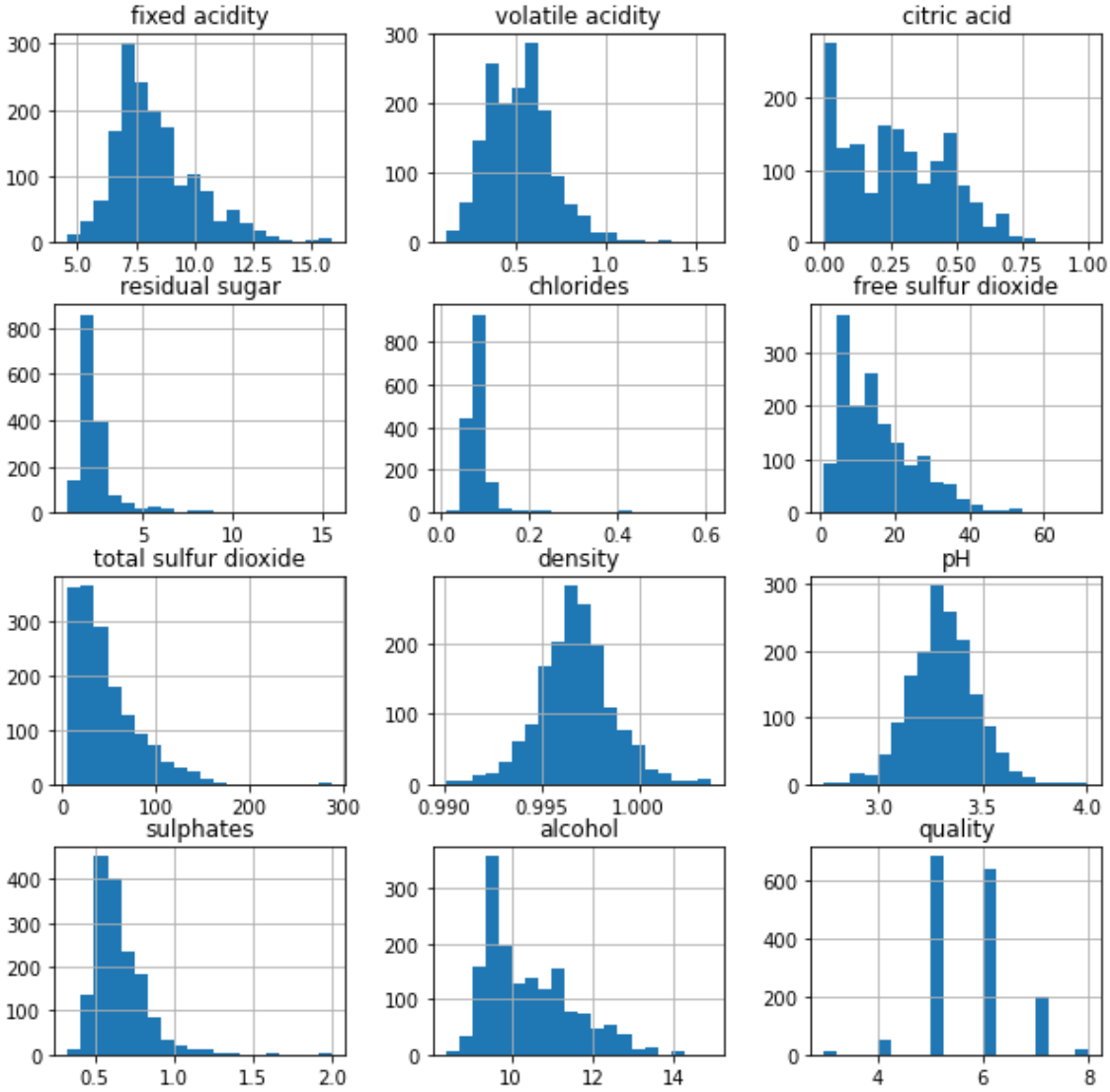
In [12]:

```
plt.figure(figsize=(10,6))
df.quality.value_counts(normalize=True).plot.pie()
plt.show()
```



