Grapes to Greatness: Machine Learning in Wine Quality Prediction

Task 1 : Load the Dataset

```
# import required libraries #21BAI1796

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


df = pd.read_csv('/content/winequality-red.csv')
df.head()
```

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_
0	7.4	0.70	0.00	1.9	0.076	
1	7.8	0.88	0.00	2.6	0.098	
2	7.8	0.76	0.04	2.3	0.092	
3	11.2	0.28	0.56	1.9	0.075	
4	7.4	0.70	0.00	1.9	0.076	

Task 2 : Data preprocessing including visualization

```
df.shape
```

```
(1599, 12)
```

```
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
```

```
df.isnull().sum() # There are no null values in the dataset.
```

```
fixed_acidity           0
volatile_acidity        0
citric_acid             0
residual_sugar          0
chlorides               0
free_sulfur_dioxide     0
total_sulfur_dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

```
df.describe() # Descriptive Statistics
```

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sul
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	

```
df.corr()
```

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide
fixed_acidity	1.000000	-0.256131	0.671703	0.114777	0.093705	-0.153794
volatile_acidity	-0.256131	1.000000	-0.552496	0.001918	0.061298	-0.010504
citric_acid	0.671703	-0.552496	1.000000	0.143577	0.203823	-0.060978
residual_sugar	0.114777	0.001918	0.143577	1.000000	0.055610	0.187049
chlorides	0.093705	0.061298	0.203823	0.055610	1.000000	0.005562
free_sulfur_dioxide	-0.153794	-0.010504	-0.060978	0.187049	0.005562	1.000000
total_sulfur_dioxide	-0.113181	0.076470	0.035533	0.203028	0.047400	0.667666
density	0.668047	0.022026	0.364947	0.355283	0.200632	-0.021946
pH	-0.682978	0.234937	-0.541904	-0.085652	-0.265026	0.070377
sulphates	0.183006	-0.260987	0.312770	0.005527	0.371260	0.051658
alcohol	-0.061668	-0.202288	0.109903	0.042075	-0.221141	-0.069408
quality	0.124052	-0.390558	0.226373	0.013732	-0.128907	-0.050656

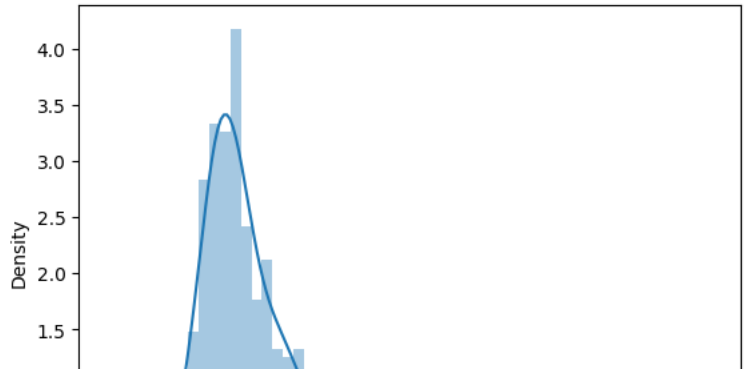
```
# Correlation of dependent varriables with the target variable

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

```
quality                 1.000000
alcohol                 0.476166
sulphates               0.251397
citric_acid             0.226373
fixed_acidity           0.124052
residual_sugar          0.013732
free_sulfur_dioxide    -0.050656
pH                     -0.057731
chlorides              -0.128907
density                -0.174919
total_sulfur_dioxide   -0.185100
volatile_acidity       -0.390558
Name: quality, dtype: float64
```

Univariate Analysis

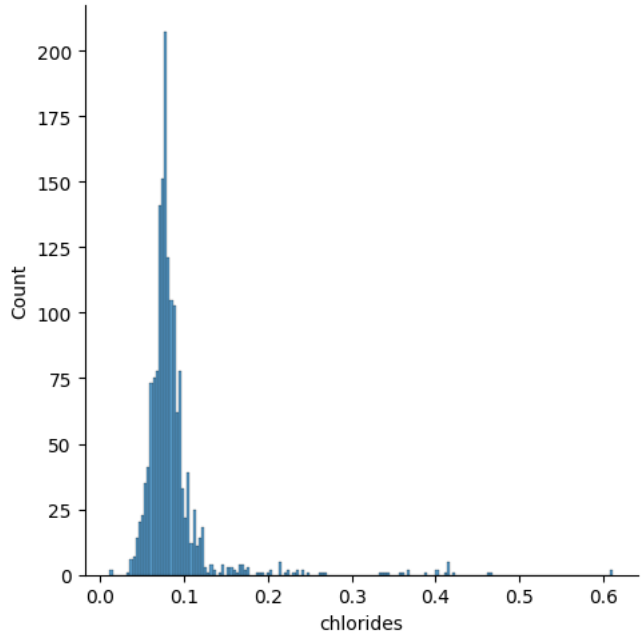
```
sns.distplot(df.sulphates)
```

```
<ipython-input-9-8b271c44c149>:1: UserWarning:

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

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

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

  sns.distplot(df.sulphates)
<Axes: xlabel='sulphates', ylabel='Density'>
```




```
sns.displot(df.chlorides)
```

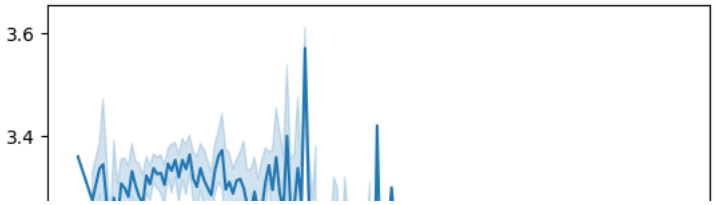
```
<seaborn.axisgrid.FacetGrid at 0x7ddd8a543160>
```



