Red Wine Quality Prediction

IMPORT LIBRARIES AS WELL AS DATASET

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
import os
%matplotlib inline
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv("Downloads/Python_Project_6_SVM.csv")
```

In [3]:
```
df
```

Out[3]:

	Quality_Category	volatileacidity	citricacid	residualsugar	chlorides	freesulfurdioxide	total:
0	0	0.30	0.34	1.6	0.049	14	
1	0	0.23	0.32	8.5	0.058	47	
2	0	0.28	0.40	6.9	0.050	30	
3	0	0.32	0.16	7.0	0.045	30	
4	0	0.27	0.36	20.7	0.045	45	
...	...	...	...	...	...	...	
4889	0	0.21	0.29	1.6	0.039	24	
4890	0	0.32	0.36	8.0	0.047	57	
4891	0	0.24	0.19	1.2	0.041	30	
4892	1	0.29	0.30	1.1	0.022	20	
4893	0	0.21	0.38	0.8	0.020	22	

4894 rows × 10 columns

In [4]:
```
#Make a Copy of the Original dataset Which can help me in future
df1 = df.copy(deep=True)
df2 = df.copy(deep=True)
```

DATA PREPROCESSING

In [5]:
```
#chaeking for the missing value
df.isnull().sum()
```

Out[5]:
```
Quality_Category      0
volatileacidity       0
citricacid            0
residualsugar         0
chlorides             0
freesulfurdioxide     0
totalsulfurdioxide    0
density               0
sulphates             0
alcohol               0
dtype: int64
```

In [6]:
```
df.info()
```

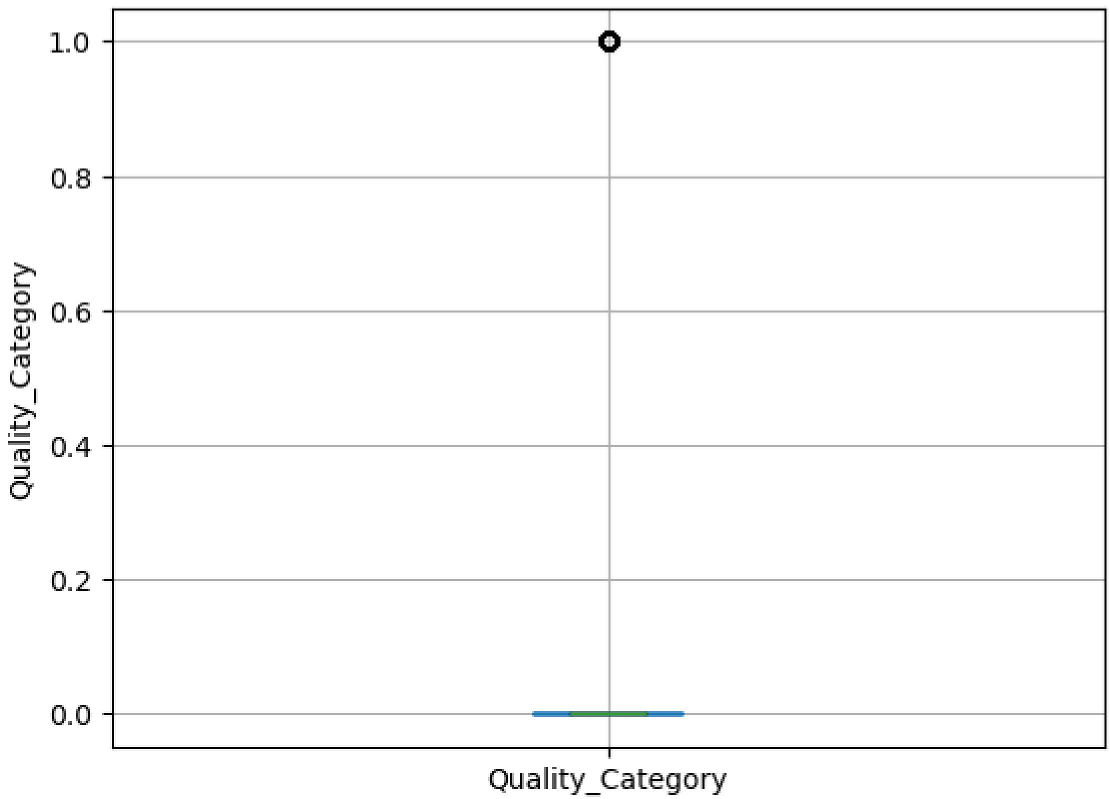
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4894 entries, 0 to 4893
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Quality_Category    4894 non-null   int64  
 1   volatileacidity     4894 non-null   float64
 2   citricacid          4894 non-null   float64
 3   residualsugar       4894 non-null   float64
 4   chlorides           4894 non-null   float64
 5   freesulfurdioxide   4894 non-null   int64  
 6   totalsulfurdioxide  4894 non-null   int64  
 7   density             4894 non-null   float64
 8   sulphates           4894 non-null   float64
 9   alcohol             4894 non-null   float64
dtypes: float64(7), int64(3)
memory usage: 382.5 KB
```

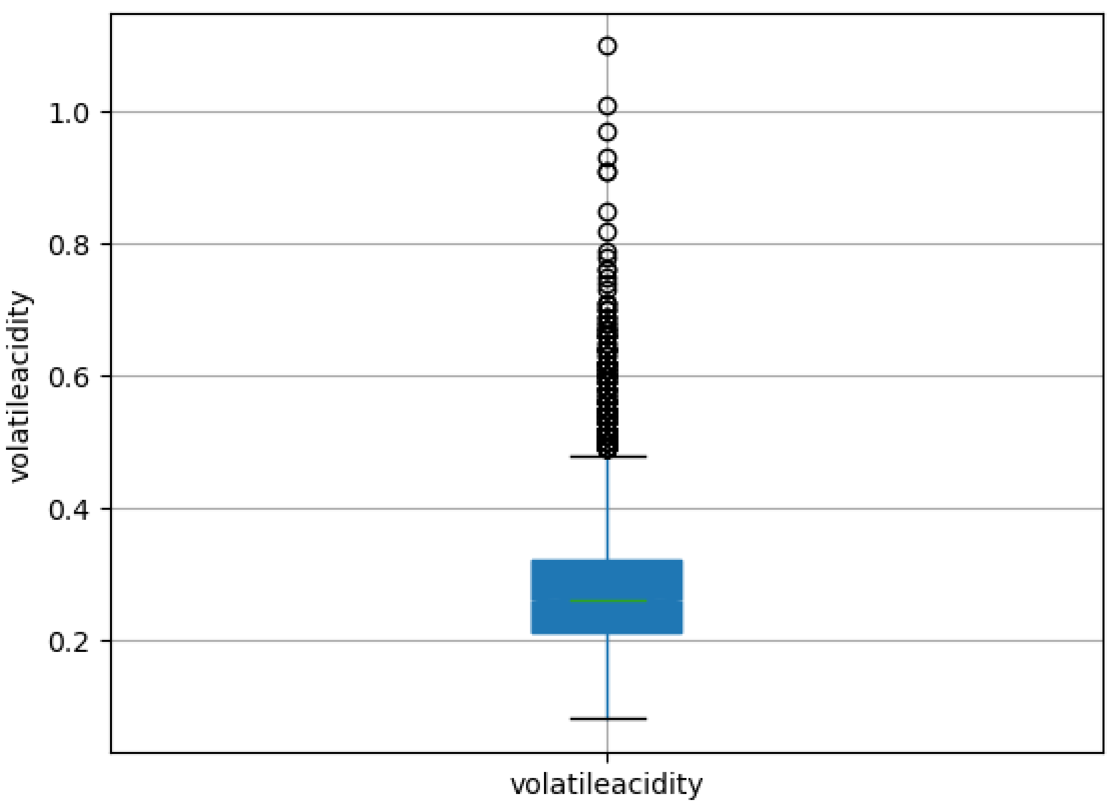
In [7]:
```
df.columns
```

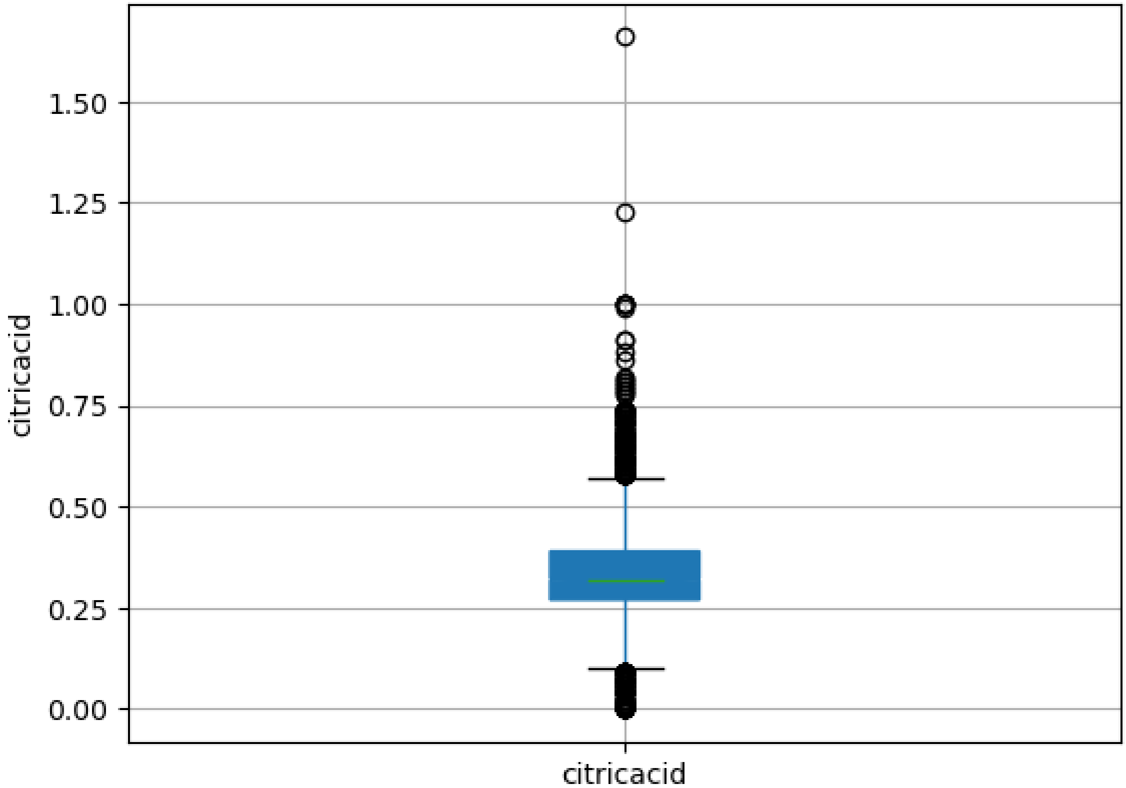
Out[7]:
```
Index(['Quality_Category', 'volatileacidity', 'citricacid', 'residualsugar',
       'chlorides', 'freesulfurdioxide', 'totalsulfurdioxide', 'density',
       'sulphates', 'alcohol'],
      dtype='object')
```