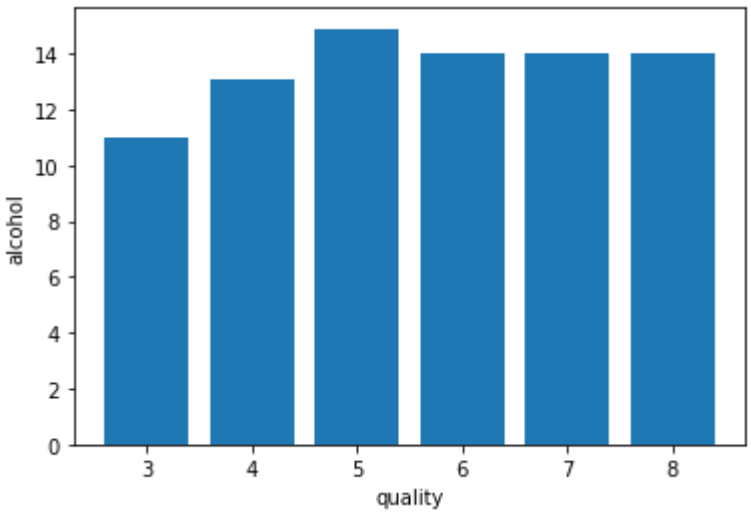
In [13]:

```
df.hist(bins=20, figsize=(10, 10))
plt.show()
```



In [14]:

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

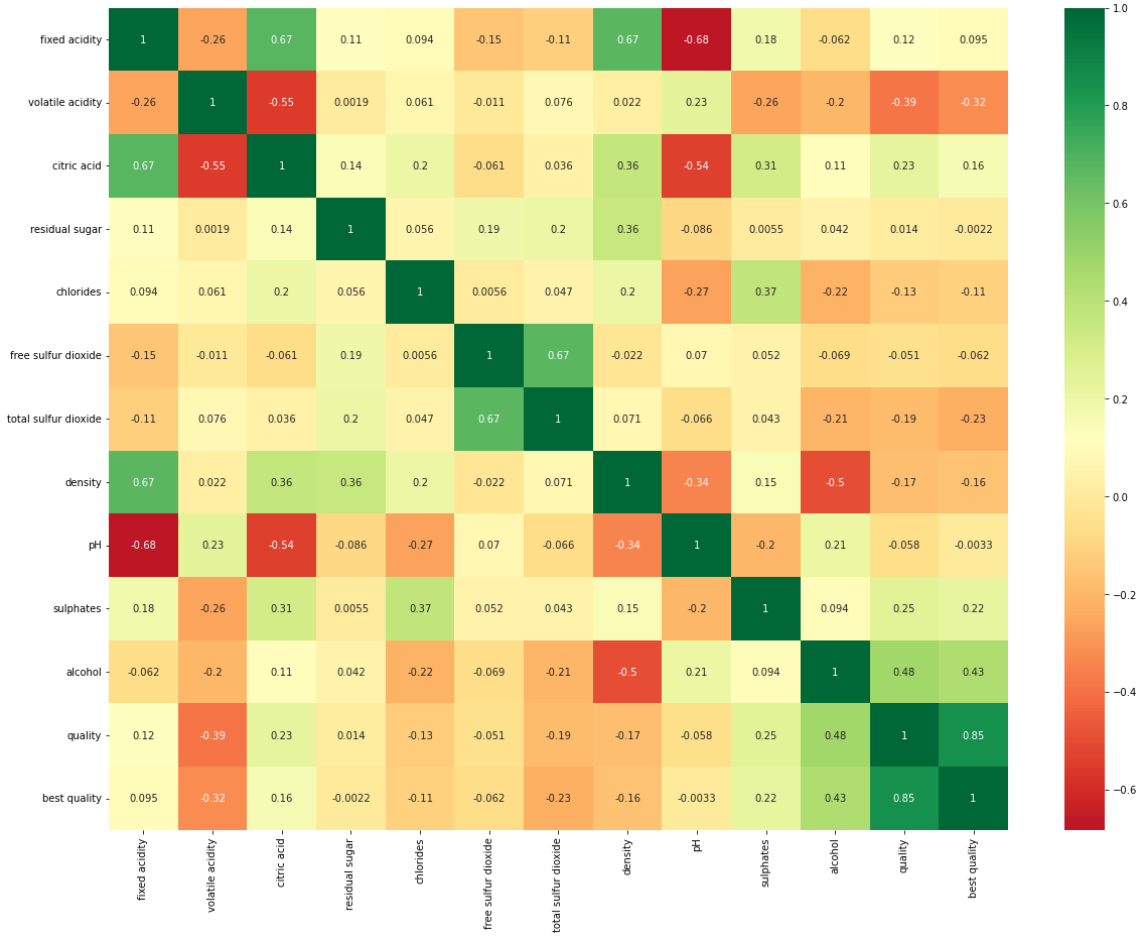


In [45]:

```
plt.subplots(figsize=(20,15))
sns.heatmap(df.corr(), annot=True, cmap = 'RdYlGn', center=0.117)
```

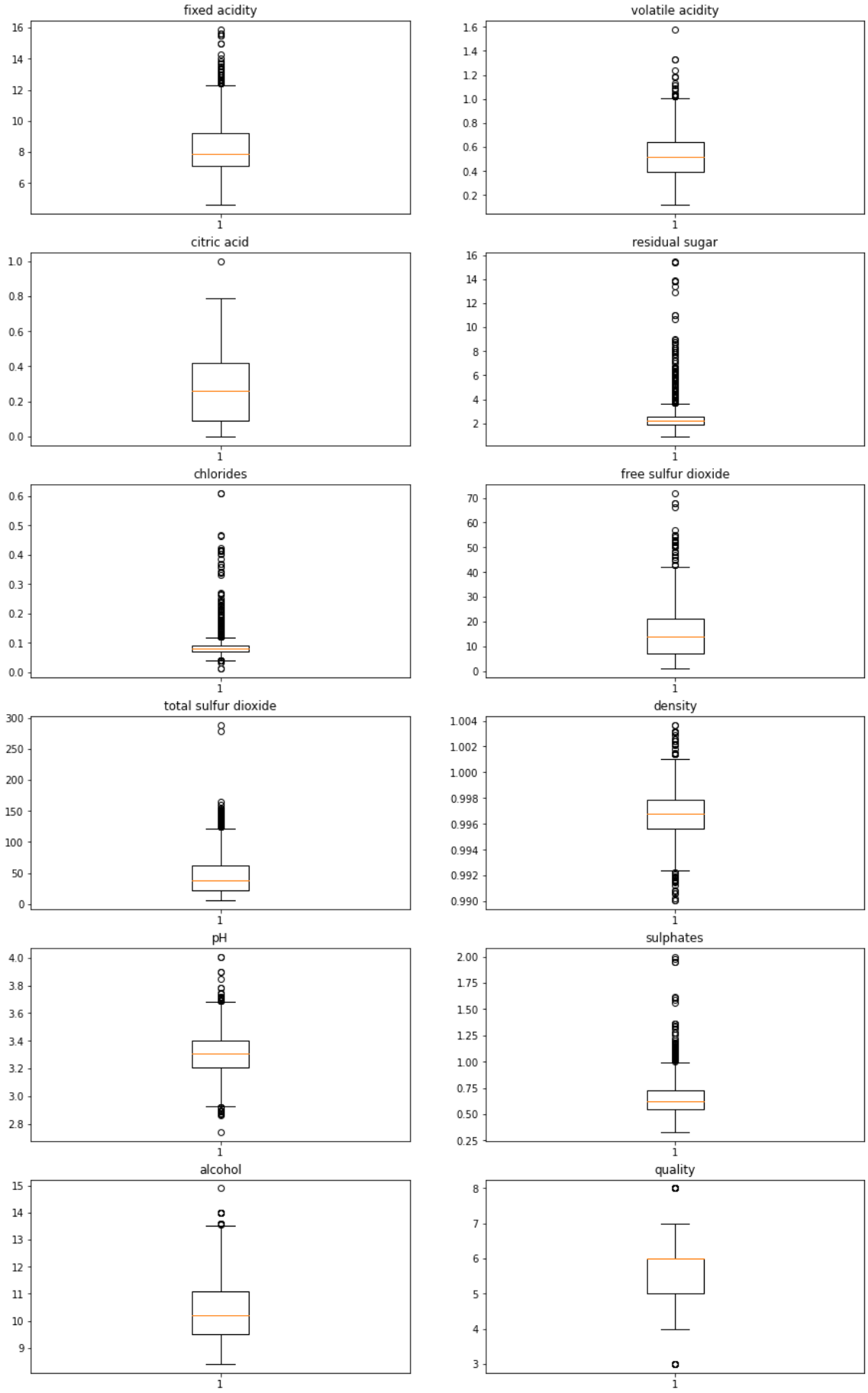
Out[45]:

<AxesSubplot:>



In [16]:

```
x = df.columns
fig, ax = plt.subplots(6, 2, figsize=(15, 25))

for i in range(0, len(x), 2):
    ax[(i // 2)][0].boxplot(df[x[i]])
    ax[(i // 2)][0].set_title(x[i])

    ax[(i // 2)][1].boxplot(df[x[i + 1]])
    ax[(i // 2)][1].set_title(x[i + 1])
#every column had outliers in them
```

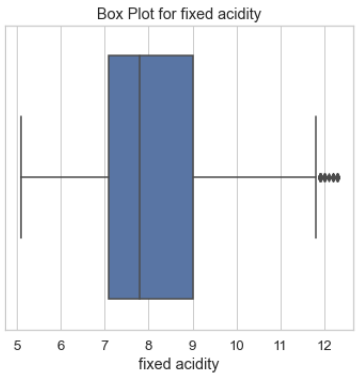

fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol
quality

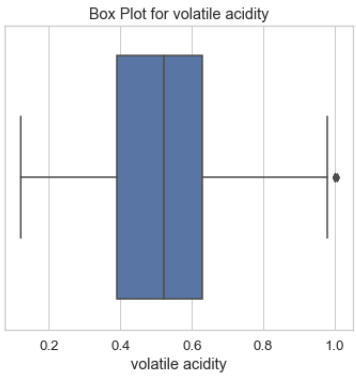
In [68]:

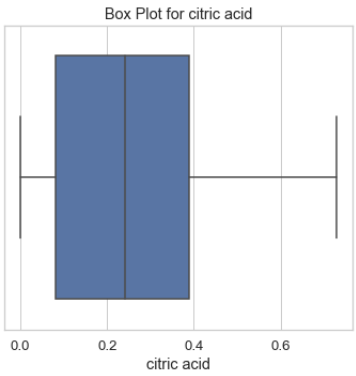
```
import pandas as pd

def remove_outliers(df, columns, threshold=1.5):
    cleaned_df = df.copy()

    for column in columns:
        q1 = df[column].quantile(0.25)
        q3 = df[column].quantile(0.75)
        iqr = q3 - q1

        upper_limit = q3 + threshold * iqr
        lower_limit = q1 - threshold * iqr
        cleaned_df = cleaned_df[(cleaned_df[column] >= lower_limit) & (cleaned_df[column
    return cleaned_df
columns_to_check = ['fixed acidity', 'volatile acidity', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol']
cleaned_df = remove_outliers(df, columns_to_check)
```

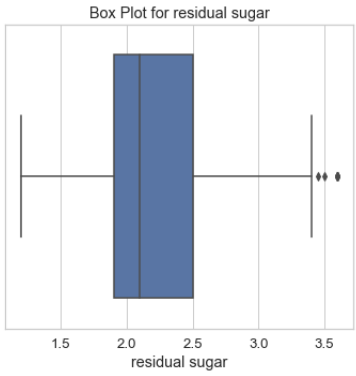
In [73]:

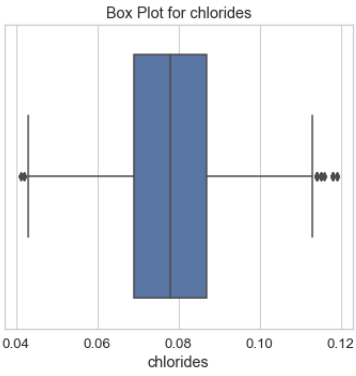
```
def boxplot_all_columns(cleaned_df):
    sns.set(style="whitegrid")
    sns.set_context("notebook", font_scale=1.2)
    num_columns = cleaned_df.shape[1]
    num_rows = (num_columns + 2) // 3  # Adjust the number of columns per row as needed

    fig, axes = plt.subplots(num_rows, 3, figsize=(15, 5 * num_rows))
    axes = axes.flatten()
    for i, column in enumerate(cleaned_df.columns):
        sns.boxplot(x=cleaned_df[column], ax=axes[i])
        axes[i].set_title(f'Box Plot for {column}')
        axes[i].set_xlabel(column)
    for j in range(i + 1, len(axes)):
        fig.delaxes(axes[j])
    plt.tight_layout()
    plt.show()
boxplot_all_columns(cleaned_df)
```

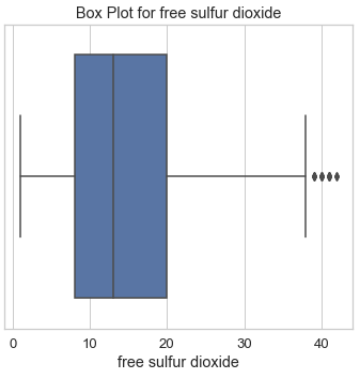

Box Plot for fixed acidity
5 6 7 8 9 10 11 12
fixed acidity

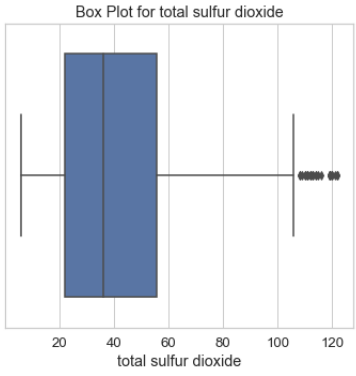

Box Plot for volatile acidity
0.2 0.4 0.6 0.8 1.0
volatile acidity

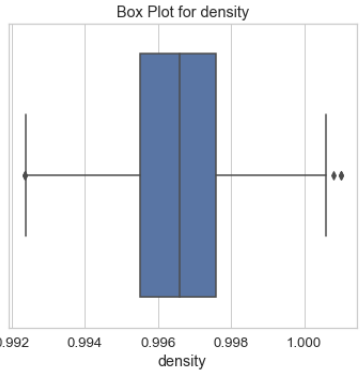

Box Plot for citric acid
0.0 0.2 0.4 0.6
citric acid

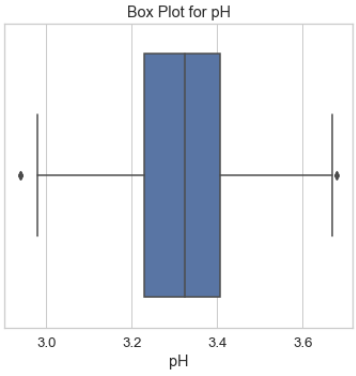

Box Plot for residual sugar
1.5 2.0 2.5 3.0 3.5
residual sugar

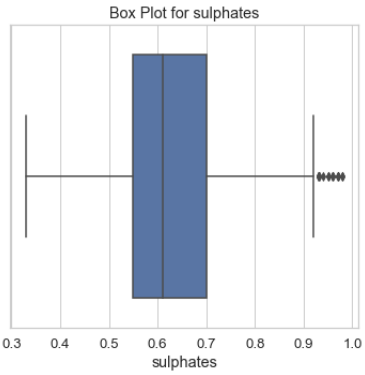

Box Plot for chlorides
0.04 0.06 0.08 0.10 0.12
chlorides

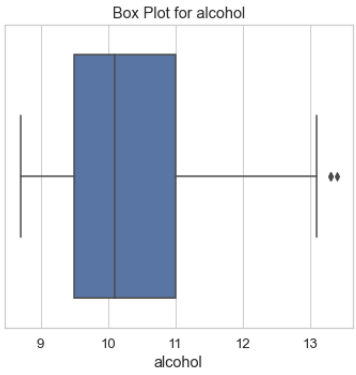

Box Plot for free sulfur dioxide
0 10 20 30 40
free sulfur dioxide

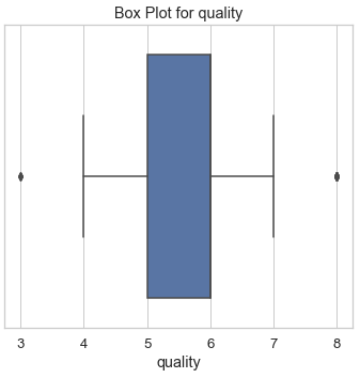

Box Plot for total sulfur dioxide
20 40 60 80 100 120
total sulfur dioxide

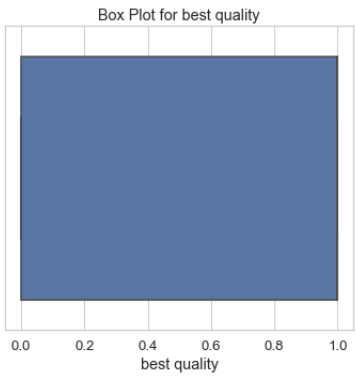

Box Plot for density
0.992 0.994 0.996 0.998 1.000
density


Box Plot for pH
3.0 3.2 3.4 3.6
pH


Box Plot for sulphates
0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
sulphates


Box Plot for alcohol
9 10 11 12 13
alcohol


Box Plot for quality
3 4 5 6 7 8
quality


Box Plot for best quality
0.0 0.2 0.4 0.6 0.8 1.0
best quality

In [70]:

```
cleaned_df.head()
```