Bivariate Analysis

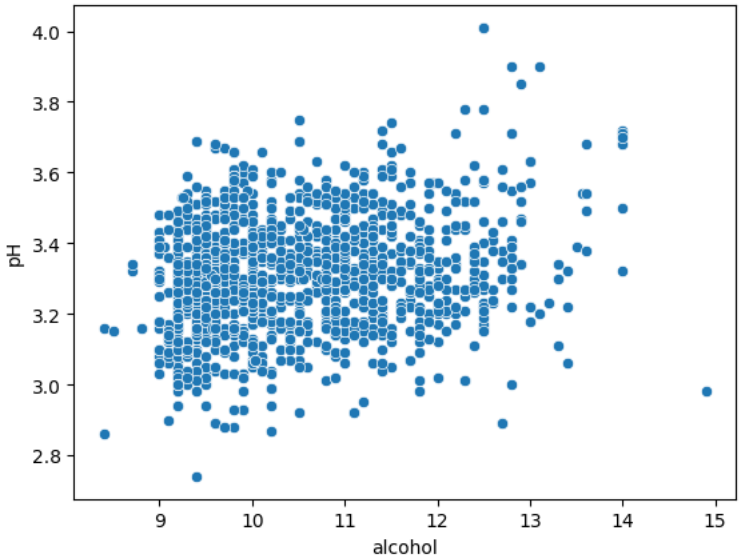
```
sns.lineplot(x=df.sulphates, y=df.pH)
```

```
<Axes: xlabel='sulphates', ylabel='pH'>
```




```
sns.scatterplot(x=df.alcohol, y=df.pH)
```

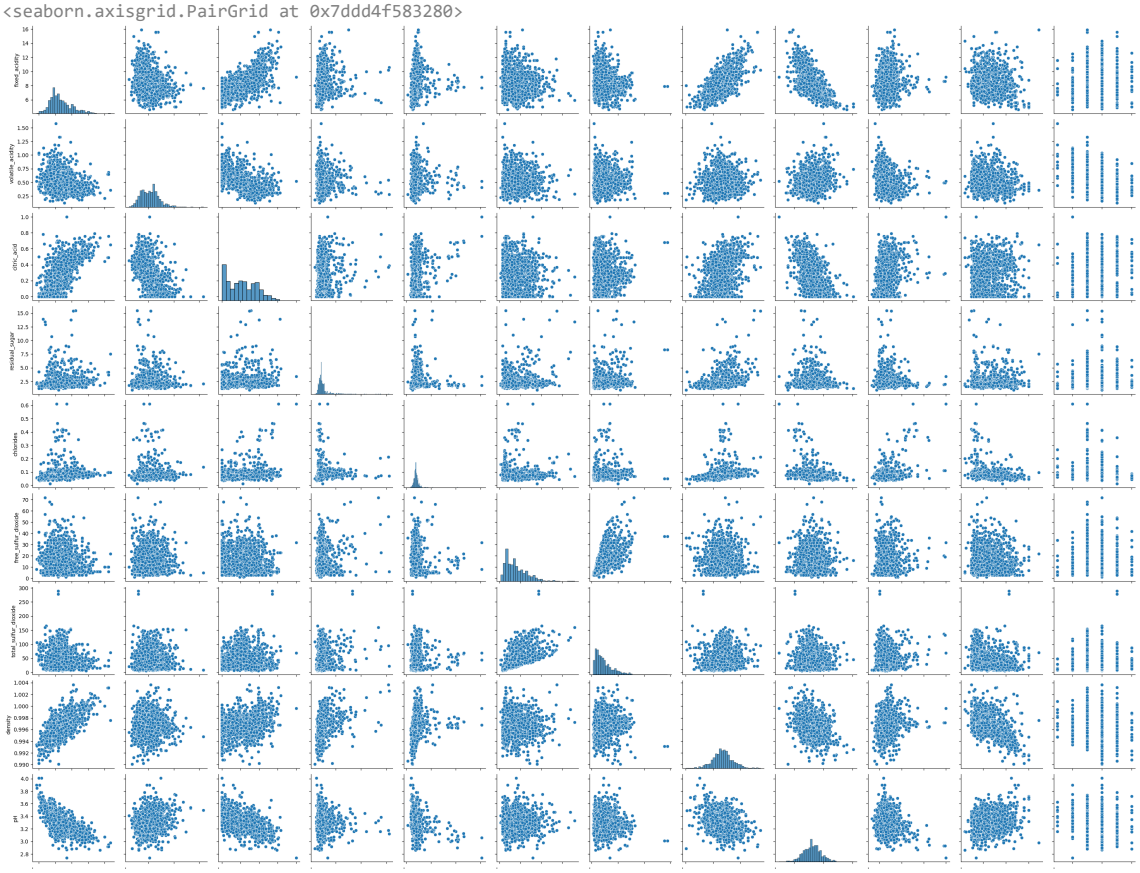
```
<Axes: xlabel='alcohol', ylabel='pH'>
```