EXPLORATORY DATA ANALYSIS

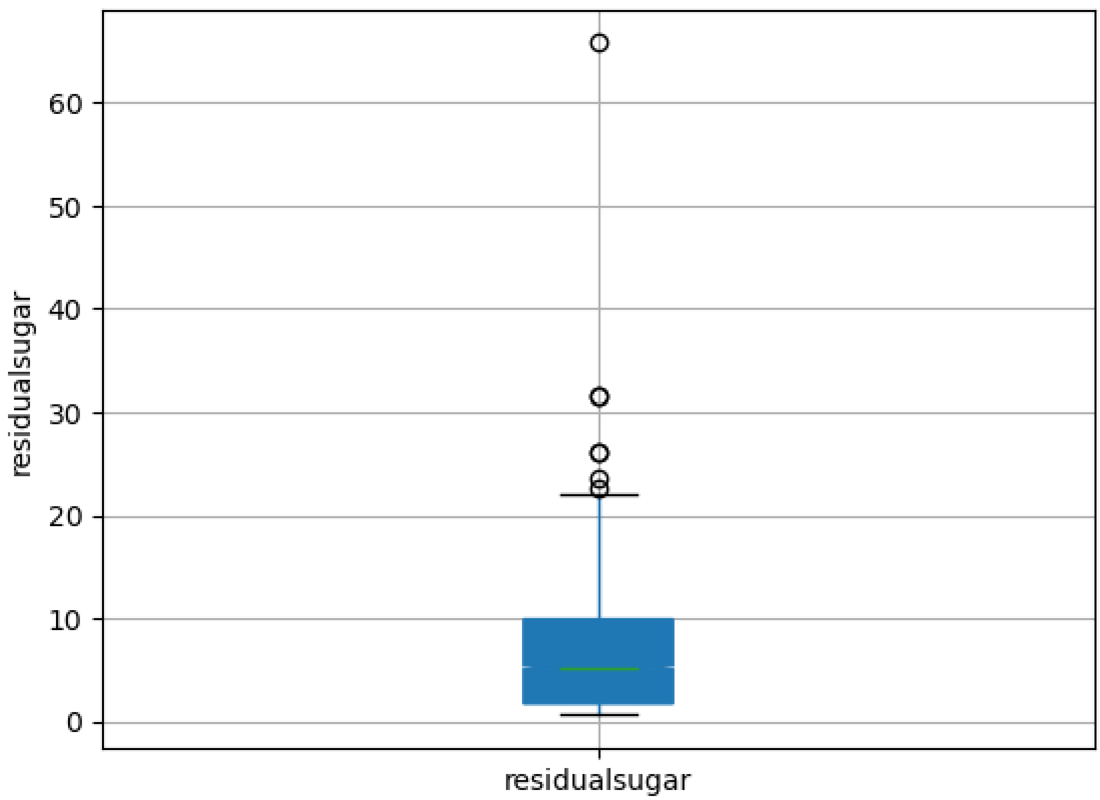
In [8]:
```
#Here, in this dataset have not Categorical features. So, now create EDA for n
umerical data.
#First, check the Outliers
for i in df:
    df.boxplot(column=i, patch_artist = True, notch ='True')
    plt.ylabel(i)
    plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
Quality_Category
Quality_Category

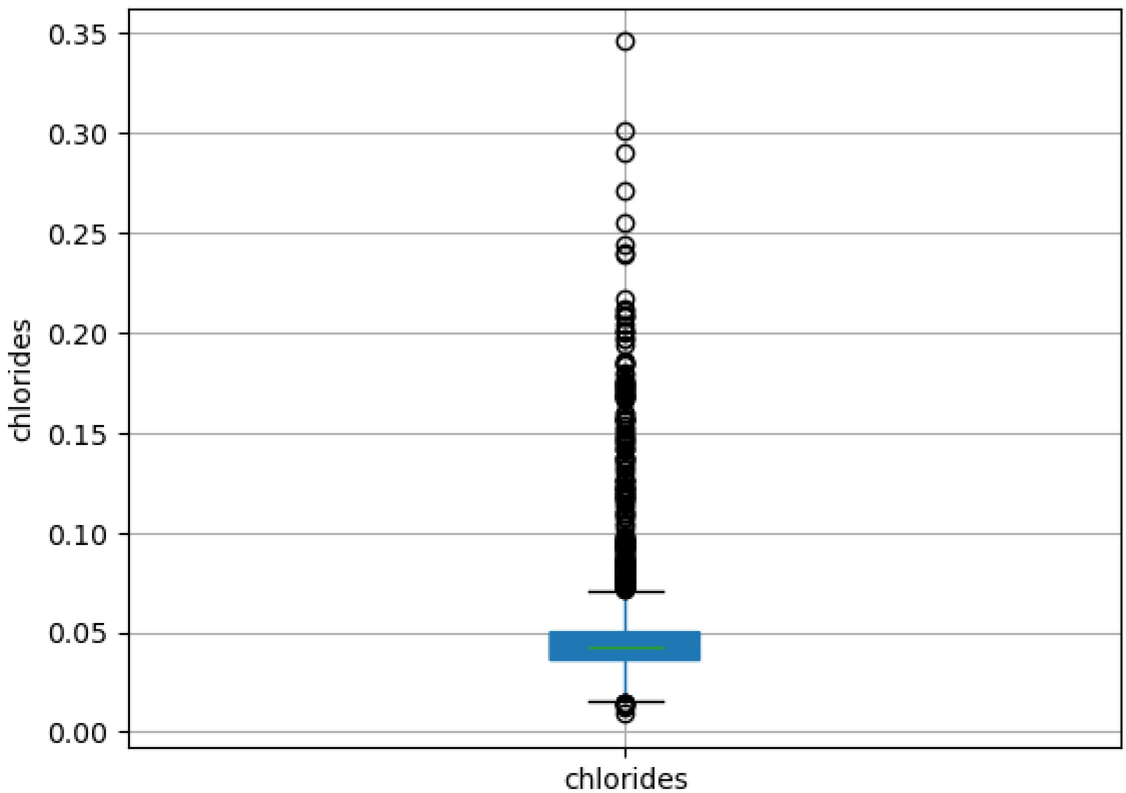

1.0
0.8
0.6
0.4
0.2
volatileacidity
volatileacidity

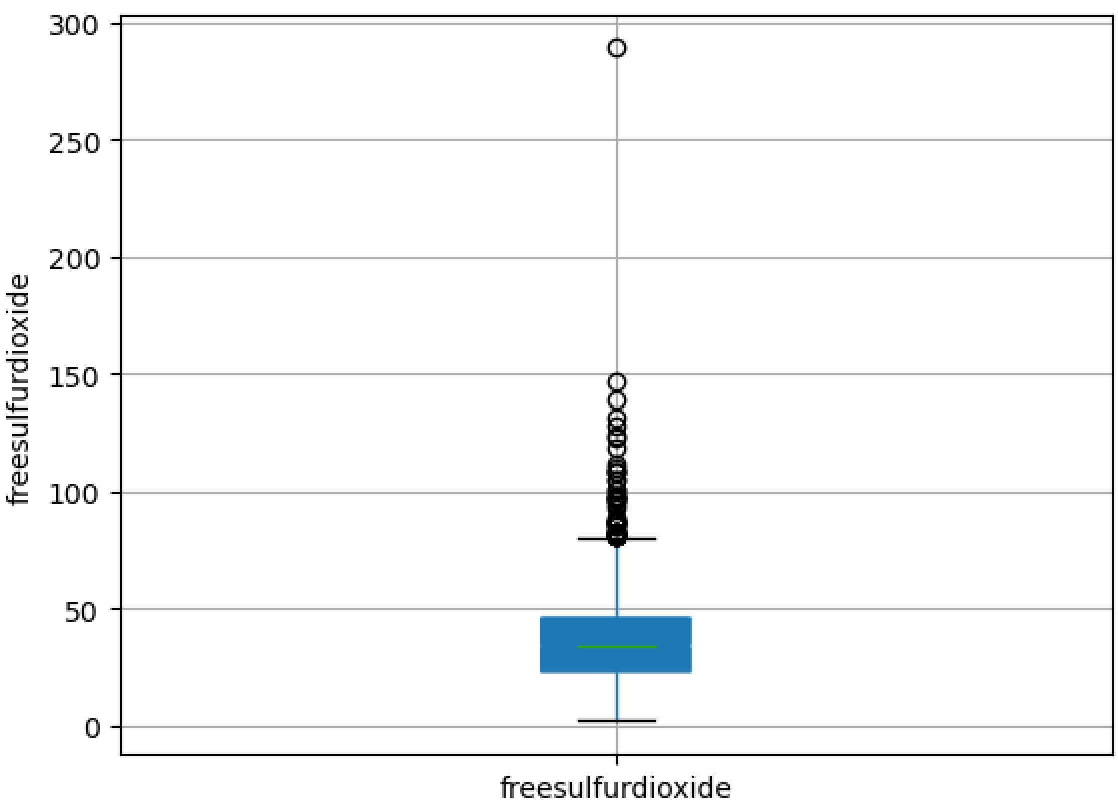


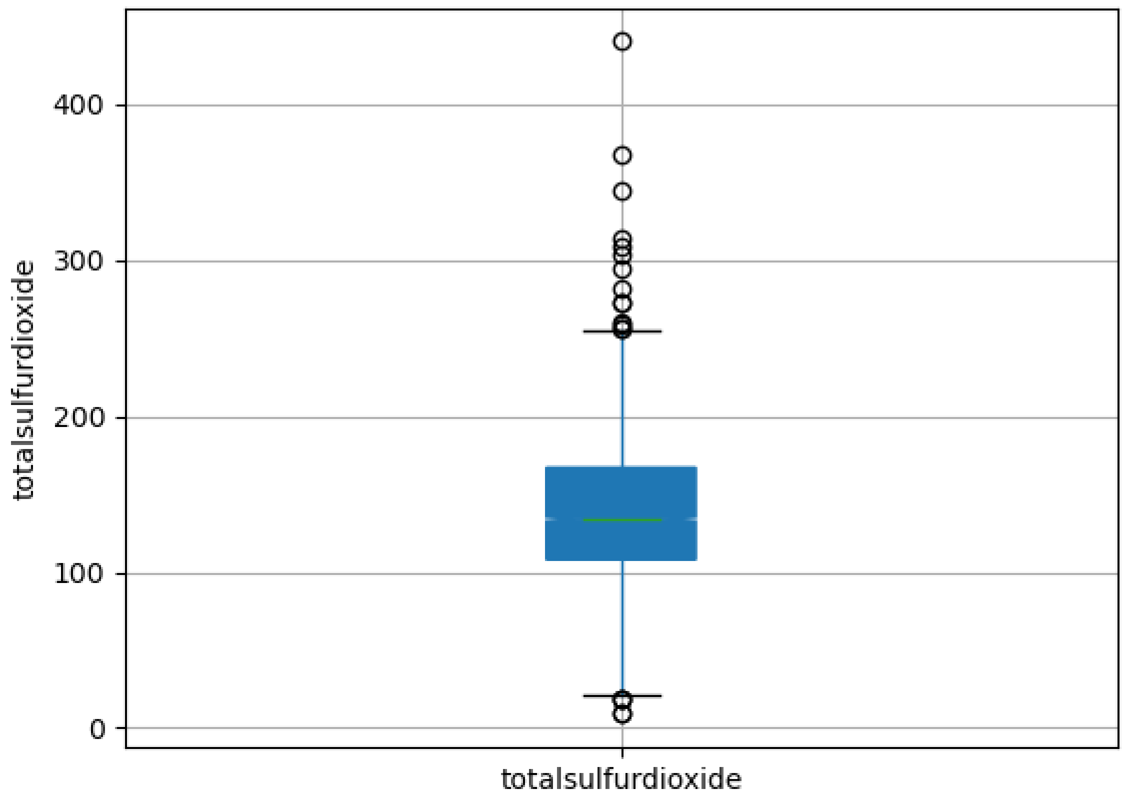
1.50
1.25
1.00
0.75
0.50
0.25
0.00
citricacid
citricacid

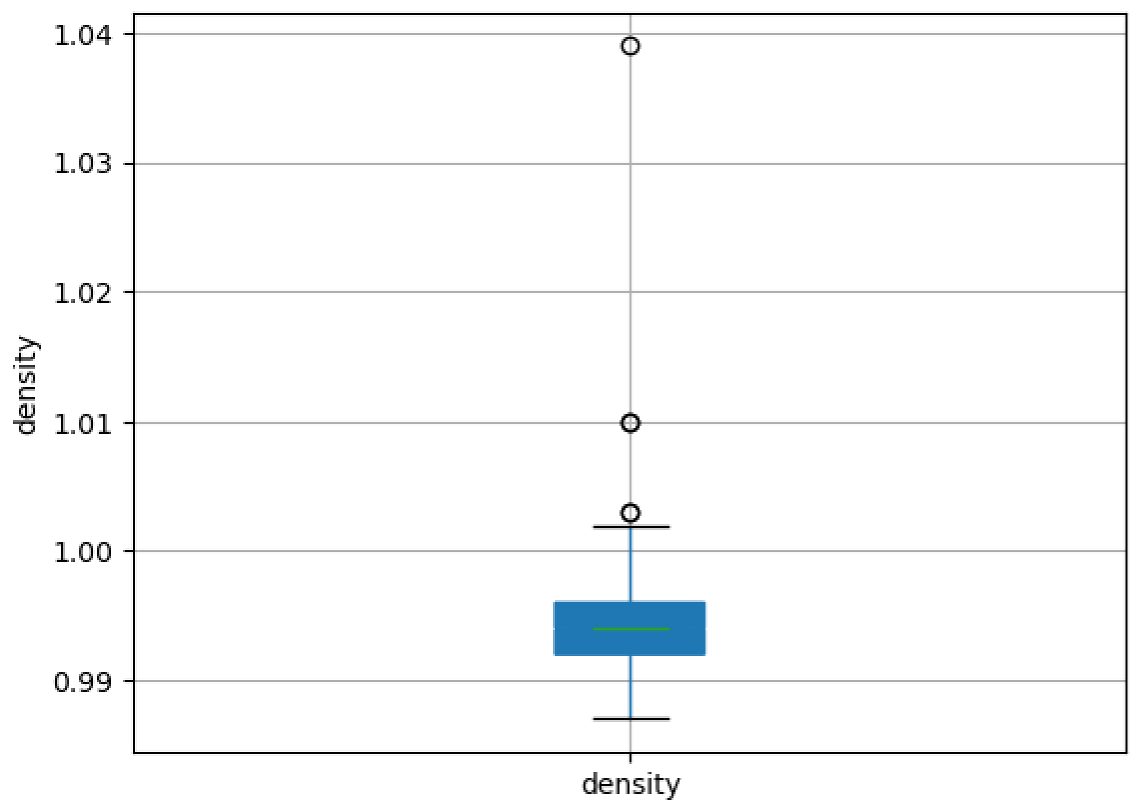


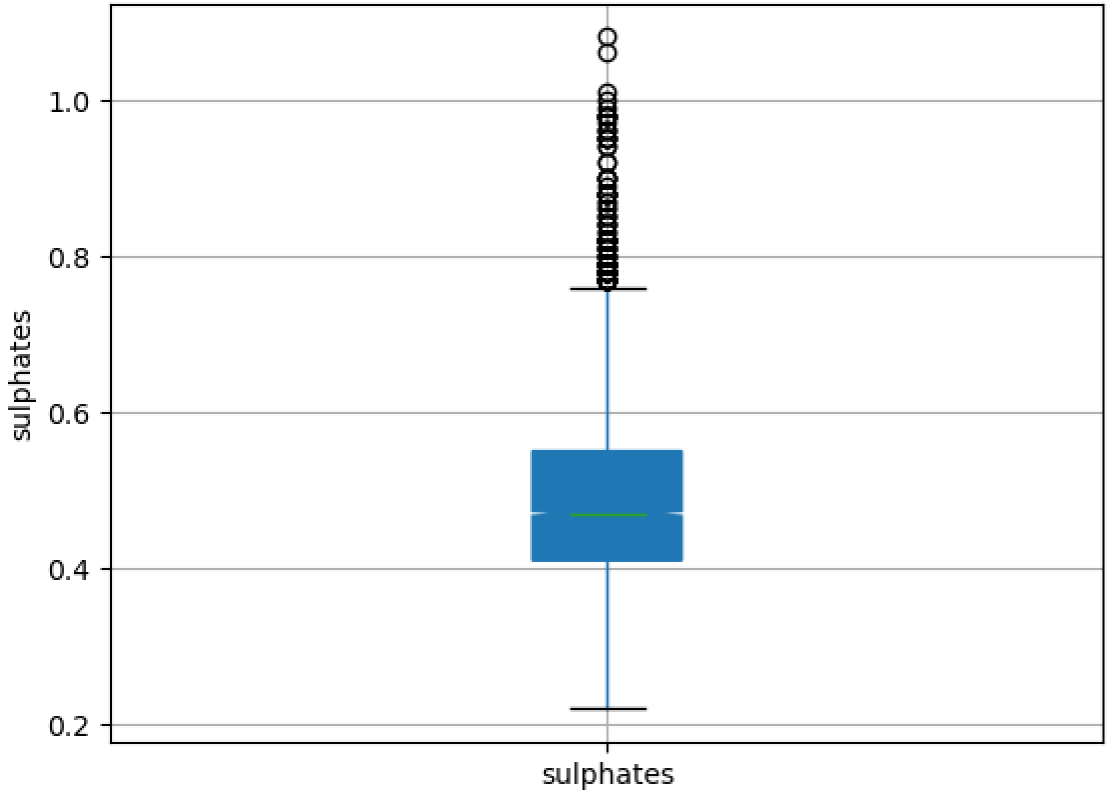
60
50
40
30
20
10
0
residualsugar
residualsugar

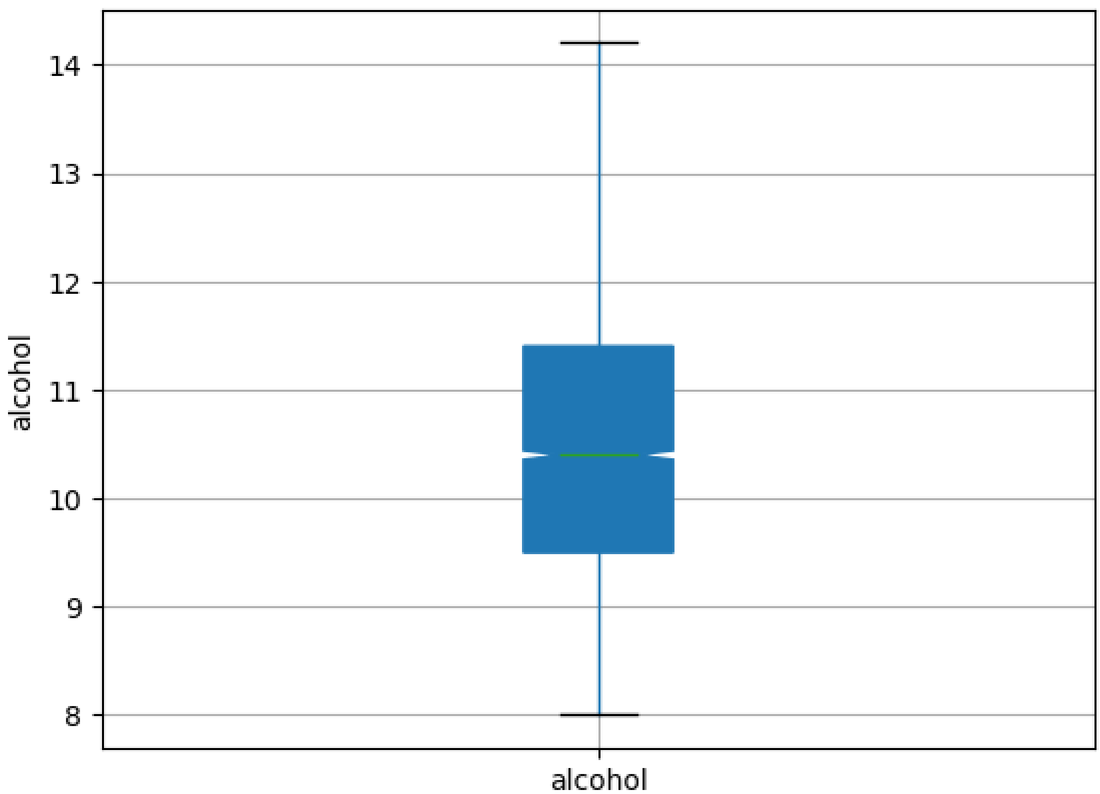

0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
chlorides
chlorides


300
250
200
150
100
50
0
freesulfurdioxide
freesulfurdioxide


totalsulfurdioxide
400
300
200
100
0
totalsulfurdioxide

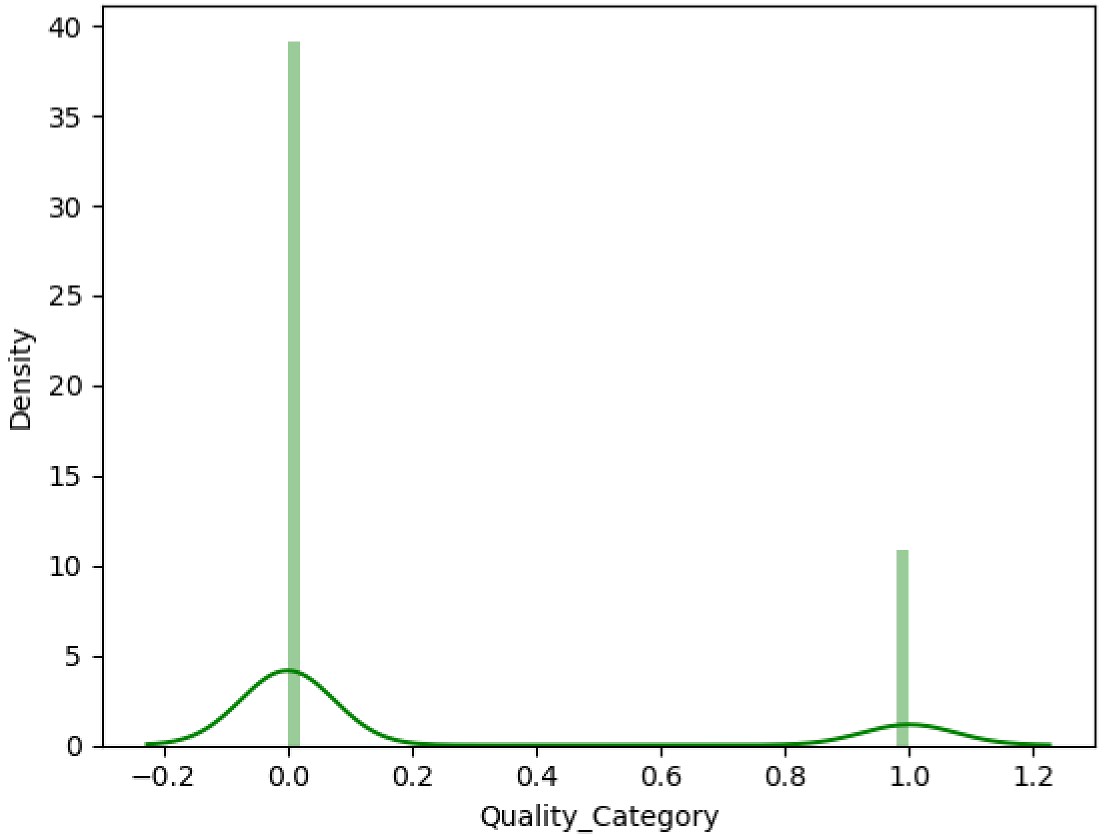

density
1.04
1.03
1.02
1.01
1.00
0.99
density



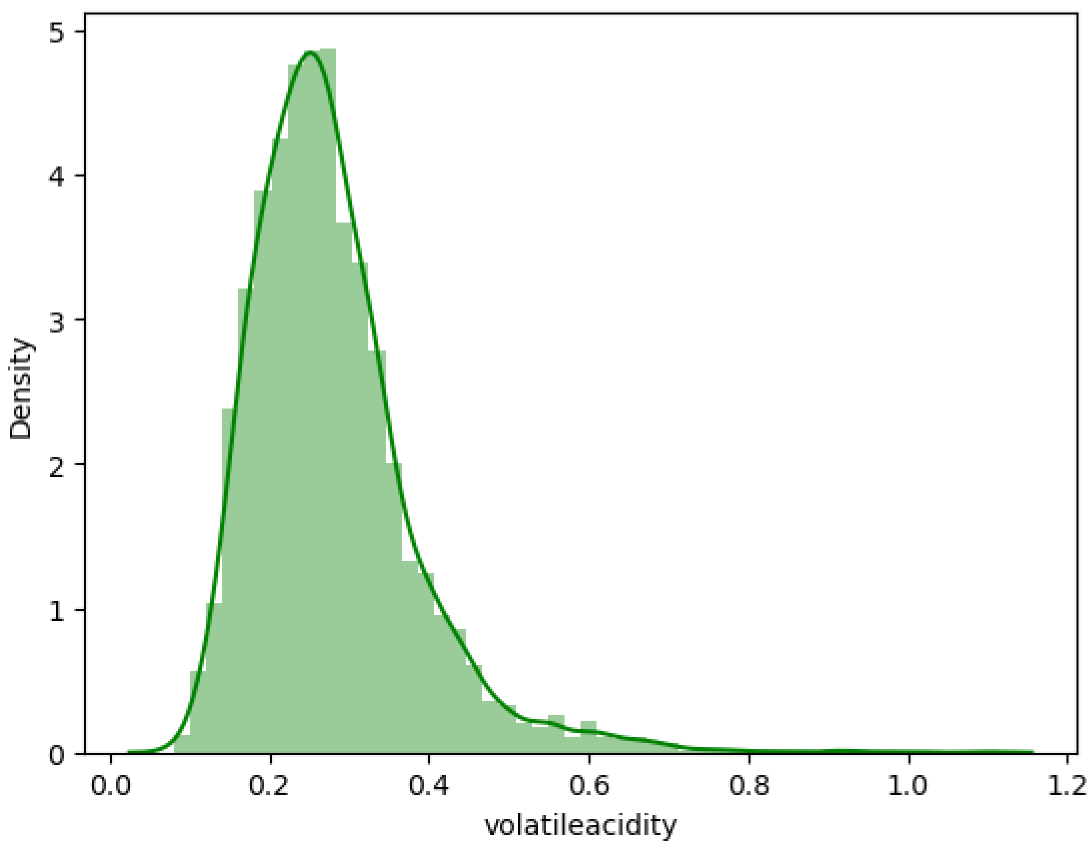


In [9]:
```
#we can see that in the numerical data has a outlier. So, we check distrubitio
n of the numerical data
```

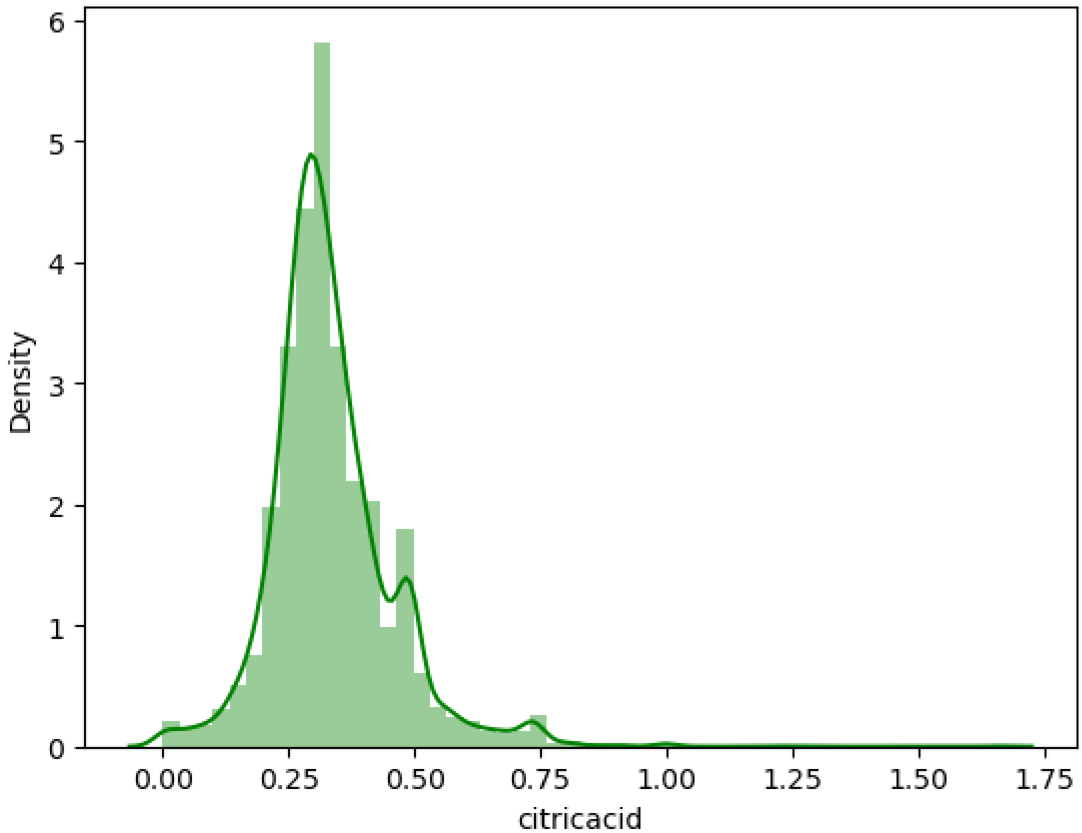
In [10]:
```
for i in df:
    sns.distplot(df[i], kde = True, color = 'green')
    plt.show()
```