Out[70]:

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

In [17]:

```
X=df.drop(columns=['quality'],axis=1)
```

In [18]:

```
from sklearn.preprocessing import MinMaxScaler
scale =MinMaxScaler()
```

In [19]:

```
x = pd.DataFrame(scale.fit_transform(X),columns =X.columns)
x.head()
```

Out[19]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	su
0	0.247788	0.397260	0.00	0.068493	0.106845	0.140845	0.098940	0.567548	0.606299	0
1	0.283186	0.520548	0.00	0.116438	0.143573	0.338028	0.215548	0.494126	0.362205	0
2	0.283186	0.438356	0.04	0.095890	0.133556	0.197183	0.169611	0.508811	0.409449	0
3	0.584071	0.109589	0.56	0.068493	0.105175	0.225352	0.190813	0.582232	0.330709	0
4	0.247788	0.397260	0.00	0.068493	0.106845	0.140845	0.098940	0.567548	0.606299	0

In [20]:

```
x
```

Out[20]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	0.247788	0.397260	0.00	0.068493	0.106845	0.140845	0.098940	0.567548	0.606299
1	0.283186	0.520548	0.00	0.116438	0.143573	0.338028	0.215548	0.494126	0.362205
2	0.283186	0.438356	0.04	0.095890	0.133556	0.197183	0.169611	0.508811	0.409449
3	0.584071	0.109589	0.56	0.068493	0.105175	0.225352	0.190813	0.582232	0.330709
4	0.247788	0.397260	0.00	0.068493	0.106845	0.140845	0.098940	0.567548	0.606299
...	...	...	...	...	...	...	...	...	...
1594	0.141593	0.328767	0.08	0.075342	0.130217	0.436620	0.134276	0.354626	0.559055
1595	0.115044	0.294521	0.10	0.089041	0.083472	0.535211	0.159011	0.370778	0.614173
1596	0.150442	0.267123	0.13	0.095890	0.106845	0.394366	0.120141	0.416300	0.535433
1597	0.115044	0.359589	0.12	0.075342	0.105175	0.436620	0.134276	0.396476	0.653543
1598	0.123894	0.130137	0.47	0.184932	0.091820	0.239437	0.127208	0.397944	0.511811

1599 rows × 11 columns

In [22]:

```
mean=df.quality.mean()
mean=int(mean)
mean
```

Out[22]:

```
5
```

In [23]:

```
df['best quality'] = [1 if x > mean else 0 for x in df.quality]
```