Multivariate Analysis

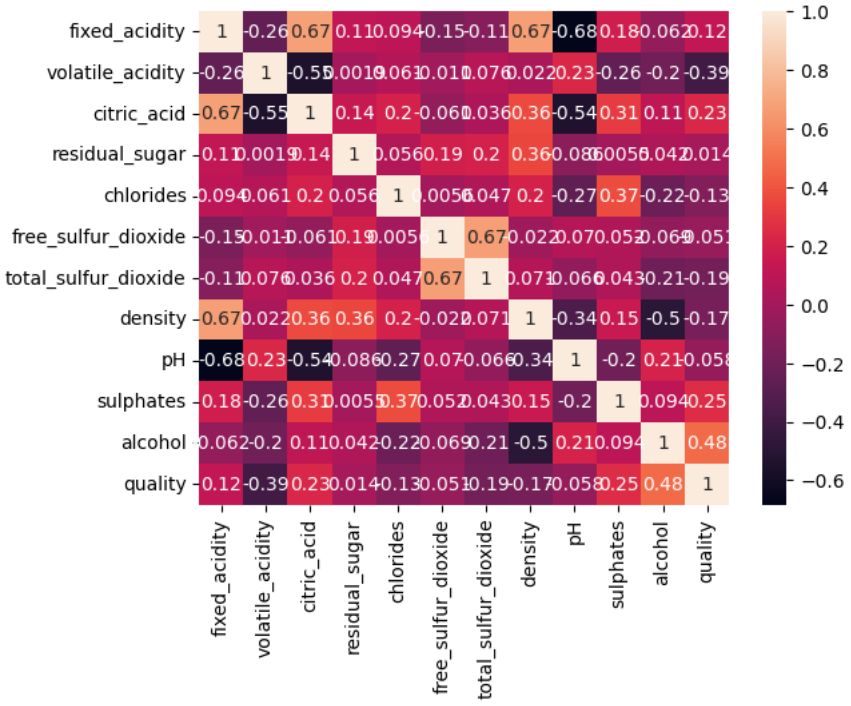
```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7ddd4f583280>
```



```
# Correlation Heatmap

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

```
<Axes: >
```



Outlier Detection and removal by percentile method & IQR MEthod

```
df.head()
```

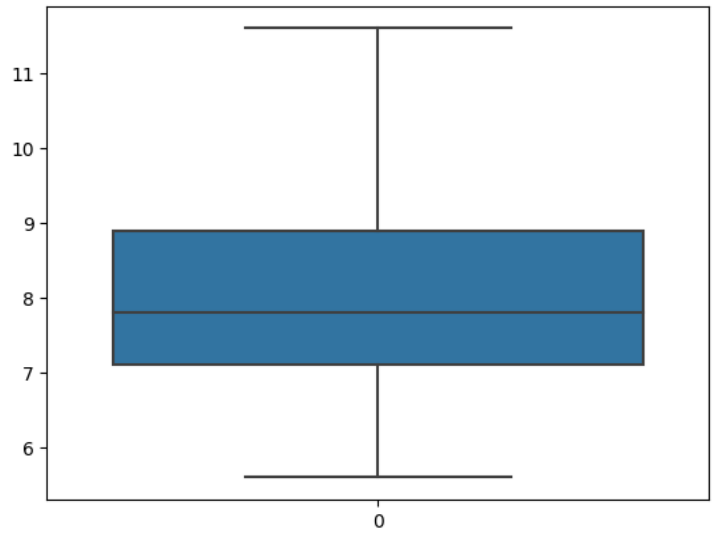
	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_d
0	7.4	0.70	0.00	1.9	0.076	11.0	
1	7.8	0.88	0.00	2.6	0.098	25.0	
2	7.8	0.76	0.04	2.3	0.092	15.0	
3	11.2	0.28	0.56	1.9	0.075	17.0	

```
# Removing outliers from fixed_acidity column

f1 = df.fixed_acidity.quantile(0.25) #Q1
f3 = df.fixed_acidity.quantile(0.75) #Q3
IQR_f = f3 - f1
upper_limit_f = f3+(1.5)*(IQR_f)
lower_limit_f = f1-(1.5)*(IQR_f)
print(f1)
print(f3)
print(IQR_f)
print(upper_limit_f)
print(lower_limit_f)
```

```
7.1
8.9
1.8000000000000007
11.600000000000001
4.399999999999999
```

```
df=df[(df.fixed_acidity<upper_limit_f) & (df.fixed_acidity>lower_limit_f)]
sns.boxplot(df.fixed_acidity)
```

```
<Axes: >
```

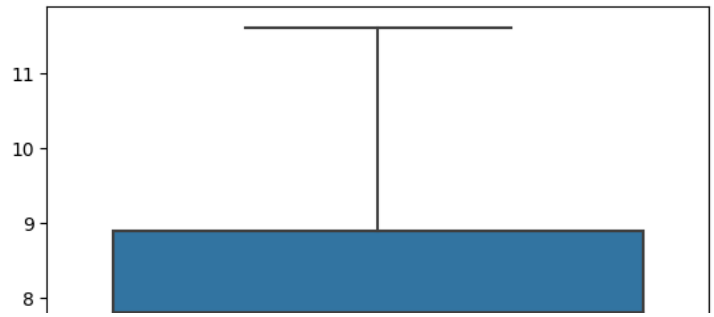



```
fa_01=df.fixed_acidity.quantile(0.01)
fa_9=df.fixed_acidity.quantile(0.98)
print(fa_01)
print(fa_98)
```

```
5.6
11.6
```

```
df=df[(df.fixed_acidity>=fa_01) & (df.fixed_acidity<=fa_98)]
sns.boxplot(df.fixed_acidity)
```

```
<Axes: >
```

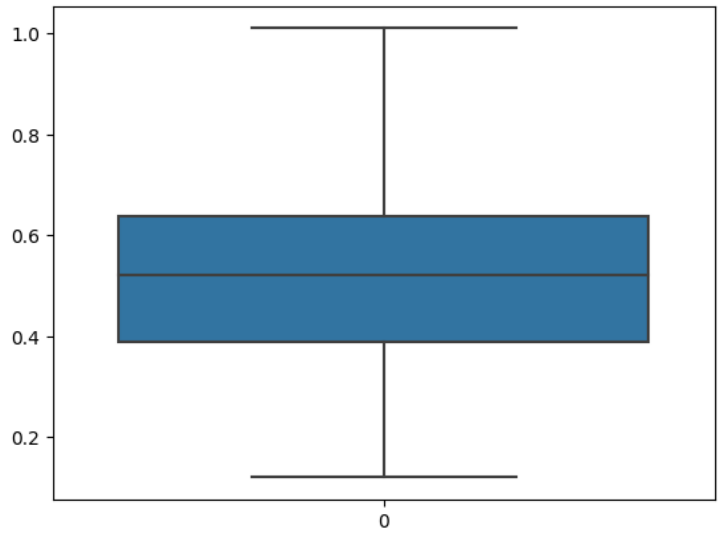



```
# Removing outliers from volatile_acidity column

v1 = df.volatile_acidity.quantile(0.25) #Q1
v3 = df.volatile_acidity.quantile(0.75) #Q3
IQR_v = v3 - v1
upper_limit_v = v3+(1.5)*(IQR_v)
lower_limit_v = v1-(1.5)*(IQR_v)
print(v1)
print(v3)
print(IQR_v)
print(upper_limit_v)
print(lower_limit_v)
```