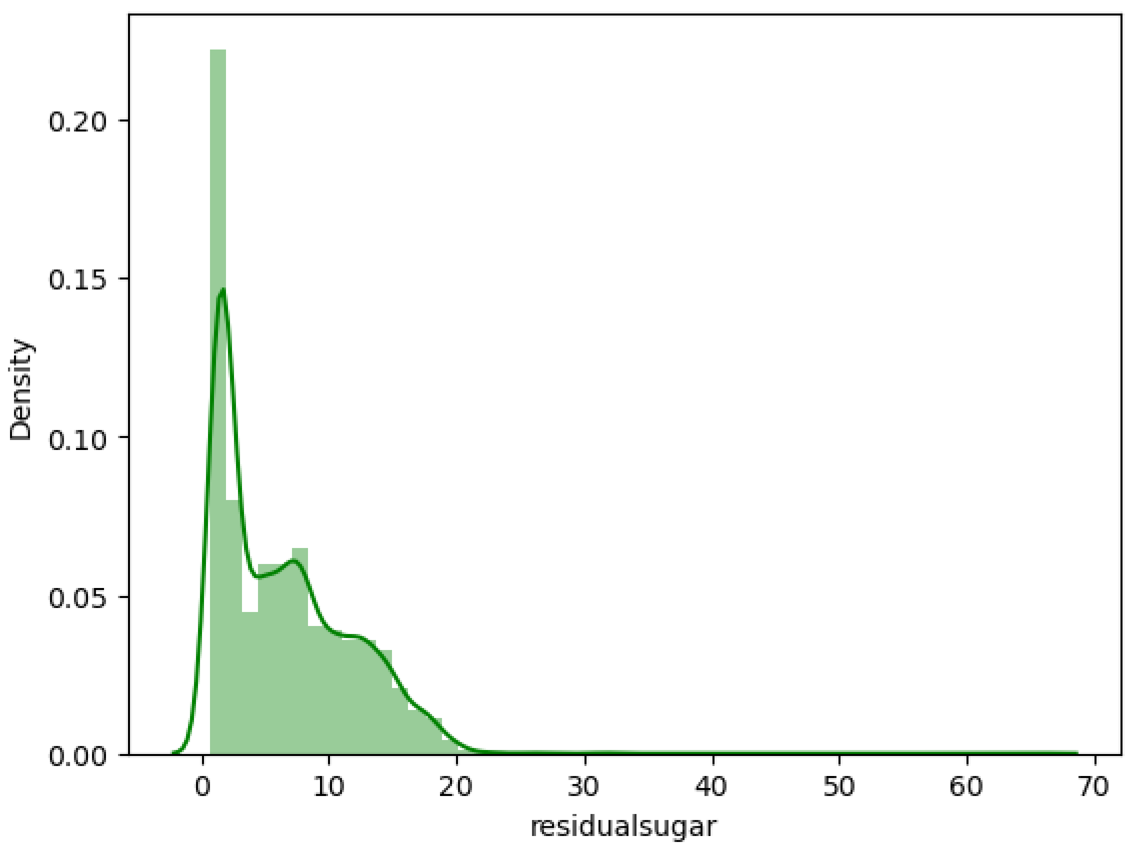


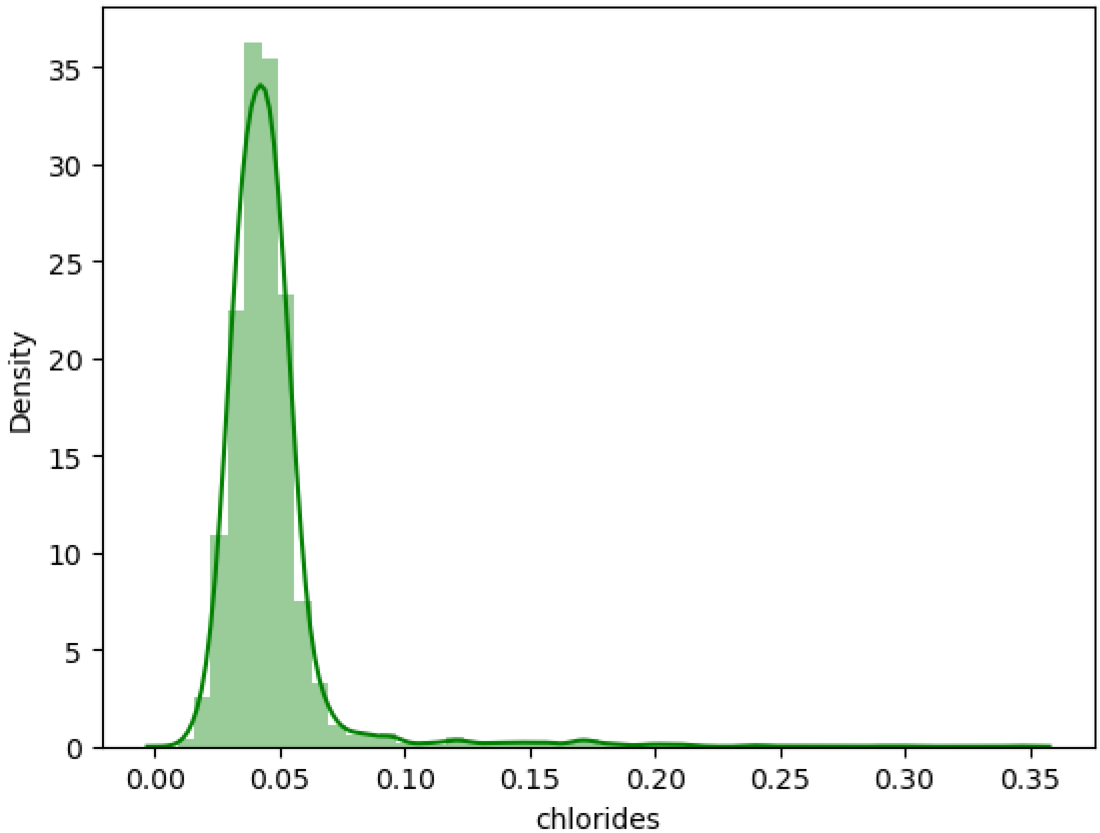
40
35
30
25
20
15
10
5
0
Density
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Quality_Category

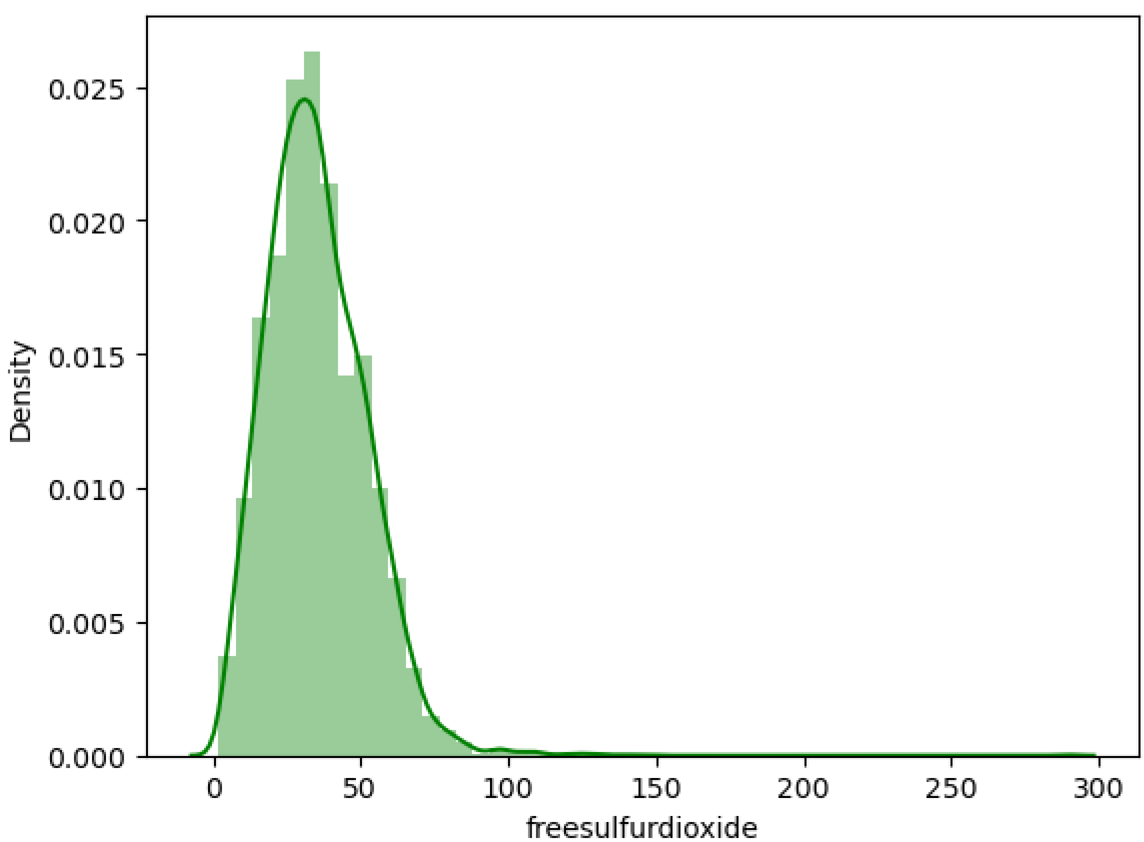


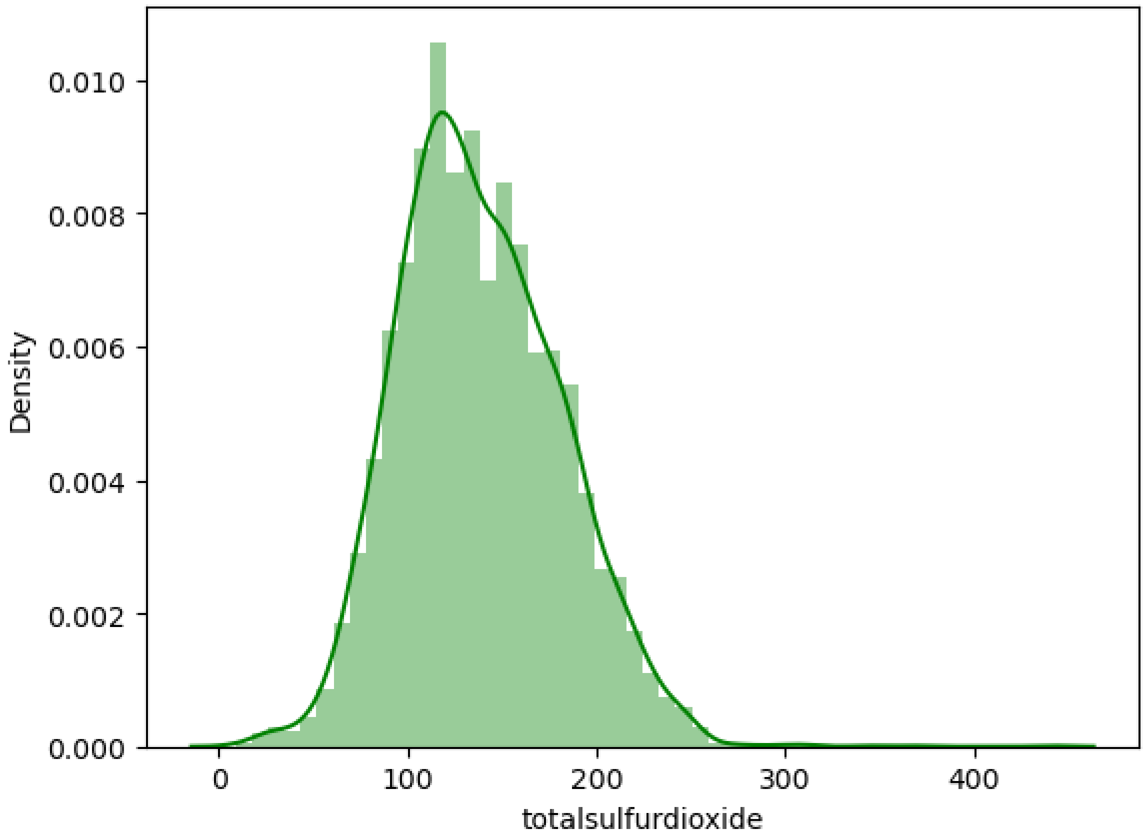
5
4
3
2
1
0
Density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
volatileacidity

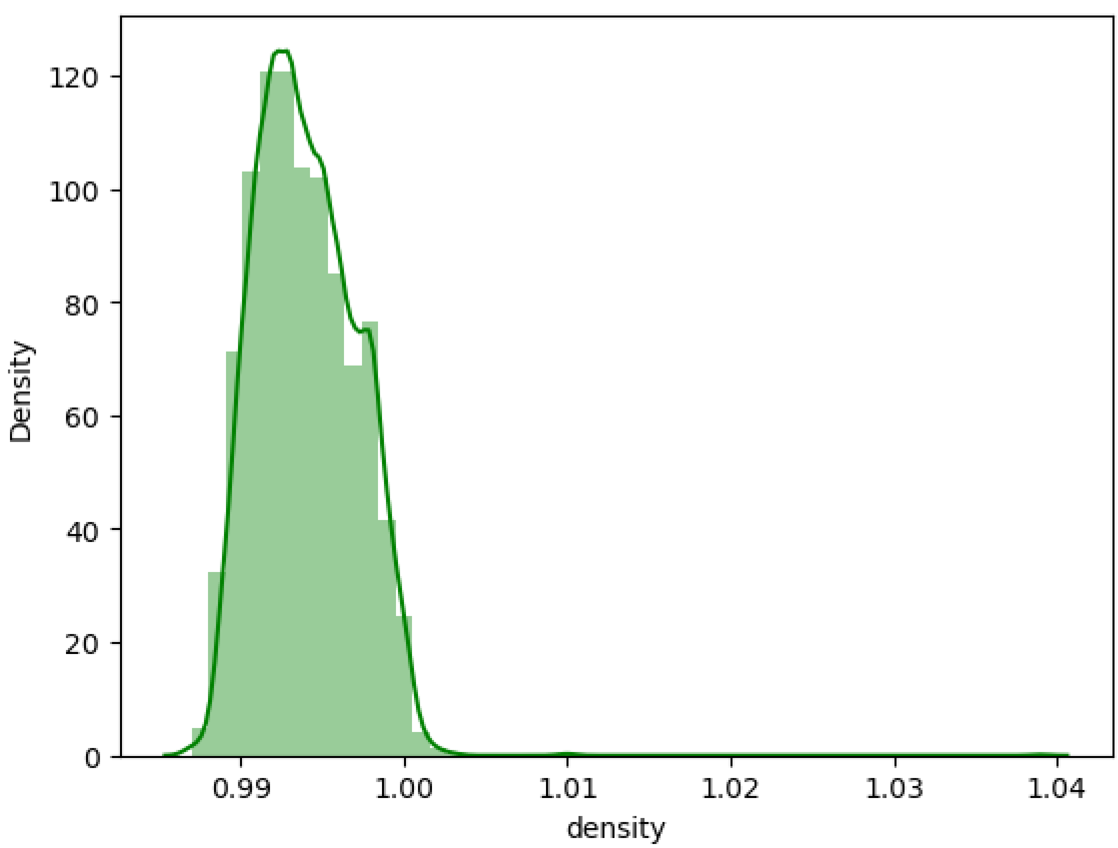

Density
citricacid
0
1
2
3
4
5
6
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75

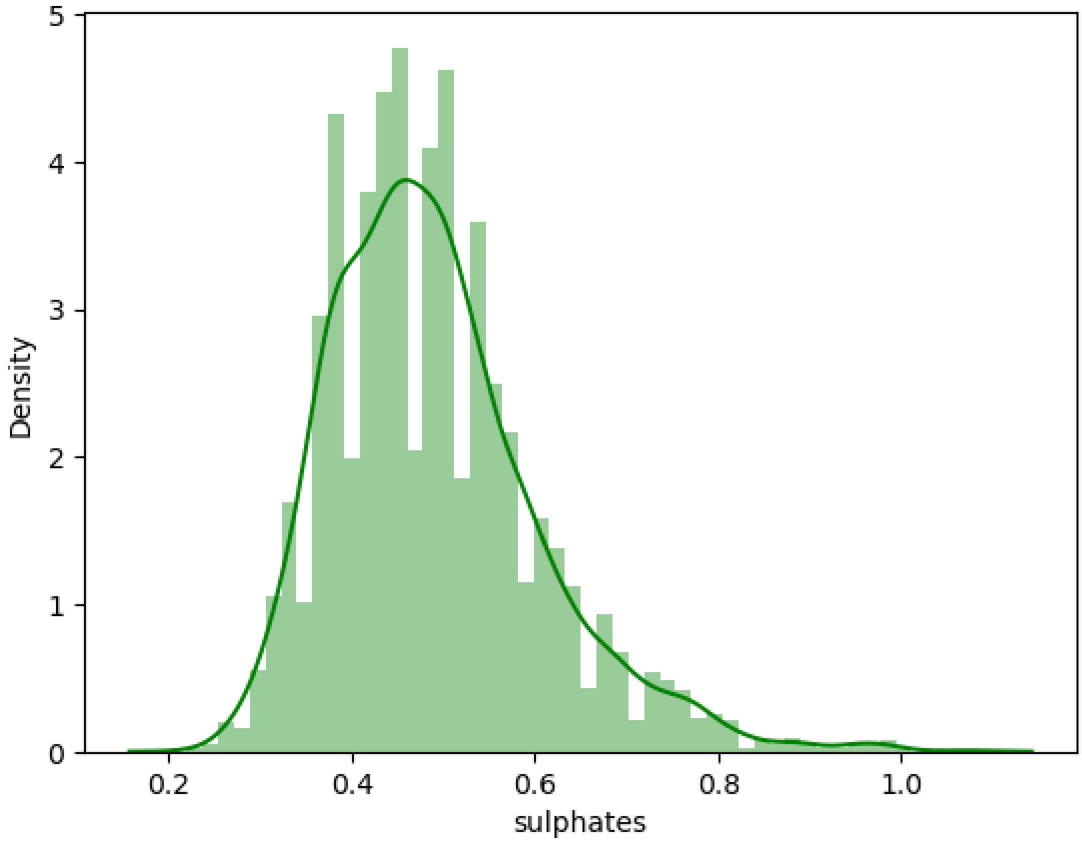

Density
residualsugar
0.00
0.05
0.10
0.15
0.20
0
10
20
30
40
50
60
70

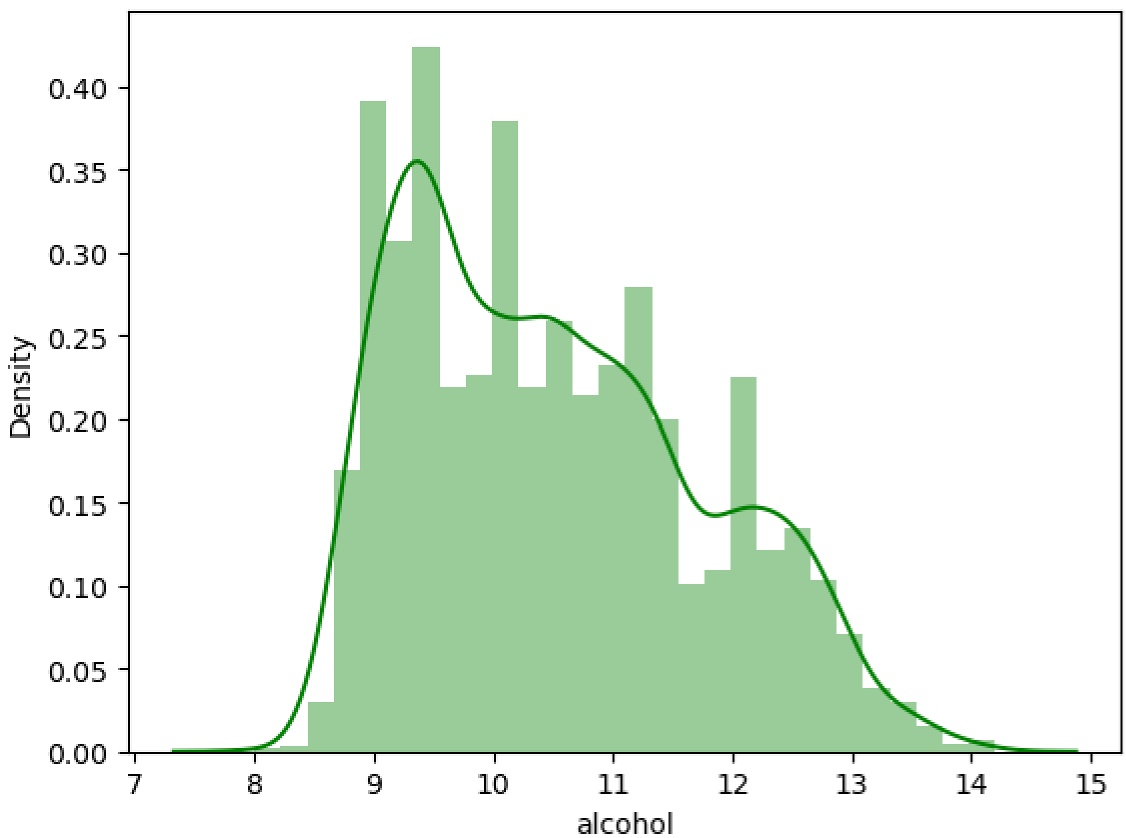

Density
chlorides
0
5
10
15
20
25
30
35
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35


Density
freesulfurdioxide
0.000
0.005
0.010
0.015
0.020
0.025
0
50
100
150
200
250
300


0.010
0.008
0.006
0.004
0.002
0.000
Density
0
100
200
300
400
totalsulfurdioxide

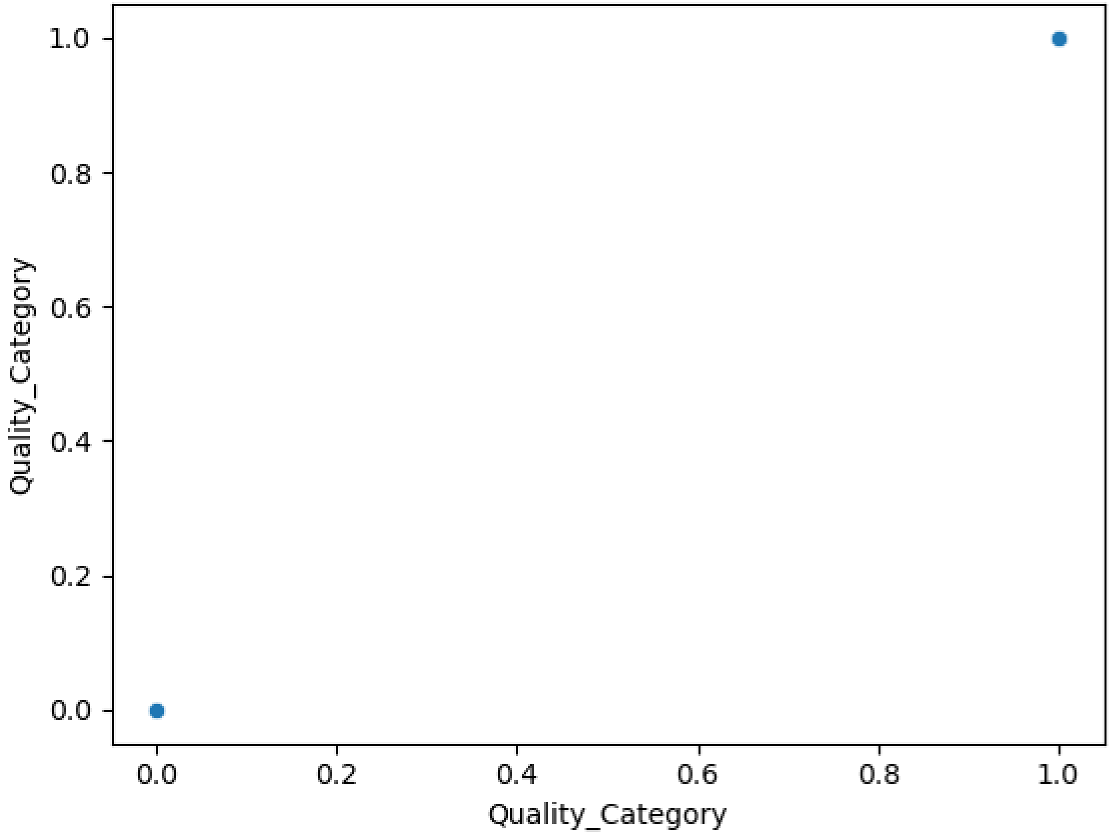

120
100
80
60
40
20
0
Density
0.99
1.00
1.01
1.02
1.03
1.04
density



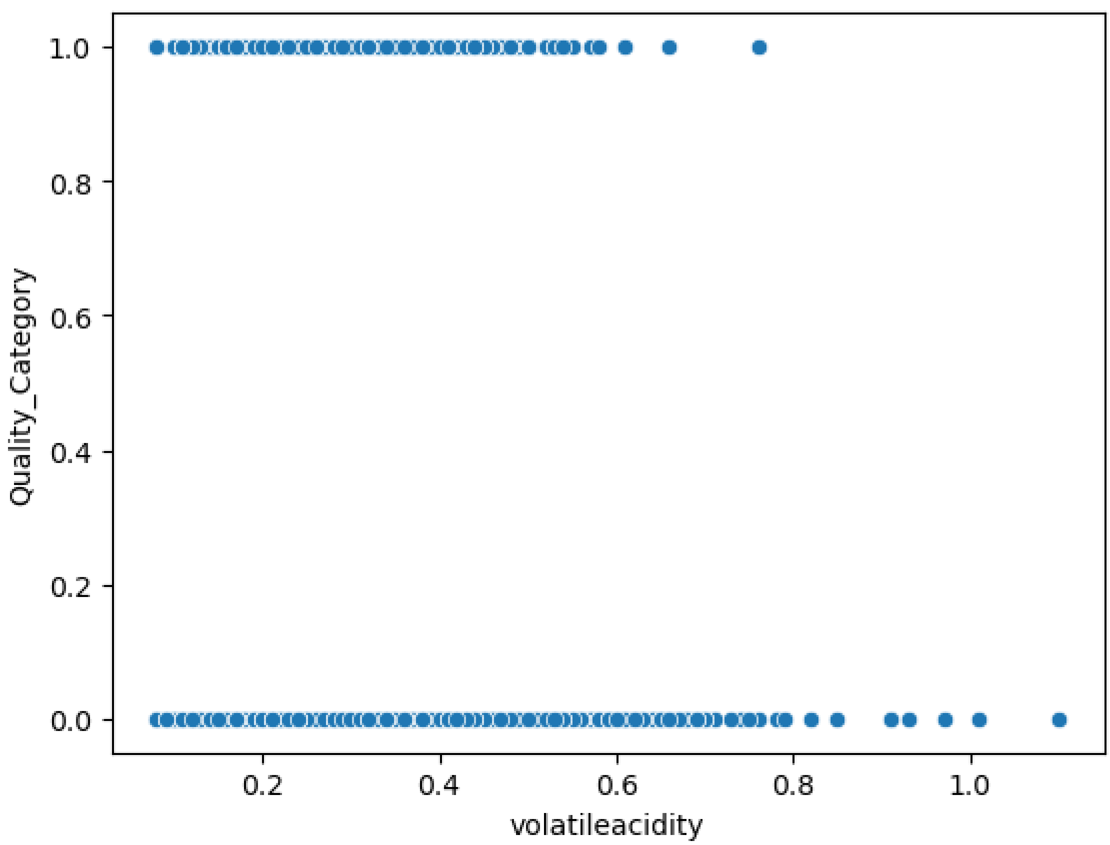


In [11]:
```
#we can see that in the dataset numerical distrubition in normal.
```

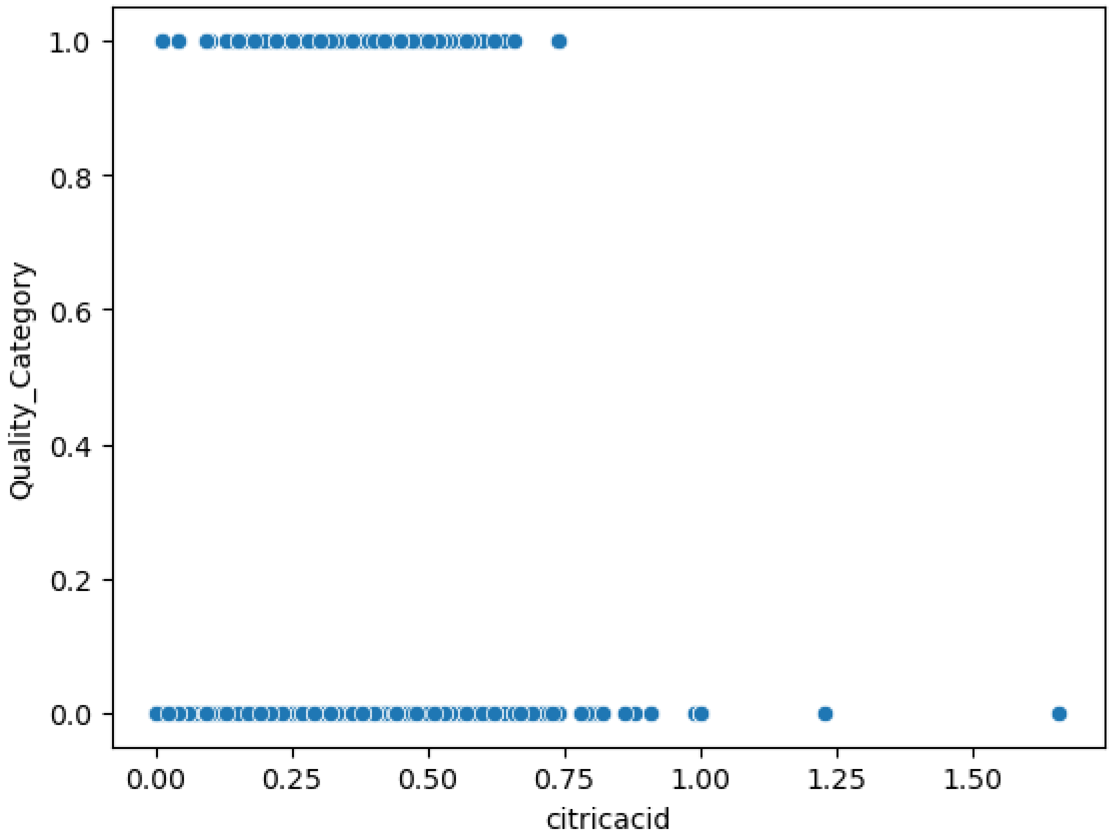
In [12]:
```
#Now, lets check the correlation of the input and output Features.
for i in df:
    sns.scatterplot(df, y=df["Quality_Category"], x=df[i])
    plt.show()
```