In [24]:

```
df.head()
```

Out[24]:

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

In [25]:

```
y=df['best quality']
```

3.MACHINE LEARNING MODEL BUILDING

USING DATA WITHOUT REMOVING OUTLIERS

In [26]:

```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(X,y, test_size=0.2, random_state=42)
```

In [27]:

```
xtrain.shape
```

Out[27]:

```
(1279, 11)
```

In [28]:

```
ytrain.shape
```

Out[28]:

```
(1279,)
```

MODEL-1 LOGISTIC REGRESSION

In [29]:

```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
```

In [30]:

```
model.fit(xtrain,ytrain)
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:
763: 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 i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-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-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

Out[30]:

```
LogisticRegression()
```

In [32]:

```
from sklearn.metrics import accuracy_score
prediction_train=model.predict(xtrain)
accuracy_train= accuracy_score(ytrain, prediction_train)
accuracy_train
```

Out[32]:

```
0.7482408131352619
```

In [33]:

```
prediction_test=model.predict(xtest)
accuracy_test= accuracy_score(ytest, prediction_test)
accuracy_test
```

Out[33]:

```
0.74375
```

In [34]:

```
from sklearn import metrics
from sklearn.metrics import confusion_matrix
print(confusion_matrix(prediction_test, ytest))
```

```
[[104  45]
 [ 37 134]]
```

MODEL-2 DECISION TREE

In [35]:

```
from sklearn.tree import DecisionTreeClassifier
model_2 = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
model_2.fit(xtrain, ytrain)
```

Out[35]:

```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

In [36]:

```
y_pred_d=model_2.predict(xtest)
accuracy_d=accuracy_score(ytest,y_pred_d)
accuracy_d
```

Out[36]:

```
0.746875
```

In [37]:

```
confusion_matrix(y_pred_d,ytest)
```

Out[37]:

```
array([[101,  41],
       [ 40, 138]], dtype=int64)
```

MODEL-3 RANDOM FOREST CLASSIFIER

In [38]:

```
from sklearn.ensemble import RandomForestClassifier
model_3 = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state
model_3.fit(xtrain, ytrain)
```

Out[38]:

```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=
42)
```

In [39]:

```
y_pred_r=model_3.predict(xtest)
accuracy_r=accuracy_score(ytest,y_pred_r)
accuracy_r
```

Out[39]:

```
0.796875
```

In [41]:

```
confusion_matrix(y_pred_r,ytest)
```

Out[41]:

```
array([[113,  37],
       [ 28, 142]], dtype=int64)
```

MODEL-4 SUPPORT VECTOR MACHINE

In [43]:

```
from sklearn import svm
model_4= svm.SVC(kernel='linear')
model_4.fit(xtrain, ytrain)
```

Out[43]:

```
SVC(kernel='linear')
```

In [44]:

```
y_pred_s=model_4.predict(xtest)
accuracy_s=accuracy_score(y_pred_s,ytest)
accuracy_s
```

Out[44]:

```
0.73125
```

4. MODEL ACCURACY COMPARISION

In [49]:

```
a={'logistic regression':accuracy_test,'decision tree':accuracy_d,'Random Forest':accura
x=0
for i in a:
    x=max(x,a[i])
for i in a:
    if a[i]==x:
        print(i,"has best accuracy of",x)
```

```
Random Forest has best accuracy of 0.796875
```

In [47]:

```
model_comparison = pd.DataFrame ({
     'Accuracy Score': [accuracy_test,accuracy_d,accuracy_r,accuracy_s],
     'Model_Name': ['Logistic Regression','Decision Tree','Random Forest','Support Vecto

})
model_comparison_df = model_comparison.sort_values(by='Accuracy Score',ascending=False)
model_comparison_df = model_comparison_df.set_index('Accuracy Score')
model_comparison_df.reset_index()
```

Out[47]:

	Accuracy Score	Model_Name
0	0.796875	Random Forest
1	0.746875	Decision Tree
2	0.743750	Logistic Regression
3	0.731250	Support Vector Machine