```
0.3925
0.64
0.2475
1.01125
0.021250000000000047
```

```
df=df[(df.volatile_acidity<upper_limit_v) & (df.volatile_acidity>lower_limit_v)]
sns.boxplot(df.volatile_acidity)
```

```
<Axes: >
```



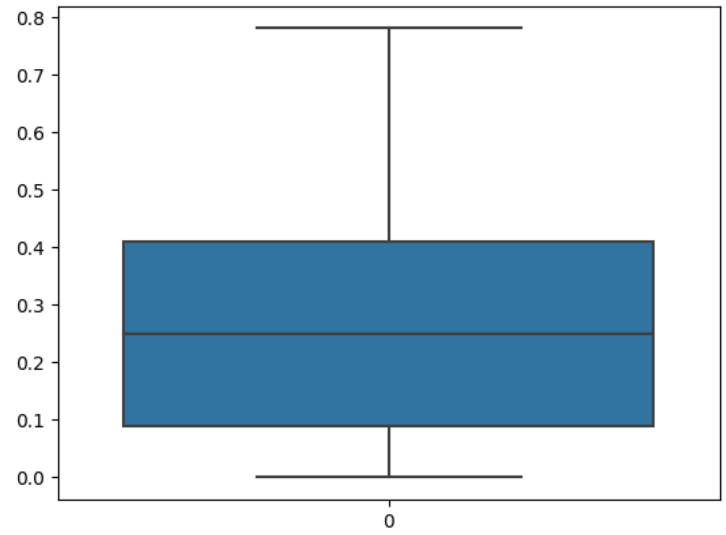

```
# Removing outliers from citric_acid column

c1 = df.citric_acid.quantile(0.25) #Q1
c3 = df.citric_acid.quantile(0.75) #Q3
IQR_c = c3 - c1
upper_limit_c = c3+(1.5)*(IQR_c)
lower_limit_c = c1-(1.5)*(IQR_c)
print(c1)
print(c3)
print(IQR_c)
print(upper_limit_c)
print(lower_limit_c)
```

```
0.09
0.41
0.31999999999999995
```

```
0.8899999999999999
-0.3899999999999999
```

```
df=df[(df.citric_acid<upper_limit_c) & (df.citric_acid>lower_limit_c)]
sns.boxplot(df.citric_acid)
```

```
<Axes: >
```

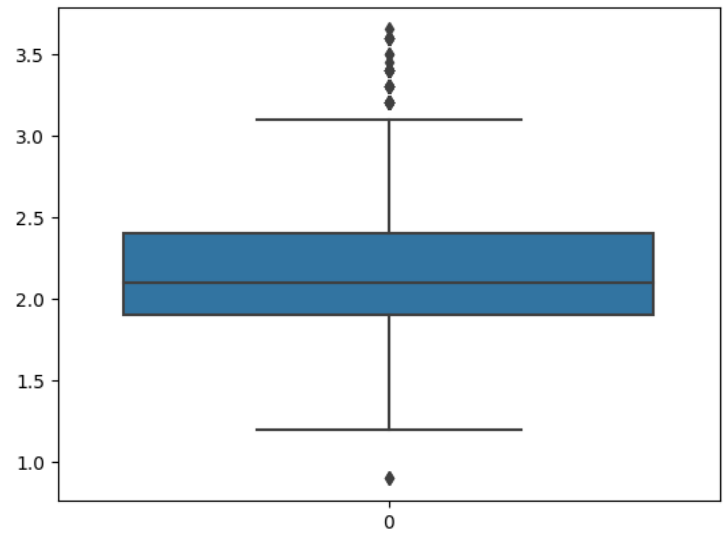



```
# Removing outliers from residual_sugar column

r1 = df.residual_sugar.quantile(0.25) #Q1
r3 = df.residual_sugar.quantile(0.75) #Q3
IQR_r = r3 - r1
upper_limit_r = r3+(1.5)*(IQR_r)
lower_limit_r = r1-(1.5)*(IQR_r)
print(r1)
print(r3)
print(IQR_r)
print(upper_limit_r)
print(lower_limit_r)
```

```
1.9
2.6
0.7000000000000002
3.6500000000000004
0.8499999999999996
```

```
df=df[(df.residual_sugar<upper_limit_r) & (df.residual_sugar>lower_limit_r)]
sns.boxplot(df.residual_sugar)
```

```
<Axes: >
```

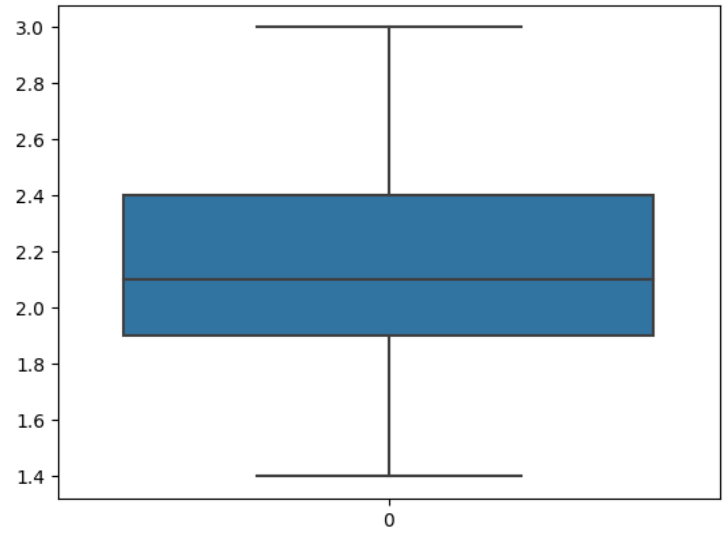


```
rs_02=df.residual_sugar.quantile(0.02)
rs_96=df.residual_sugar.quantile(0.96)
print(rs_02)
print(rs_96)
```