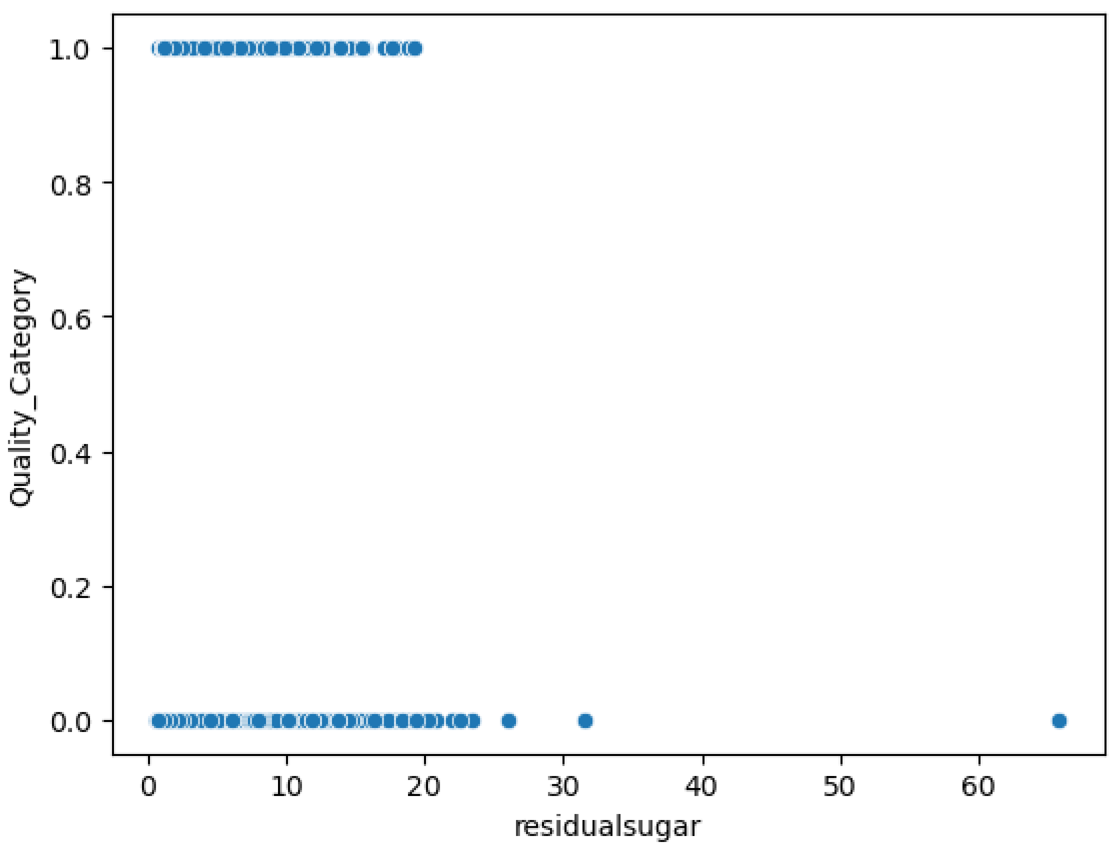


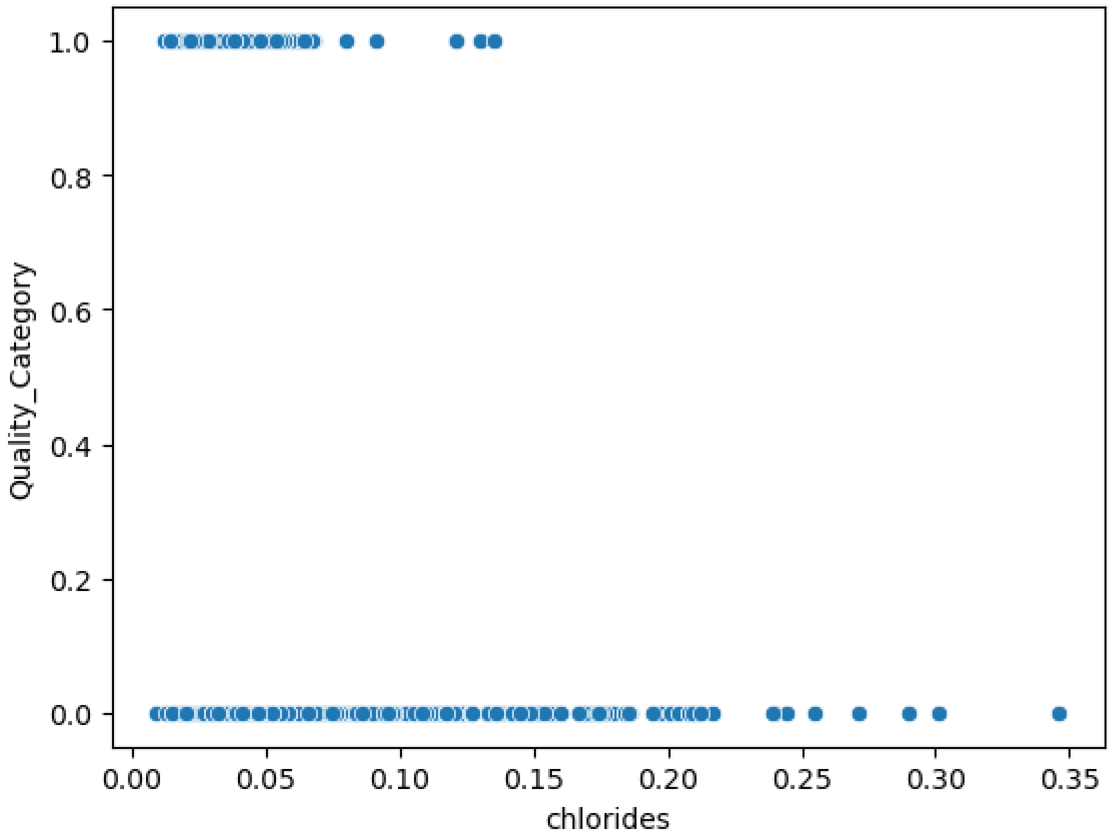
1.0
0.8
0.6
0.4
0.2
0.0
Quality_Category
0.0
0.2
0.4
0.6
0.8
1.0
Quality_Category

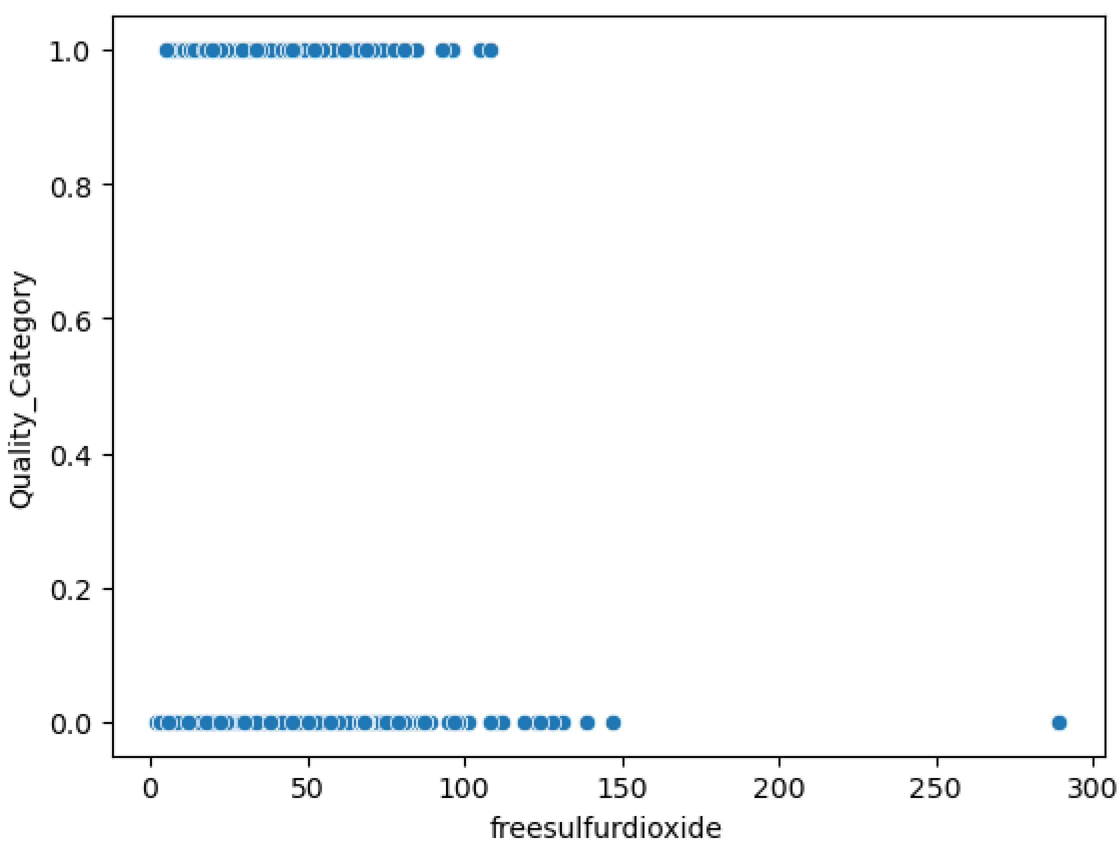


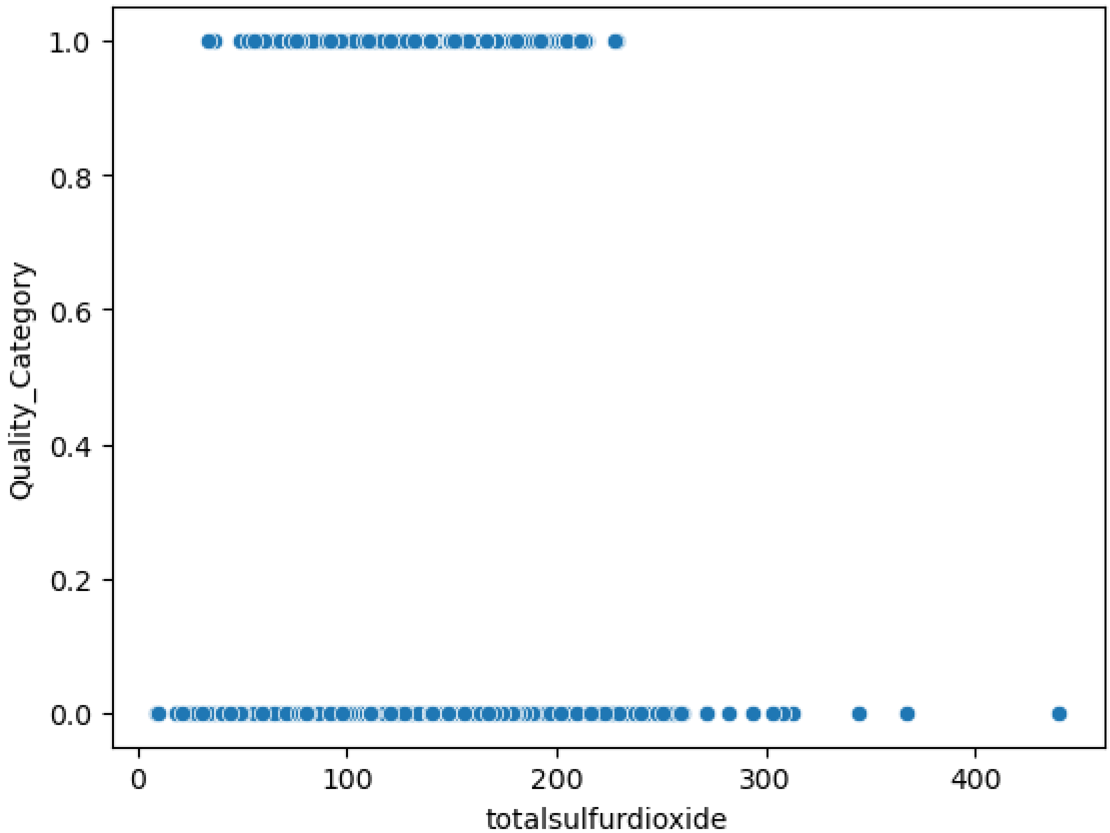
1.0
0.8
0.6
0.4
0.2
0.0
Quality_Category
0.2
0.4
0.6
0.8
1.0
volatileacidity

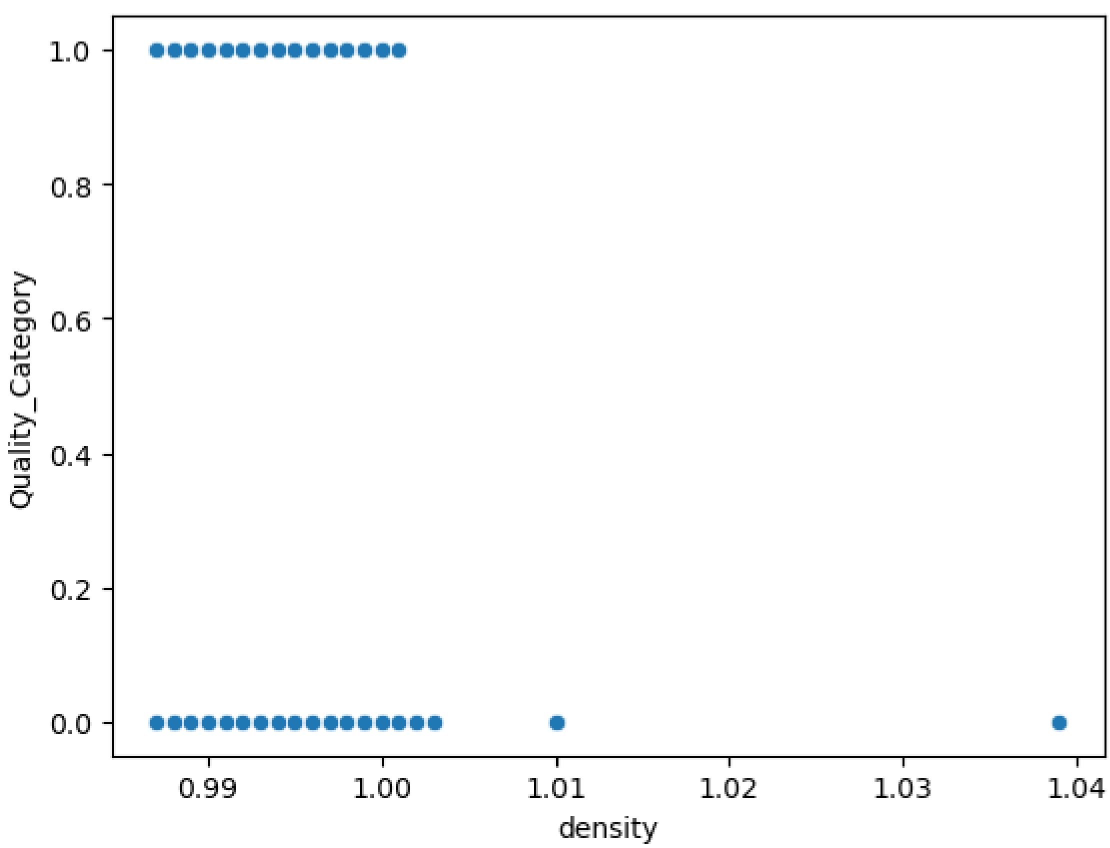

1.0
0.8
0.6
0.4
0.2
0.0
Quality_Category
0.00
0.25
0.50
0.75
1.00
1.25
1.50
citricacid

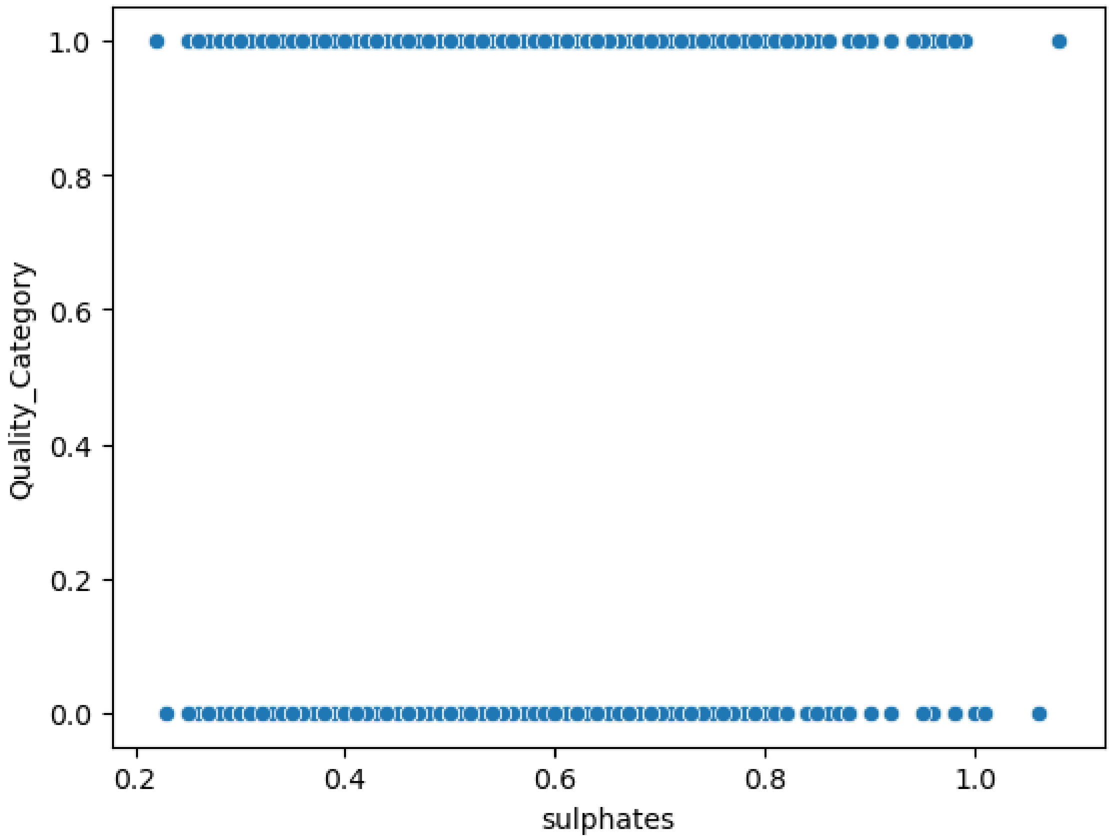

1.0
0.8
0.6
0.4
0.2
0.0
Quality_Category
0
10
20
30
40
50
60
residualsugar

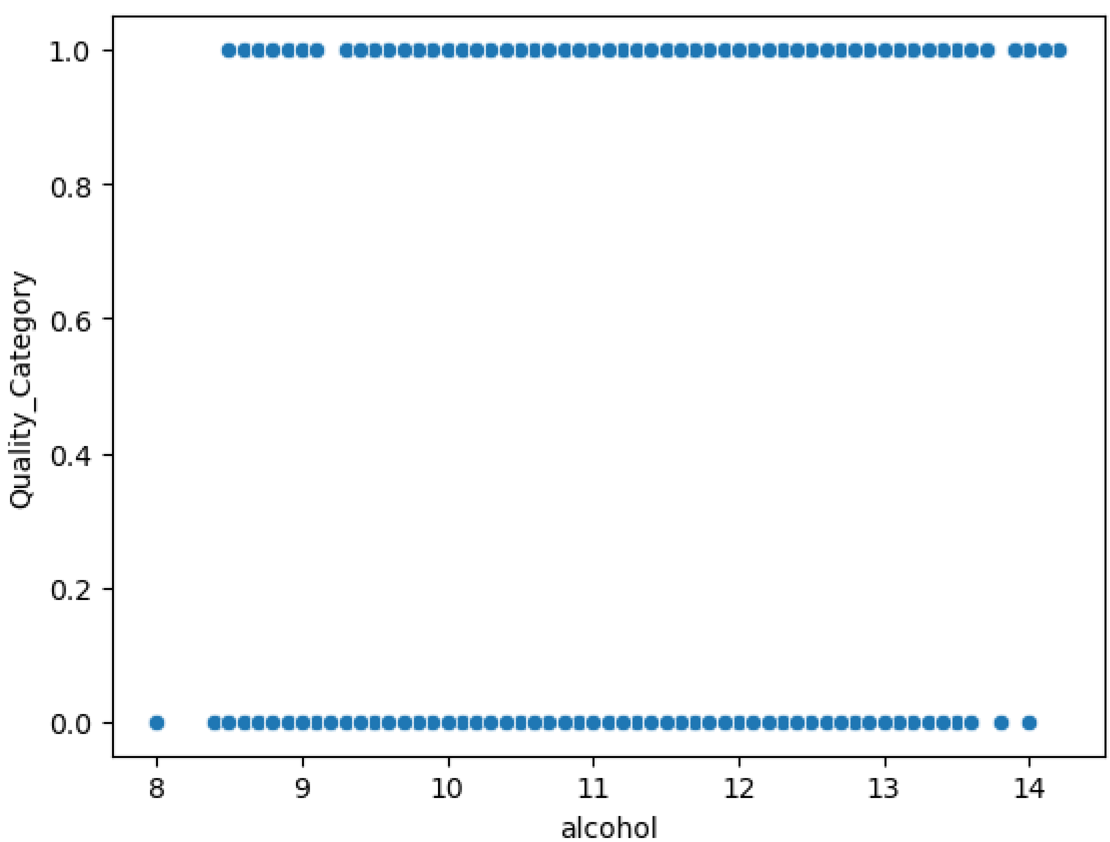

1.0
0.8
0.6
0.4
0.2
0.0
Quality_Category
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
chlorides


1.0
0.8
0.6
0.4
0.2
0.0
Quality_Category
0
50
100
150
200
250
300
freesulfurdioxide


Quality_Category
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
totalsulfurdioxide


Quality_Category
1.0
0.8
0.6
0.4
0.2
0.0
0.99
1.00
1.01
1.02
1.03
1.04
density

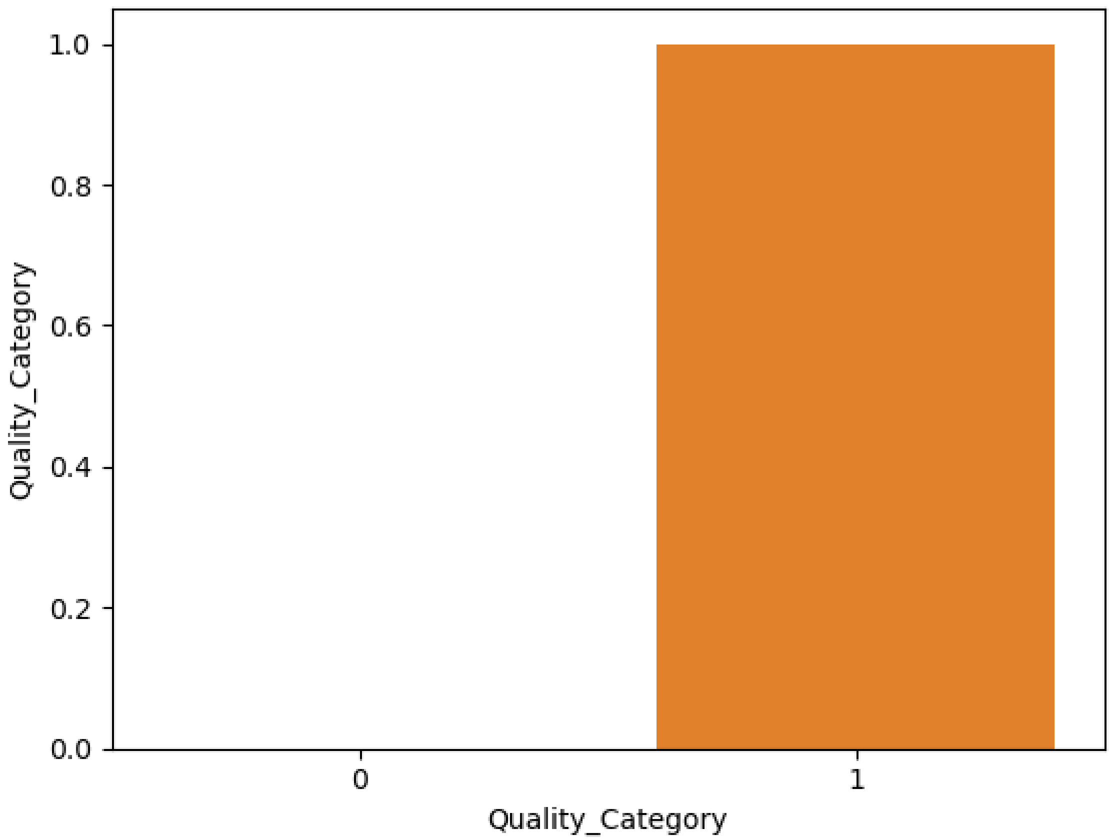

1.0
0.8
0.6
0.4
0.2
0.0
Quality_Category
0.2
0.4
0.6
0.8
1.0
sulphates

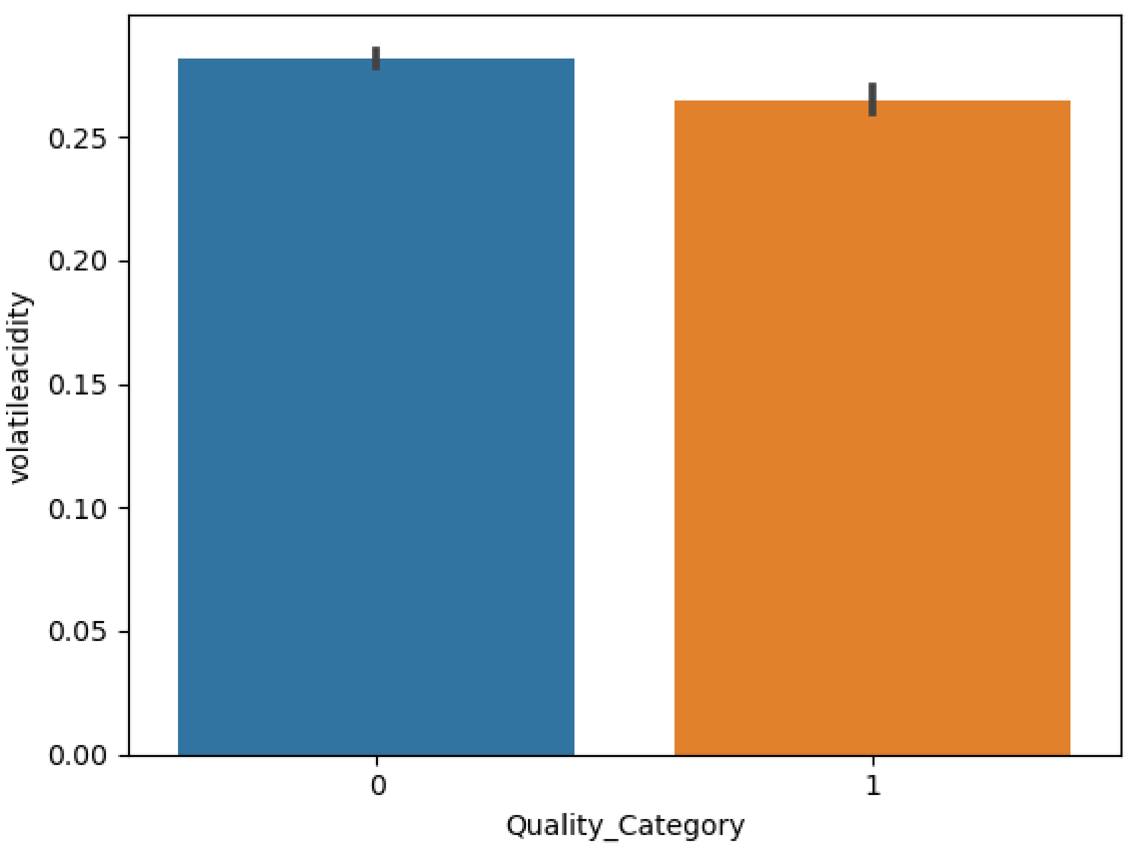

1.0
0.8
0.6
0.4
0.2
0.0
Quality_Category
8
9
10
11
12
13
14
alcohol

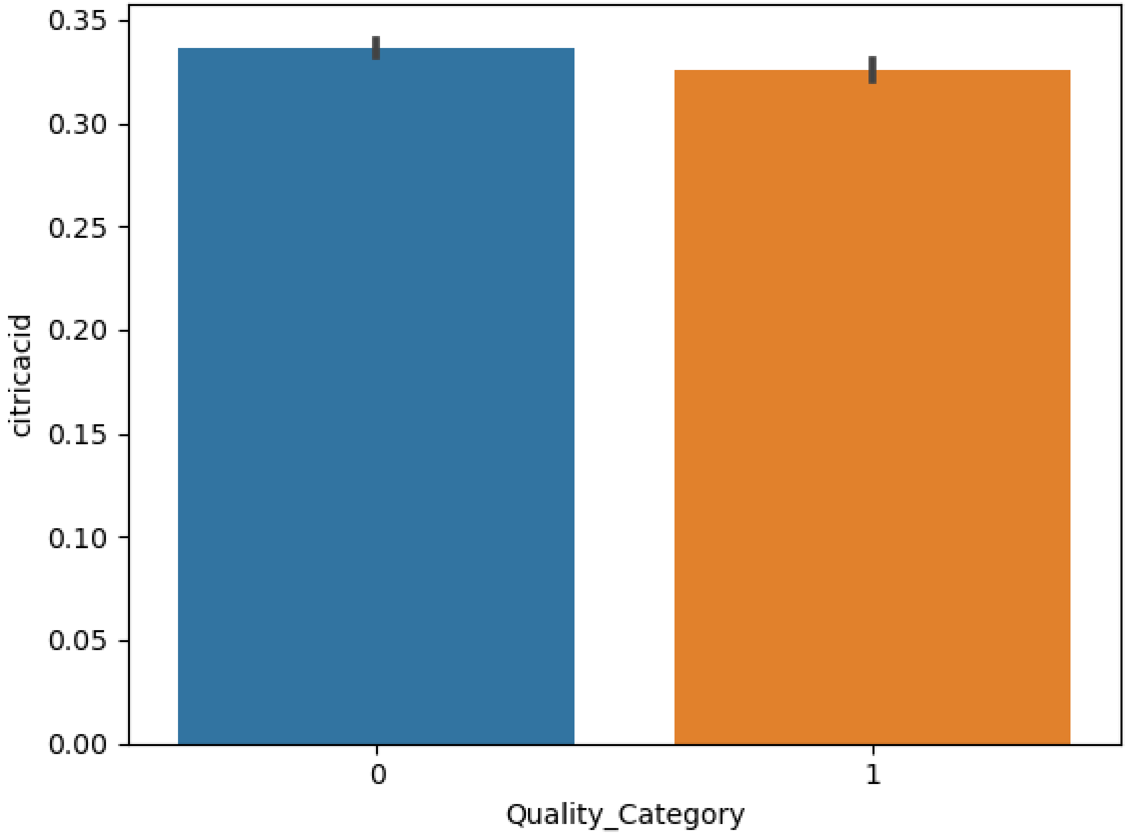
In [13]:
```
df.columns
```

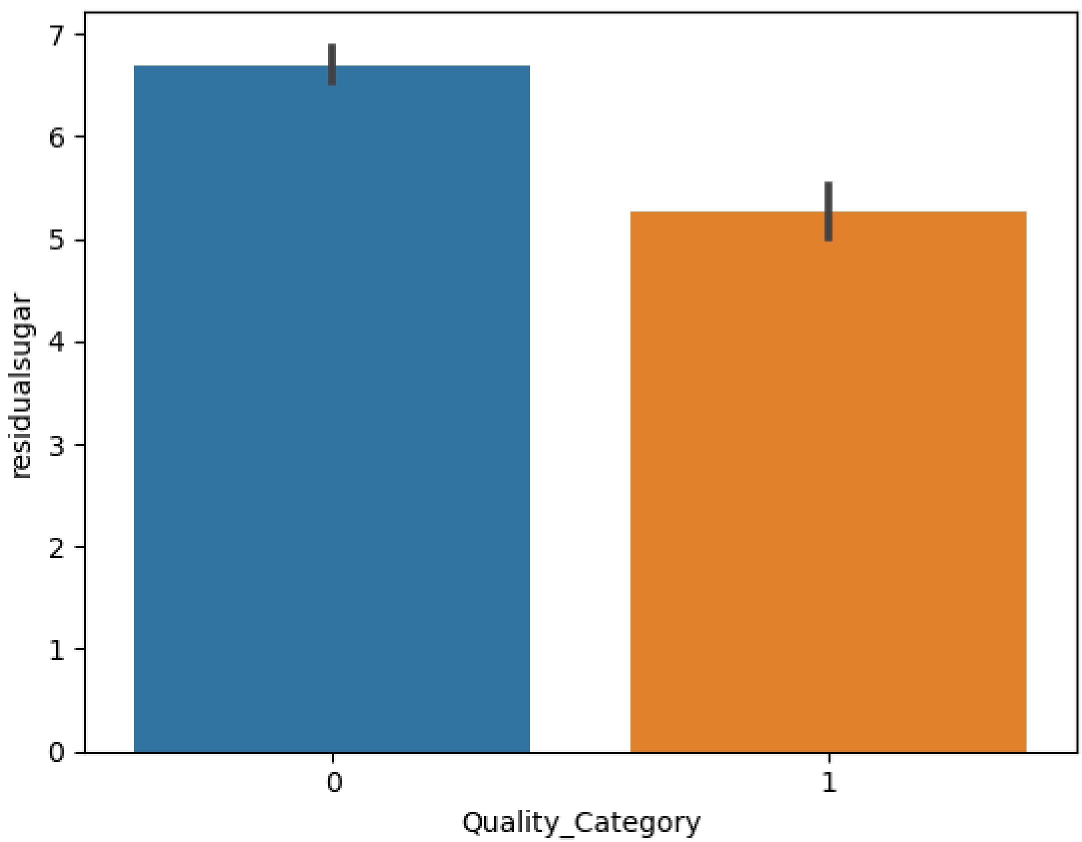
Out[13]:
```
Index(['Quality_Category', 'volatileacidity', 'citricacid', 'residualsugar',