5. Test with random observation

In [61]:

```
input_data = (6.4,0.67,0.08,2.1,0.045,19,48,0.9949,3.49,0.49,11.4)

# changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the input data
std_data = scale.transform(input_data_reshaped)
print("data after applying scaler is: ")
print(std_data)
print("\n")
prediction = model_4.predict(std_data)

print(f"predicted value of wine is {prediction}")

if (prediction[0] == 0):
    print('Wine Quality is good')
else:
    print('Wine Quality is not good')
```

```
data after applying scaler is:
[[0.15929204 0.37671233 0.08       0.08219178 0.05509182 0.25352113
  0.14840989 0.35462555 0.59055118 0.09580838 0.46153846]]


predicted value of wine is [0]
Wine Quality is good
```

USING THE DATA AFTER REMOVING THE OUTLIERS

In [95]:

```
X=cleaned_df.drop(columns=['quality','best quality'],axis=1)
y=cleaned_df['best quality']
```

In [96]:

```
x = pd.DataFrame(scale.fit_transform(X),columns =X.columns)
x.head()
```

Out[96]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	0.319444	0.655367	0.000000	0.291667	0.448718	0.243902	0.241379	0.630058	0.770270
1	0.375000	0.858757	0.000000	0.583333	0.730769	0.585366	0.525862	0.514451	0.351351
2	0.375000	0.723164	0.054795	0.458333	0.653846	0.341463	0.413793	0.537572	0.432432
3	0.847222	0.180791	0.767123	0.291667	0.435897	0.390244	0.465517	0.653179	0.297297
4	0.319444	0.655367	0.000000	0.291667	0.448718	0.243902	0.241379	0.630058	0.770270

In [97]:

```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(X,y, test_size=0.2, random_state=42)
```

In [98]:

```
model.fit(xtrain,ytrain)
prediction_test=model.predict(xtest)
accuracy_test= accuracy_score(ytest, prediction_test)


model_2.fit(xtrain, ytrain)
y_pred_d=model_2.predict(xtest)
accuracy_d=accuracy_score(ytest,y_pred_d)


model_3.fit(xtrain, ytrain)
y_pred_r=model_3.predict(xtest)
accuracy_r=accuracy_score(ytest,y_pred_r)


model_4.fit(xtrain, ytrain)
y_pred_s=model_4.predict(xtest)
accuracy_s=accuracy_score(y_pred_s,ytest)
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:
763: 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 i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-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-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

In [99]:

```
accuracy_test
```

Out[99]:

```
0.7949790794979079
```

In [100]:

```
accuracy_d
```

Out[100]:

```
0.7280334728033473
```

In [101]:

```
accuracy_r
```

Out[101]:

```
0.8158995815899581
```

In [102]:

```
accuracy_s
```

Out[102]:

```
0.7740585774058577
```

In [103]:

```
model_comparison = pd.DataFrame ({
     'Accuracy Score': [accuracy_test,accuracy_d,accuracy_r,accuracy_s],
     'Model_Name': ['Logistic Regression','Decision Tree','Random Forest','Support Vecto

})
model_comparison_df = model_comparison.sort_values(by='Accuracy Score',ascending=False)
model_comparison_df = model_comparison_df.set_index('Accuracy Score')
model_comparison_df.reset_index()
```

Out[103]:

	Accuracy Score	Model_Name
0	0.815900	Random Forest
1	0.794979	Logistic Regression
2	0.774059	Support Vector Machine
3	0.728033	Decision Tree

5. Test with random observation

In [105]:

```
input_data = (5.6,0.615,0,1.6,0.089,16,59,0.9943,3.58,0.52,9.9)

# changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the input data
std_data = scale.transform(input_data_reshaped)
print("data after applying scaler is: ")
print(std_data)
print("\n")
prediction = model_4.predict(std_data)

print(f"predicted value of wine is {prediction}")

if (prediction[0] == 0):
    print('Wine Quality is good')
else:
    print('Wine Quality is not good')
```

```
data after applying scaler is:
[[0.06944444 0.55932203 0.         0.16666667 0.61538462 0.36585366
  0.45689655 0.22543353 0.86486486 0.29230769 0.25531915]]


predicted value of wine is [0]
Wine Quality is good
```