```
1.4
3.0159999999999854
```

```
df=df[(df.residual_sugar>=rs_02) & (df.residual_sugar<=rs_96)]
sns.boxplot(df.residual_sugar)
```

```
<Axes: >
```

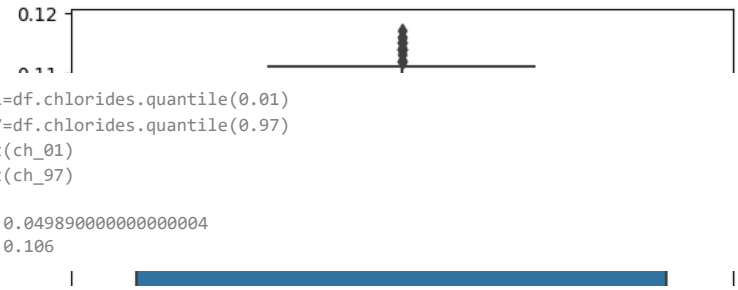



```
# Removing outliers from chlorides column

ch1 = df.chlorides.quantile(0.25) #Q1
ch3 = df.chlorides.quantile(0.75) #Q3
IQR_ch = ch3 - ch1
upper_limit_ch = ch3+(1.5)*(IQR_ch)
lower_limit_ch = ch1-(1.5)*(IQR_ch)
print(ch1)
print(ch3)
print(IQR_ch)
print(upper_limit_ch)
print(lower_limit_ch)
```

```
0.07
0.089
0.01899999999999999
0.11749999999999998
0.04150000000000002
```

```
df=df[(df.chlorides<upper_limit_ch) & (df.chlorides>lower_limit_ch)]
sns.boxplot(df.chlorides)
```

```
<Axes: >
```

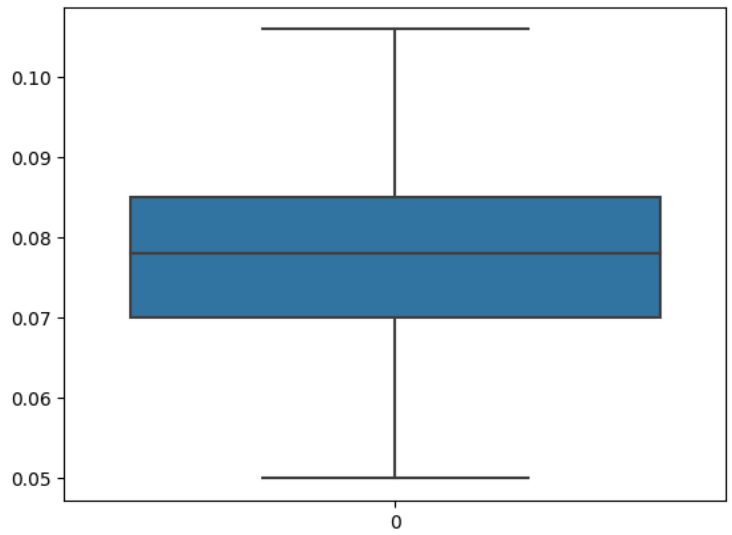



```
ch_01=df.chlorides.quantile(0.01)
ch_97=df.chlorides.quantile(0.97)
print(ch_01)
print(ch_97)
```

```
0.049890000000000004
0.106
```

```
df=df[(df.chlorides>=ch_01) & (df.chlorides<=ch_97)]
sns.boxplot(df.chlorides)
```

```
<Axes: >
```

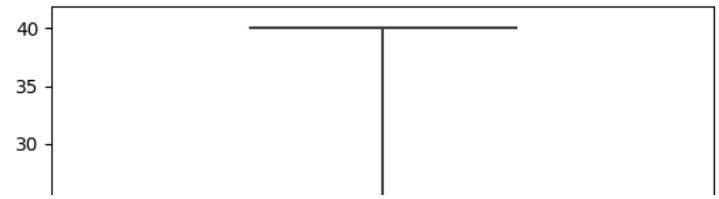



```
# Removing outliers from free_sulfur_dioxide column

fs1 = df.free_sulfur_dioxide.quantile(0.25) #Q1
fs3 = df.free_sulfur_dioxide.quantile(0.75) #Q3
IQR_fs = fs3 - fs1
upper_limit_fs = fs3+(1.5)*(IQR_fs)
lower_limit_fs = fs1-(1.5)*(IQR_fs)
print(fs1)
print(fs3)
print(IQR_fs)
print(upper_limit_fs)
print(lower_limit_fs)
```

```
8.0
21.0
13.0
40.5
-11.5
```

```
df=df[(df.free_sulfur_dioxide<upper_limit_fs) & (df.free_sulfur_dioxide>lower_limit_fs)]
sns.boxplot(df.free_sulfur_dioxide)
```