       'chlorides', 'freesulfurdioxide', 'totalsulfurdioxide', 'density',
       'sulphates', 'alcohol'],
      dtype='object')
```

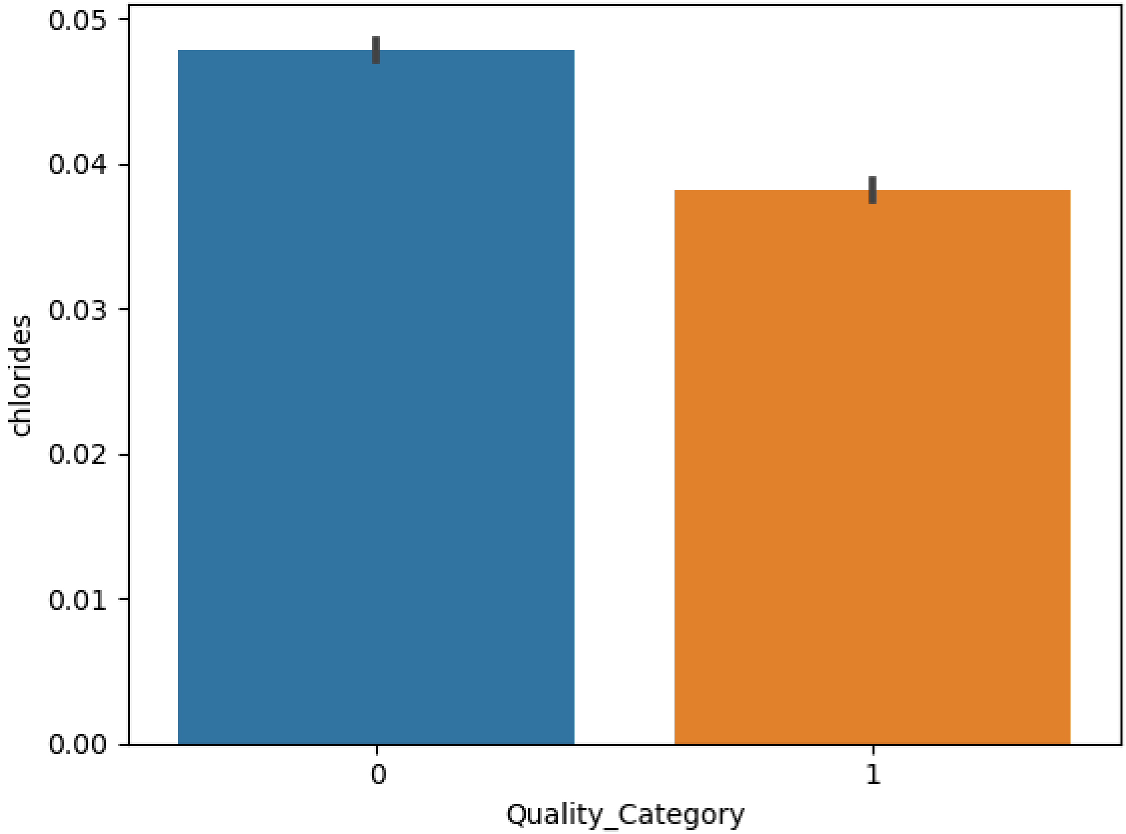
In [14]:
```
for i in df:
    sns.barplot(df, x=df["Quality_Category"], y=df[i])
    plt.show()
```

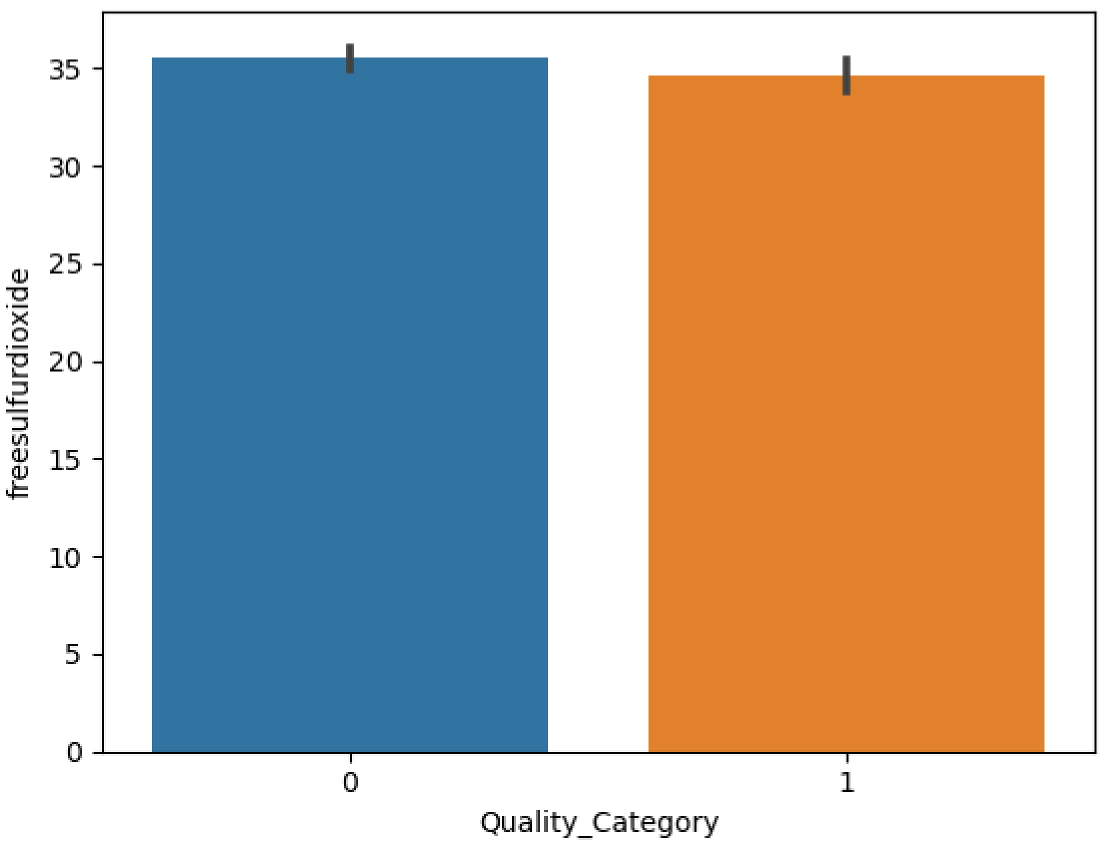

1.0
0.8
0.6
0.4
0.2
0.0
Quality_Category
0
1
Quality_Category

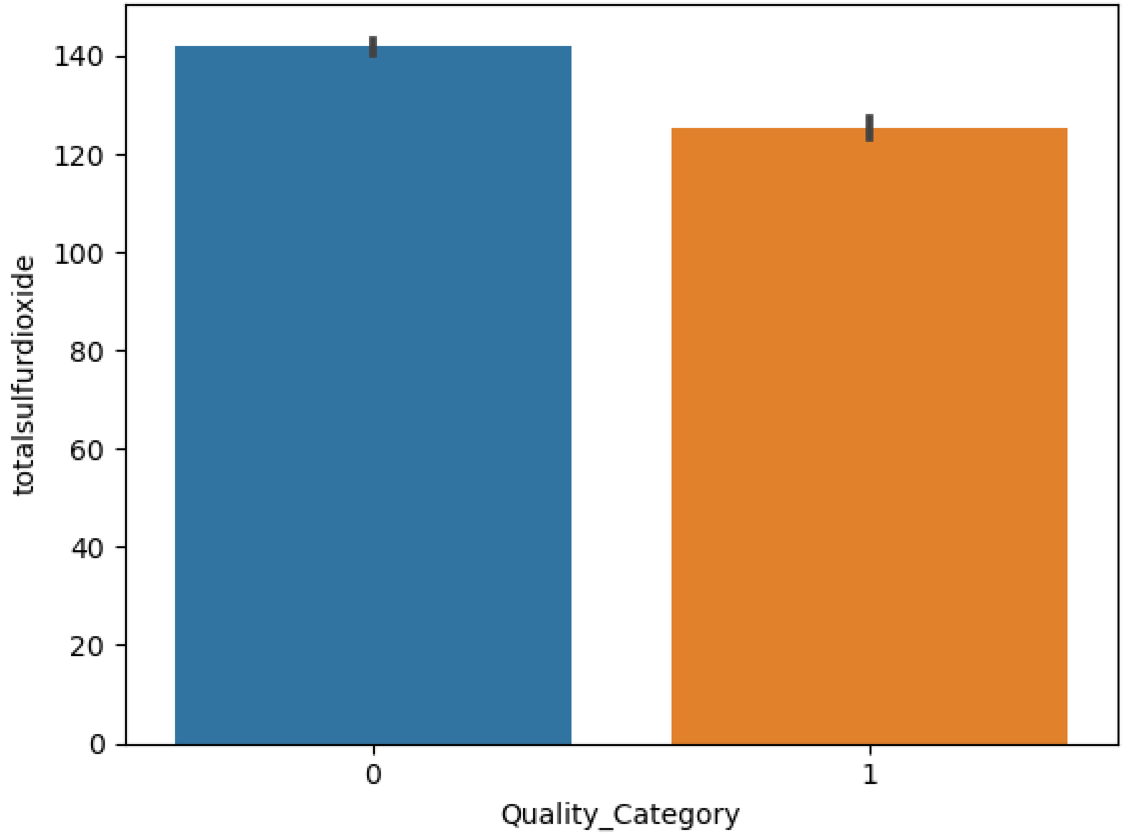

0.25
0.20
0.15
0.10
0.05
0.00
volatileacidity
0
1
Quality_Category

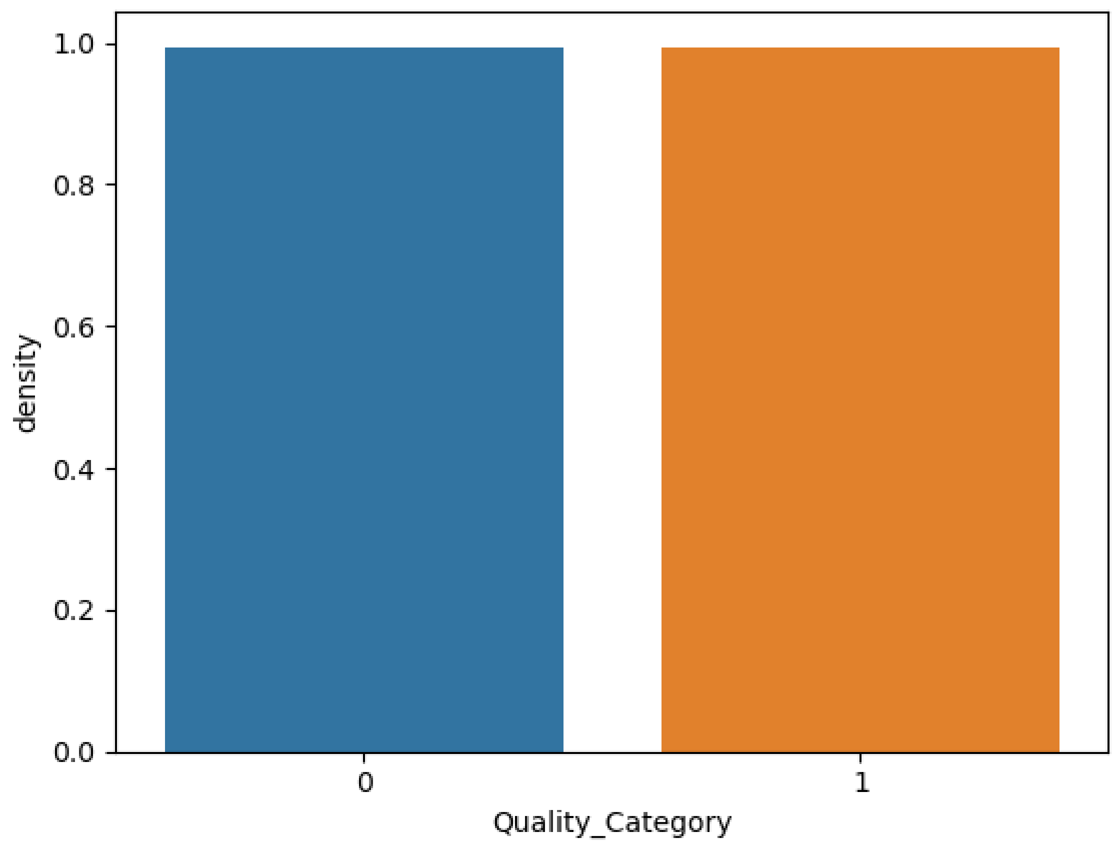

citricacid
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
1
Quality_Category

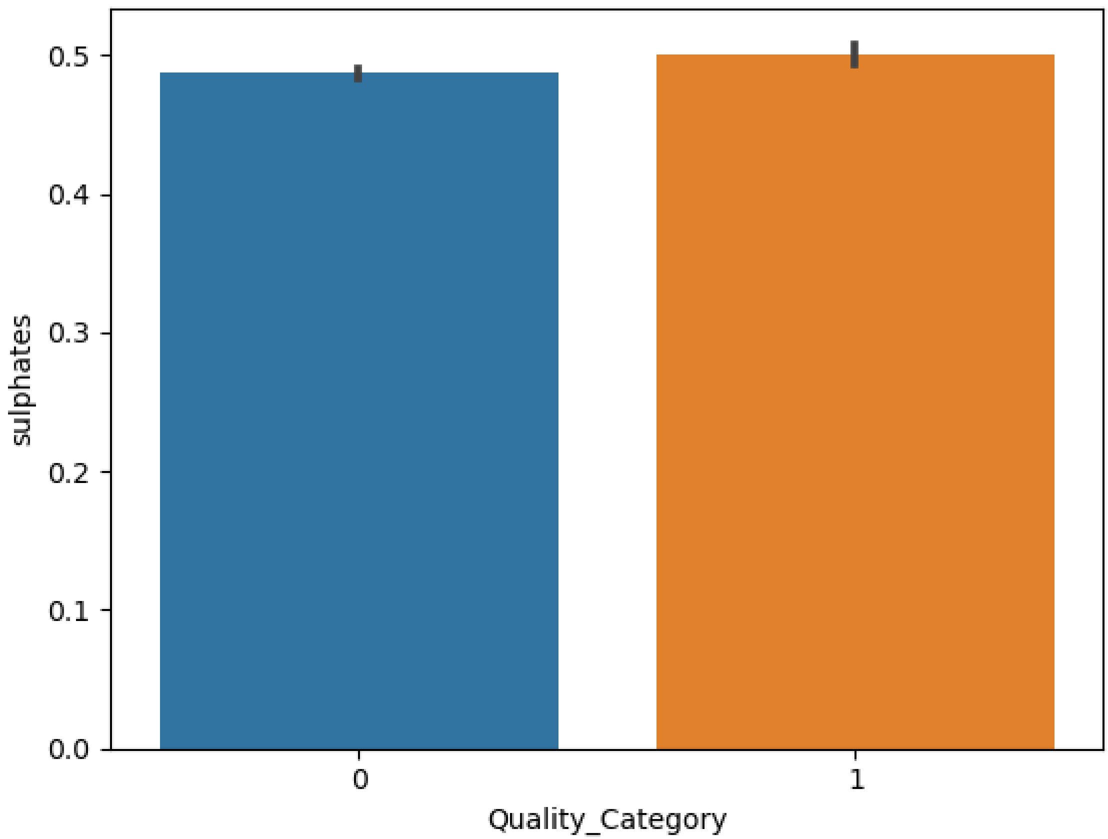

residualsugar
7
6
5
4
3
2
1
0
0
1
Quality_Category

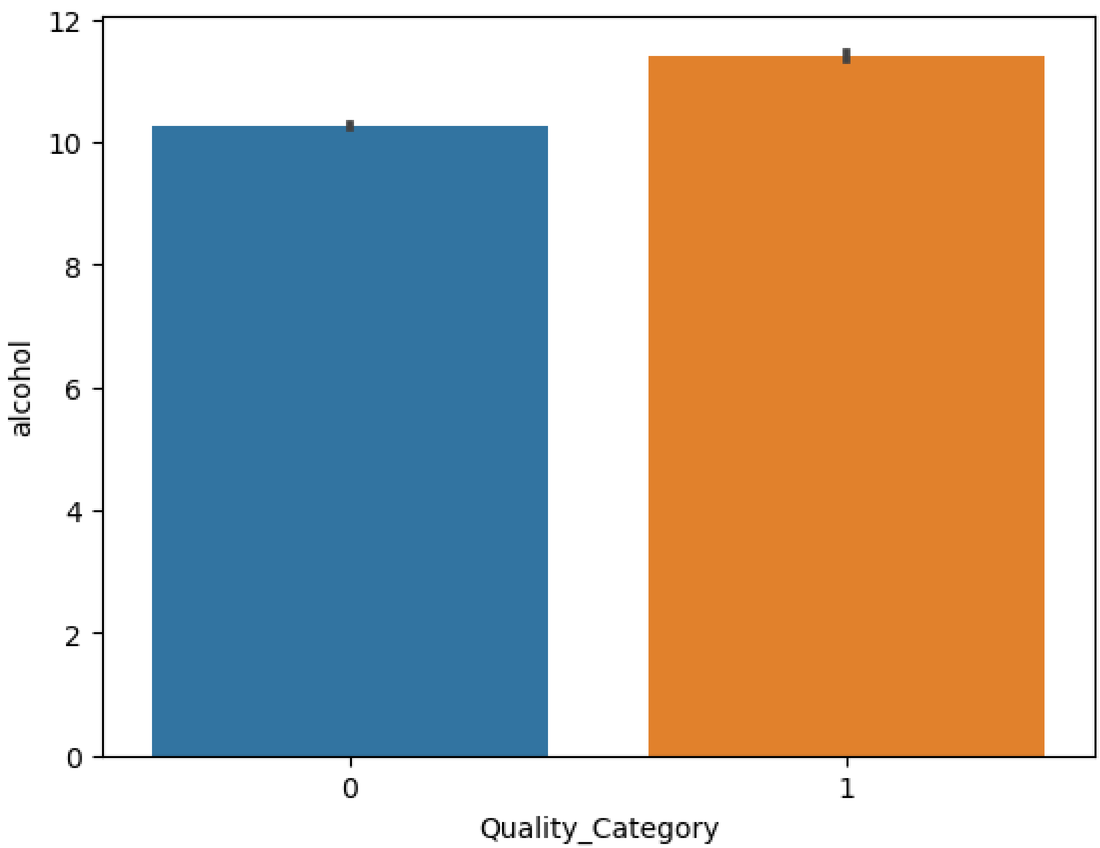

0.05
0.04
0.03
0.02
0.01
0.00
chlorides
0
1
Quality_Category


35
30
25
20
15
10
5
0
freesulfurdioxide
0
1
Quality_Category


140
120
100
80
60
40
20
0
totalsulfurdioxide
0
1
Quality_Category


1.0
0.8
0.6
0.4
0.2
0.0
density
0
1
Quality_Category





In [15]:
```
#SO, All the Features of data have important for Wine Quality.
```

In [16]:
```
#Check the duplicate
df[df.duplicated]
```

Out[16]:

	Quality_Category	volatileacidity	citricacid	residualsugar	chlorides	freesulfurdioxide	totals
5	0	0.30	0.34	1.6	0.049	14	
35	0	0.24	0.39	18.0	0.057	45	
44	0	0.31	0.26	7.4	0.069	28	
57	0	0.19	0.26	12.4	0.048	50	
59	0	0.38	0.15	4.6	0.044	25	