```
<Axes: >
```

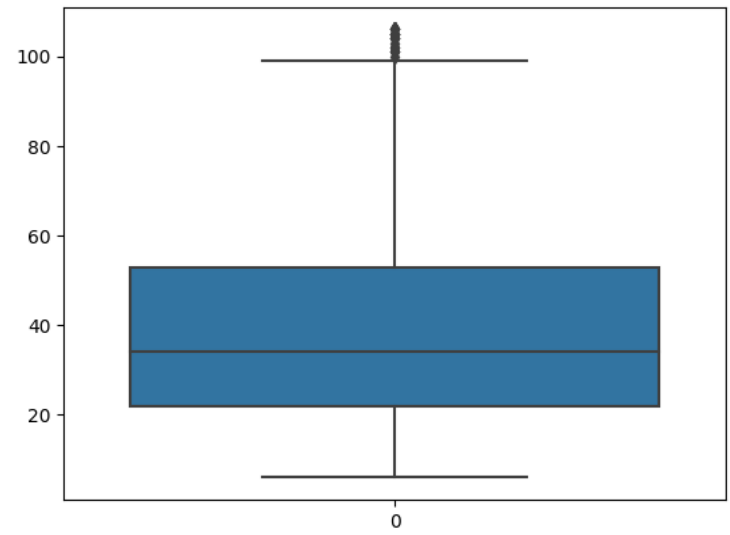



```
# Removing outliers from total_sulfur_dioxide column

ts1 = df.total_sulfur_dioxide.quantile(0.25) #Q1
ts3 = df.total_sulfur_dioxide.quantile(0.75) #Q3
IQR_ts = ts3 - ts1
upper_limit_ts = ts3+(1.5)*(IQR_ts)
lower_limit_ts = ts1-(1.5)*(IQR_ts)
print(ts1)
print(ts3)
print(IQR_ts)
print(upper_limit_ts)
print(lower_limit_ts)
```

```
23.0
57.0
34.0
108.0
-28.0
```

```
df=df[(df.total_sulfur_dioxide<upper_limit_ts) & (df.total_sulfur_dioxide>lower_limit_ts)]
sns.boxplot(df.total_sulfur_dioxide)
```

```
<Axes: >
```

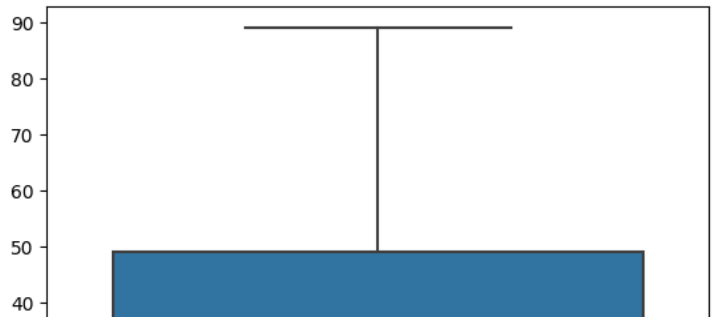



```
ts_01=df.total_sulfur_dioxide.quantile(0.01)
ts_97=df.total_sulfur_dioxide.quantile(0.97)
print(ts_01)
print(ts_97)
```

```
9.0
89.0
```

```
df=df[(df.total_sulfur_dioxide>=ts_01) & (df.total_sulfur_dioxide<=ts_97)]
sns.boxplot(df.total_sulfur_dioxide)
```

```
<Axes: >
```

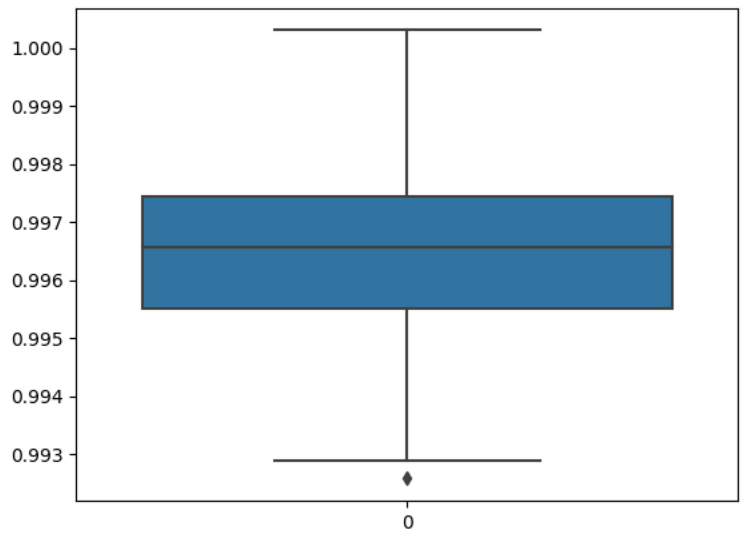



```
# Removing outliers from density column

d1 = df.density.quantile(0.25) #Q1
d3 = df.density.quantile(0.75) #Q3
IQR_d = d3 - d1
upper_limit_d = d3+(1.5)*(IQR_d)
lower_limit_d = d1-(1.5)*(IQR_d)
print(d1)
print(d3)
print(IQR_d)
print(upper_limit_d)
print(lower_limit_d)
```

```
0.9955
0.99745
0.0019499999999998963
1.0003749999999998
0.9925750000000002
```

```
df=df[(df.density<upper_limit_d) & (df.density>lower_limit_d)]
sns.boxplot(df.density)
```

```
<Axes: >
```

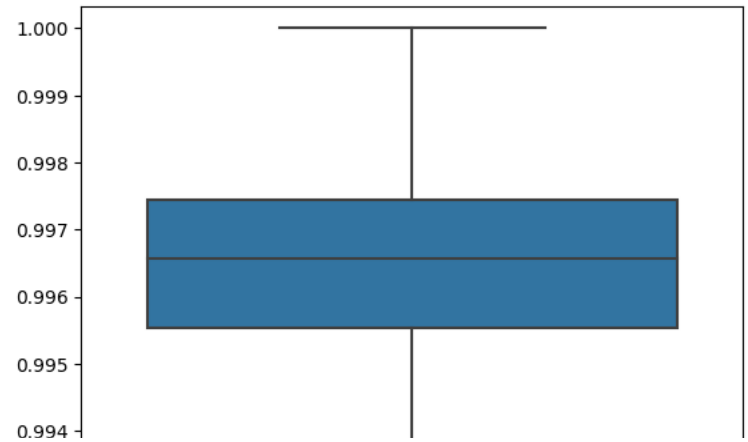



```
d_01=df.density.quantile(0.01)
d_99=df.density.quantile(0.99)
print(d_01)
print(d_99)
```

```
0.9933132
1.0
```

```
df=df[(df.density>=d_01) & (df.density<=d_99)]
sns.boxplot(df.density)
```

```
<Axes: >
```

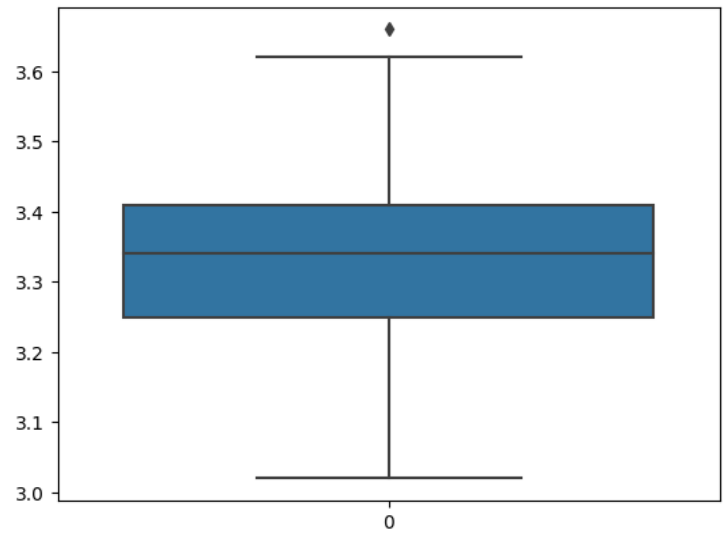



```
# Removing outliers from pH column

pH1 = df.pH.quantile(0.25) #Q1
pH3 = df.pH.quantile(0.75) #Q3
IQR_pH = pH3 - pH1
upper_limit_pH = pH3+(1.5)*(IQR_pH)
lower_limit_pH = pH1-(1.5)*(IQR_pH)
print(pH1)
print(pH3)
print(IQR_pH)
print(upper_limit_pH)
print(lower_limit_pH)
```

```
3.2425
3.41
0.16749999999999998
3.66125
2.99125
```

```
df=df[(df.pH<upper_limit_pH) & (df.pH>lower_limit_pH)]
sns.boxplot(df.pH)
```

```
<Axes: >
```

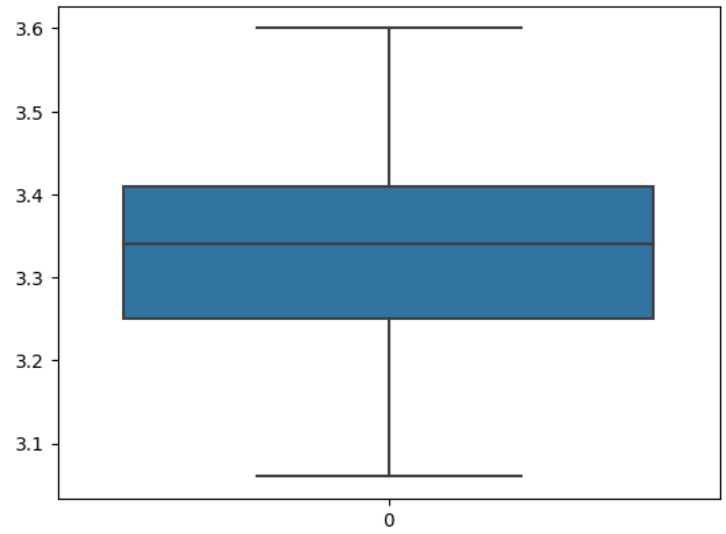



```
pH_01=df.pH.quantile(0.01)
pH_99=df.pH.quantile(0.99)
print(pH_01)
print(pH_99)
```

```
3.06
3.6066
```

```
df=df[(df.pH>=pH_01) & (df.pH<=pH_99)]
sns.boxplot(df.pH)
```

```
<Axes: >
```

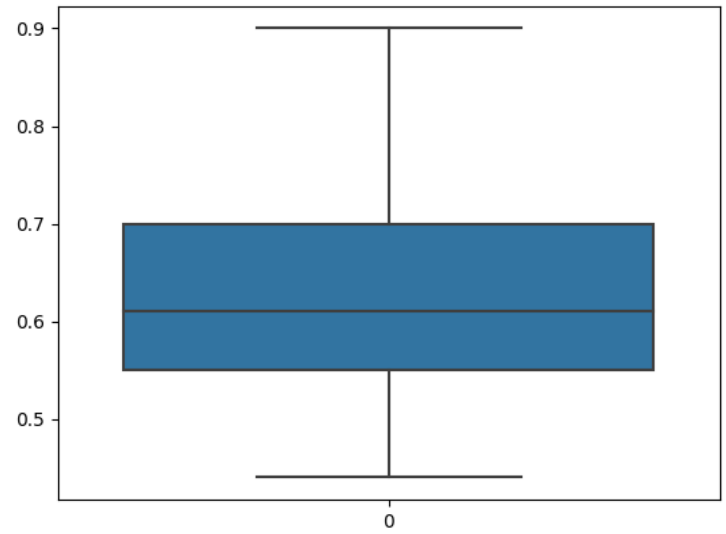



```
# Removing outliers from fixed_acidity column

su_01=df.sulphates.quantile(0.01)
su_98=df.sulphates.quantile(0.98)
print(su_01)
print(su_98)
```

```
0.44
0.9
```

```
df=df[(df.sulphates>=su_01) & (df.sulphates<=su_98)]
sns.boxplot(df.sulphates)
```