...	...	...	...	...	...	...	
4846	0	0.36	0.35	2.5	0.048	67	
4847	0	0.33	0.44	8.9	0.055	52	
4852	0	0.23	0.39	13.7	0.058	26	
4876	0	0.34	0.40	8.1	0.046	68	
4885	0	0.24	0.27	11.8	0.030	34	

955 rows × 10 columns

Here, in this dataset have almost 955 row of duplicte but we can not delete because every red wine have their own different quality need that's why we don't drop duplicate.

Seperate data in X and Y as well as Split data into train and Test

In [17]:
```
# I am using a df1 data which was copy of the original data set.
x = df1.drop(["Quality_Category"], axis=1)
y = df1["Quality_Category"]
```

In [18]:
```
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(x,y, random_state=50, test
_size=0.2, stratify=y)
```

In [19]:
```
train_x
```

Out[19]:

	volatileacidity	citricacid	residualsugar	chlorides	freesulfurdioxide	totalsulfurdioxide	dens
3073	0.25	0.38	7.9	0.045	54	208	0.9
2483	0.27	0.27	1.7	0.034	25	122	0.9
2155	0.24	0.37	1.8	0.031	6	61	0.9
4235	0.28	0.36	1.8	0.041	38	90	0.9
635	0.34	0.14	5.8	0.046	49	197	0.9
...	...	...	...	...	...	...	
3851	0.18	0.29	4.6	0.032	68	137	0.9
2015	0.32	0.22	16.7	0.046	38	133	0.9
3337	0.17	0.27	4.6	0.050	23	98	0.9
1978	0.20	0.30	14.2	0.056	53	213	0.9
1475	0.16	0.49	2.0	0.056	20	124	0.9

3915 rows × 9 columns

In [20]:
```
#create reset index
train_x.reset_index(inplace=True, drop=True)
test_x.reset_index(inplace=True, drop=True)