```
<Axes: >
```

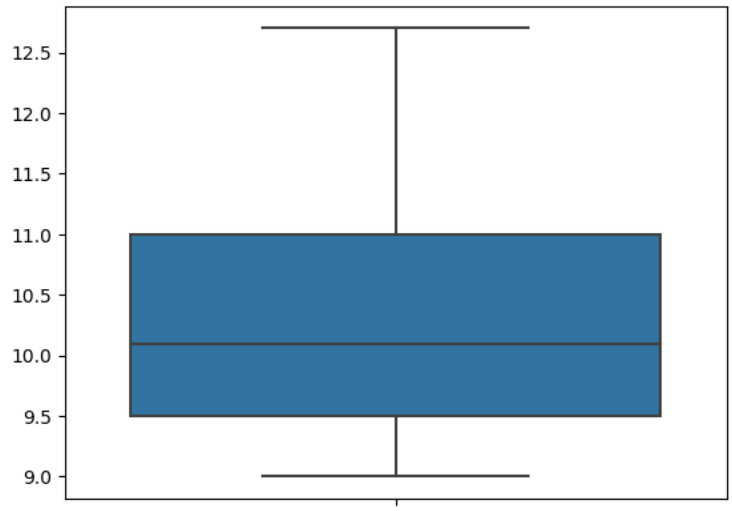



```
# Removing outliers from alcohol column

a_01=df.alcohol.quantile(0.01)
a_99=df.alcohol.quantile(0.99)
print(a_01)
print(a_99)
```

```
9.0
12.724
```

```
df=df[(df.alcohol>=a_01) & (df.alcohol<=a_99)]
sns.boxplot(df.alcohol)
```

```
<Axes: >
```



Therefore all the outliers are removed

Task - 3 : Machine Learning Model Building

```
# split into X and y

X =df.iloc[:,:-1]
X.head()
```

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_d
0	7.4	0.70	0.00	1.9	0.076	11.0	
1	7.8	0.88	0.00	2.6	0.098	25.0	
2	7.8	0.76	0.04	2.3	0.092	15.0	
3	11.2	0.28	0.56	1.9	0.075	17.0	
4	7.4	0.70	0.00	1.9	0.076	11.0	

```
Y =df.quality
Y.head()
```

```
0    5
1    5
2    5
3    6
4    5
Name: quality, dtype: int64
```

Label Binarisation (Conidering alcohol quality > 7 as good and assigning '1' to it else assigning '0')

```
Y = df['quality'].apply(lambda y_value: 1 if y_value>=7 else 0)
```

```
print(Y)
```

```
0       0
1       0
2       0
3       0
4       0
       ..
1593    0
1594    0
1595    0
1596    0
1597    0
Name: quality, Length: 866, dtype: int64
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=3)
```

```
X_train.shape
```

```
(692, 11)
```

```
X_test.shape
```

```
(174, 11)
```

```
print(Y.shape, Y_train.shape, Y_test.shape)
```

```
(866,) (692,) (174,)
```

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
model1 = DecisionTreeClassifier(max_depth=2,splitter='best',criterion='entropy')
model1.fit(X_train,Y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=2)
```

```
d_y_predict = model1.predict(X_test)
d_y_predict
```

```
array([1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0])
```

```
d_y_predict_train = model1.predict(X_train)
```

Task - 4 : Evaluating the model (Decision tree classifier)

```
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
print('Testing Accuracy = ', accuracy_score(Y_test,d_y_predict))
print('Training Accuracy = ', accuracy_score(Y_train,d_y_predict_train))
```

```
Testing Accuracy =  0.8793103448275862
Training Accuracy =  0.8916184971098265
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
model2 =RandomForestClassifier(n_estimators=200,criterion='entropy')
model2.fit(X_train,Y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=200)
```

```
r_y_predict = model2.predict(X_test)
r_y_predict_train = model2.predict(X_train)
```

Task - 4 : Evaluating Random Forest Model

```
print('Testing Accuracy = ', accuracy_score(Y_test,r_y_predict))
print('Training Accuracy = ', accuracy_score(Y_train,r_y_predict_train))
```

```
Testing Accuracy =  0.9425287356321839
Training Accuracy =  1.0
```

Naive Bayesian Classification Model

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train,Y_train)
```

```
GaussianNB()
```

```
y_pred2 = gnb.predict(X_test)
y_pred2
```

```
array([1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0])
```

Task - 4 : Evaluating Naive Bayesian Classification Model

```
from sklearn.metrics import accuracy_score
gnb_acc=accuracy_score(Y_test,y_pred2)
gnb_acc
```

```
0.8850574712643678
```

Accuracies of all the algorithms used in model nuilding phase :

Decision Tree Classification : 87.93 %

Random Forset Classification : 94.25 %

Naive Bayesian Classification : 88.50 %

Conclusion : Random Forest Classifier Model is best suited for the wine quality dataset.

Task - 5 : Test with random observation

```
input_data = [7.9, 1.0, 0, 3.0, 0.08, 30, 100, 0.9562, 3.1, 0.74, 11.5]
prediction = model1.predict([input_data])
prediction
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but DecisionTreeClassifie
  warnings.warn(
array([0])
```

According to "decision tree classifier" model, the above random observation gives prediction "array([0])" i.e., bad quality alcohol

```
input_data_2 = [7.9, 1.0, 0, 3.0, 0.08, 30, 100, 0.9562, 3.1, 0.74, 11.5]
prediction2 = model2.predict([input_data_2])
prediction2
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but RandomForestClassifie
  warnings.warn(
array([0])
```

According to "Random Forest classifier" model, the above random observation gives prediction "array([0])" i.e., bad quality alcohol

```
input_data_3 = [7.9, 1.0, 0, 3.0, 0.08, 30, 100, 0.9562, 3.1, 0.74, 11.5]
prediction3 = gnb.predict([input_data_3])
prediction3
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but GaussianNB was fitted
  warnings.warn(
array([0])
```

According to "Naive Bayesian classifier" model, the above random observation gives prediction "array([0])" i.e., bad quality alcohol

CONCLUSION : For the same random observation, all the three models gave the "alchohol quality is BAD"

The End !!!!