train_y.reset_index(inplace=True, drop=True)
test_y.reset_index(inplace=True, drop=True)
```

In [21]:
```
train_x
```

Out[21]:

	volatileacidity	citricacid	residualsugar	chlorides	freesulfurdioxide	totalsulfurdioxide	dens
0	0.25	0.38	7.9	0.045	54	208	0.9
1	0.27	0.27	1.7	0.034	25	122	0.9
2	0.24	0.37	1.8	0.031	6	61	0.9
3	0.28	0.36	1.8	0.041	38	90	0.9
4	0.34	0.14	5.8	0.046	49	197	0.9
...	...	...	...	...	...	...	
3910	0.18	0.29	4.6	0.032	68	137	0.9
3911	0.32	0.22	16.7	0.046	38	133	0.9
3912	0.17	0.27	4.6	0.050	23	98	0.9
3913	0.20	0.30	14.2	0.056	53	213	0.9
3914	0.16	0.49	2.0	0.056	20	124	0.9

3915 rows × 9 columns

In [22]:
```
train_y
```

Out[22]:
```
0       0
1       0
2       0
3       1
4       0
       ..
3910    0
3911    0
3912    0
3913    1
3914    0
Name: Quality_Category, Length: 3915, dtype: int64
```

Scaling Using Robustscaler

In [23]:
```
from sklearn.preprocessing import RobustScaler, MinMaxScaler, StandardScaler
scaler = MinMaxScaler()
scaler.fit(train_x)
```

Out[23]:
```
▾ MinMaxScaler
MinMaxScaler()
```

In [24]:
```
train_x = pd.DataFrame(scaler.transform(train_x), columns=train_x.columns)
test_x = pd.DataFrame(scaler.transform(test_x), columns=test_x.columns)
```

In [25]:
```
train_x
```

Out[25]:

	volatileacidity	citricacid	residualsugar	chlorides	freesulfurdioxide	totalsulfurdioxide	der
0	0.166667	0.228916	0.111963	0.106825	0.181185	0.461717	0.17:
1	0.186275	0.162651	0.016871	0.074184	0.080139	0.262181	0.07(
2	0.156863	0.222892	0.018405	0.065282	0.013937	0.120650	0.05:
3	0.196078	0.216867	0.018405	0.094955	0.125436	0.187935	0.05:
4	0.254902	0.084337	0.079755	0.109792	0.163763	0.436195	0.13
...	...	...	...	...	...	...	
3910	0.098039	0.174699	0.061350	0.068249	0.229965	0.296984	0.09(
3911	0.235294	0.132530	0.246933	0.109792	0.125436	0.287703	0.21
3912	0.088235	0.162651	0.061350	0.121662	0.073171	0.206497	0.13
3913	0.117647	0.180723	0.208589	0.139466	0.177700	0.473318	0.23(
3914	0.078431	0.295181	0.021472	0.139466	0.062718	0.266821	0.15:

3915 rows × 9 columns

In [26]:
```
train_x.isnull().sum().sum()
```

Out[26]: 0

In [27]:
```
test_x.isnull().sum().sum()
```

Out[27]: 0

Model Building And Evaluation

In [28]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
import xgboost as Xgb
```

In [29]:
```
from sklearn.metrics import classification_report, accuracy_score, precision_s
core, recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
```

In [30]:
```
#LOGISTIC REGRESSION
log_model = LogisticRegression(random_state=50)
log_model.fit(train_x, train_y)
pred_log = log_model.predict(test_x)
print(classification_report(test_y, pred_log))
```

```
              precision    recall  f1-score   support

           0       0.82      0.95      0.88       767
           1       0.55      0.22      0.32       212

    accuracy                           0.79       979
   macro avg       0.68      0.59      0.60       979
weighted avg       0.76      0.79      0.76       979
```

In [31]:
```
log_model.score(train_x, train_y)
```

Out[31]: 0.8068965517241379

In [32]:
```
log_model.score(test_x, test_y)
```

Out[32]: 0.7916241062308478

In [33]:
```
#KNEARASTNEIGHBORS CLASSIFIER
knn_model = KNeighborsClassifier(n_neighbors=10)
knn_model.fit(train_x, train_y)
pred_knn = knn_model.predict(test_x)
print(classification_report(test_y, pred_knn))
```

```
              precision    recall  f1-score   support

           0       0.85      0.92      0.89       767
           1       0.61      0.42      0.50       212

    accuracy                           0.82       979
   macro avg       0.73      0.67      0.69       979
weighted avg       0.80      0.82      0.80       979
```

In [34]:
```
knn_model.score(train_x, train_y)
```

Out[34]: 0.8413793103448276

In [35]:
```
knn_model.score(test_x, test_y)
```

Out[35]: 0.81511746680286

In [36]:
```
# NAIVE BAYES CLASSIFICATION
nbc_model = GaussianNB()
nbc_model.fit(train_x, train_y)
pred_nbc = nbc_model.predict(test_x)
print(classification_report(test_y, pred_nbc))
```

```
              precision    recall  f1-score   support

           0       0.91      0.75      0.82       767
           1       0.44      0.72      0.55       212

    accuracy                           0.74       979
   macro avg       0.67      0.73      0.68       979
weighted avg       0.81      0.74      0.76       979
```

In [37]:
```
nbc_model.score(train_x, train_y)
```

Out[37]: 0.7218390804597701

In [38]:
```
nbc_model.score(test_x, test_y)
```

Out[38]: 0.7405515832482125

In [39]:
```
# SUPPORT VECTOR CLASSIFICATION
svm_model = SVC(kernel="rbf")
svm_model.fit(train_x, train_y)
pred_svm = svm_model.predict(test_x)
print(classification_report(test_y, pred_svm))
```

```
              precision    recall  f1-score   support

           0       0.83      0.96      0.89       767
           1       0.64      0.28      0.39       212

    accuracy                           0.81       979
   macro avg       0.73      0.62      0.64       979
weighted avg       0.79      0.81      0.78       979
```

In [40]:
```
svm_model.score(train_x, train_y)
```

Out[40]: 0.8209450830140486

In [41]:
```
svm_model.score(test_x, test_y)
```

Out[41]: 0.8100102145045965

In [42]:
```
#DECISION TREE CLASSIFICATION
dt_model = DecisionTreeClassifier(random_state=50, criterion="gini")
dt_model.fit(train_x, train_y)
pred_dt = dt_model.predict(test_x)
print(classification_report(test_y, pred_dt))
```

```
              precision    recall  f1-score   support

           0       0.90      0.88      0.89       767
           1       0.61      0.66      0.63       212

    accuracy                           0.83       979
   macro avg       0.76      0.77      0.76       979
weighted avg       0.84      0.83      0.84       979
```

In [43]:
```
dt_model.score(train_x, train_y)
```

Out[43]: 1.0

In [44]:
```
dt_model.score(test_x, test_y)
```

Out[44]: 0.8345250255362615

In [45]:
```
#RANDOM FOREST CLASSIFICATION
rfc_model = RandomForestClassifier(random_state=50)
rfc_model.fit(train_x, train_y)
pred_rfc = rfc_model.predict(test_x)
print(classification_report(test_y, pred_rfc))
```

```
              precision    recall  f1-score   support

           0       0.90      0.96      0.93       767
           1       0.80      0.61      0.70       212

    accuracy                           0.88       979
   macro avg       0.85      0.79      0.81       979
weighted avg       0.88      0.88      0.88       979
```

In [46]:
```
rfc_model.score(train_x, train_y)
```

Out[46]: 1.0

In [47]:
```
rfc_model.score(test_x, test_y)
```

Out[47]: 0.8835546475995915

In [48]:
```
#XGBOOST CLASSIFICATION
xgb_model = Xgb.XGBClassifier(n_estimators=100)
xgb_model.fit(train_x, train_y)
pred_xgb = xgb_model.predict(test_x)
print(classification_report(test_y, pred_xgb))
```

```
              precision    recall  f1-score   support

           0       0.90      0.95      0.93       767
           1       0.77      0.63      0.69       212

    accuracy                           0.88       979
   macro avg       0.84      0.79      0.81       979
weighted avg       0.87      0.88      0.87       979
```

In [49]:
```
xgb_model.score(train_x, train_y)
```

Out[49]: 0.9961685823754789

In [50]:
```
xgb_model.score(test_x, test_y)
```

Out[50]: 0.8794688457609806

In [51]:
```
#ADABOOST CLASSIFICATION
from sklearn.ensemble import AdaBoostClassifier
adb_model = AdaBoostClassifier(random_state=50)
adb_model.fit(train_x, train_y)
pred_adb = adb_model.predict(test_x)
print(classification_report(test_y, pred_adb))
```

```
              precision    recall  f1-score   support

           0       0.84      0.93      0.89       767
           1       0.60      0.36      0.45       212

    accuracy                           0.81       979
   macro avg       0.72      0.65      0.67       979
weighted avg       0.79      0.81      0.79       979
```

In [52]:
```
adb_model.score(train_x, train_y)
```

Out[52]: 0.8189016602809707

In [53]:
```
adb_model.score(test_x, test_y)
```

Out[53]: 0.8100102145045965

Here, Decision Tree, Random forest and Xgboost model are overfiting of train and test dataset. So, we do a Hyper parameter tuning and Features selections.

HYPERPARAMETER TUNING

In [54]:
```
#HYPERPERAMETER TUNING OF LOGISTIC REGRESSOR
from sklearn.model_selection import GridSearchCV
log = LogisticRegression()
params = { "tol" : [0.1,0.5,0.8,0.9], "C" :[1,2,8,6,9],
          "solver": ['lbfgs', "liblinear", "newton-cg", "newton-cholesky", "sa
g", "saga"]}
clf1 = GridSearchCV(log, params, cv=5, scoring="accuracy")
clf1.fit(train_x, train_y)
print(clf1.best_params_)
print(clf1.best_score_)
```

```
{'C': 1, 'solver': 'newton-cholesky', 'tol': 0.1}
0.8066411238825031
```

In [55]:
```
#HYPERPERAMETER TUING OF KNN
knn =KNeighborsClassifier()
params_knn= {'algorithm' :['auto', 'ball_tree', 'kd_tree', 'brute'], 'weight
s': ['uniform', 'distance'],
         "n_neighbors" : [1,25,14,13,26,85,45]}
clf2 = GridSearchCV(knn, params_knn, cv=5, scoring="accuracy")
clf2.fit(train_x, train_y)
print(clf2.best_params_)
print(clf2.best_score_)
```

```
{'algorithm': 'auto', 'n_neighbors': 45, 'weights': 'distance'}
0.8679438058748403
```

In [56]:
```
#HYPERPERAMETER TUNING OF NB
nb = GaussianNB()
params_nb = {'var_smoothing' : [0.96,0.25,0.30,0.40, 0.50]}
clf3 = GridSearchCV(nb, params_nb, cv=5, scoring="accuracy")
clf3.fit(train_x, train_y)
print(clf3.best_params_)
print(clf3.best_score_)
```

```
{'var_smoothing': 0.5}
0.7984674329501915
```

In [57]:
```
#HYPERPERAMETER TUNING OF SUPPORT VECTOR
svm = SVC()
params_svm = {"gamma" :["scale", "auto"]}
clf4 = GridSearchCV(svm, params_svm, cv=5, scoring="accuracy")
clf4.fit(train_x, train_y)
print(clf4.best_params_)
print(clf4.best_score_)
```

```
{'gamma': 'scale'}
0.8153256704980842
```

In [58]:
```
#HYPERPERAMETER TUNING OF DECISION TREE
dt = DecisionTreeClassifier()
params_dt = {'criterion':['gini', 'entropy', 'log_loss'], 'max_depth' :[1,25,14,13,45,75,26],'splitter':['best', 'random']}
clf5 = GridSearchCV(dt, params_dt, cv=5, scoring="accuracy")
clf5.fit(train_x, train_y)
print(clf5.best_params_)
print(clf5.best_score_)
```

```
{'criterion': 'log_loss', 'max_depth': 75, 'splitter': 'best'}
0.823499361430396
```

In [59]:
```
#HYPERPERAMETER TUNING OF RANDOMFOREST
rfc = RandomForestClassifier()
params_rfc = {"n_estimators" : [10,15,125,10,8,85],"max_depth" : [10,25,48,85,42,3]}
clf6 = GridSearchCV(rfc, params_rfc, cv=5, scoring="accuracy")
clf6.fit(train_x, train_y)
print(clf6.best_params_)
print(clf6.best_score_)
```

```
{'max_depth': 48, 'n_estimators': 125}
0.8720306513409962
```

In [60]:
```
#HYPERPERAMETER TUNING OF XGBOOST
xgb = Xgb.XGBClassifier()
params_xgb = {'eta': [0.1, 0.2, 0.3,0.4,0.5], 'n_estimators' : [10, 50, 100,12,15], 'max_depth': [3, 6, 9,14]}
clf7 = GridSearchCV(xgb, params_xgb, cv=5, scoring="accuracy")
clf7.fit(train_x, train_y)
print(clf7.best_params_)
print(clf7.best_score_)
```

```
{'eta': 0.2, 'max_depth': 9, 'n_estimators': 100}
0.867432950191571
```

In [61]:
```
#HYPERPERAMETER TUNING OF ADABOOST
adb = AdaBoostClassifier()
params_adb = {'n_estimators' : [10, 50, 100,12,15]}
clf8 = GridSearchCV(xgb, params_adb, cv=5, scoring="accuracy")
clf8.fit(train_x, train_y)
print(clf8.best_params_)
print(clf8.best_score_)
```

```
{'n_estimators': 100}
0.8592592592592592
```

In [62]:
```
#best perameter for model
print("LogisticRegression score is :", clf1.best_params_)
print("KNeighborsClassifier score is :", clf2.best_params_)
print("GaussianNB score is :", clf3.best_params_)
print("Support vector machine score is :", clf4.best_params_)
print("DecisionTreeClassifier score is :", clf5.best_params_)
print("RandomForestClassifier score is :", clf6.best_params_)
print("XGBOOST score is :", clf7.best_params_)
print("AdaBoostClassifier score is :", clf8.best_params_)
```

```
LogisticRegression score is : {'C': 1, 'solver': 'newton-cholesky', 'tol': 0.
1}
KNeighborsClassifier score is : {'algorithm': 'auto', 'n_neighbors': 45, 'wei
ghts': 'distance'}
GaussianNB score is : {'var_smoothing': 0.5}
Support vector machine score is : {'gamma': 'scale'}
DecisionTreeClassifier score is : {'criterion': 'log_loss', 'max_depth': 75,
'splitter': 'best'}
RandomForestClassifier score is : {'max_depth': 48, 'n_estimators': 125}
XGBOOST score is : {'eta': 0.2, 'max_depth': 9, 'n_estimators': 100}
AdaBoostClassifier score is : {'n_estimators': 100}
```

In [63]:
```
#Score for all model
print("LogisticRegression score is :", clf1.best_score_)
print("KNeighborsClassifier score is :", clf2.best_score_)
print("GaussianNB score is :", clf3.best_score_)
print("Support vector machine score is :", clf4.best_score_)
print("DecisionTreeClassifier score is :", clf5.best_score_)
print("RandomForestClassifier score is :", clf6.best_score_)
print("XGBOOST score is :", clf7.best_score_)
print("AdaBoostClassifier score is :", clf8.best_score_)
```

```
LogisticRegression score is : 0.8066411238825031
KNeighborsClassifier score is : 0.8679438058748403
GaussianNB score is : 0.7984674329501915
Support vector machine score is : 0.8153256704980842
DecisionTreeClassifier score is : 0.823499361430396
RandomForestClassifier score is : 0.8720306513409962
XGBOOST score is : 0.867432950191571
AdaBoostClassifier score is : 0.8592592592592592
```

Feature Selection

In [64]:
```
#Correlation
corr = train_x.corr()
corr.style.background_gradient(cmap='coolwarm')
```

Out[64]:

	volatileacidity	citricacid	residualsugar	chlorides	freesulfurdioxide	totalsulfurc
volatileacidity	1.000000	-0.155631	0.063650	0.086718	-0.097849	0.
citricacid	-0.155631	1.000000	0.096165	0.118984	0.095232	0.
residualsugar	0.063650	0.096165	1.000000	0.094485	0.311085	0.
chlorides	0.086718	0.118984	0.094485	1.000000	0.096071	0.
freesulfurdioxide	-0.097849	0.095232	0.311085	0.096071	1.000000	0.
totalsulfurdioxide	0.094374	0.122249	0.403077	0.195789	0.621998	1.
density	0.032416	0.160393	0.840780	0.255892	0.307270	0.
sulphates	-0.026503	0.080483	-0.024425	0.025187	0.070016	0.
alcohol	0.052898	-0.083037	-0.460439	-0.355091	-0.259787	-0.

In [65]:
```
def correlation(dataset, threshold):
    col_corr = set()
    corr_matrix = dataset.corr()
    for i in range (len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i,j]) > threshold:
                colname = corr_matrix.columns[i]
                col_corr.add(colname)
    return col_corr
```

In [66]:
```
corr_features = correlation(train_x, 0.7)
len(set(corr_features))
```

Out[66]: 2

In [67]:
```
corr_features
```

Out[67]:
```
{'alcohol', 'density'}
```

In [68]:
```
#Apply SelectKbest class to extract top Features
from sklearn.feature_selection import SelectKBest, chi2
bestfeatures = SelectKBest(score_func=chi2, k=7)
fit = bestfeatures.fit(x,y)
```

In [69]:
```
fit
```

Out[69]:
```
SelectKBest
SelectKBest(k=7, score_func=<function chi2 at 0x0000026C3E79B7F0>)
```

In [70]:
```
dfscores = pd.DataFrame(fit.scores_)
```

In [71]:
```
dfcolumns = pd.DataFrame(x.columns)
```

In [72]:
```
features = pd.concat([dfcolumns, dfscores], axis=1)
features.columns = ["specs", "score"]
```

In [73]:
```
features
```

Out[73]:

	specs	score
0	volatileacidity	0.849358
1	citricacid	0.274252
2	residualsugar	266.889301
3	chlorides	1.706955
4	freesulfurdioxide	21.459592
5	totalsulfurdioxide	1668.727815
6	density	0.003509
7	sulphates	0.295022
8	alcohol	104.224001

Feature Importance

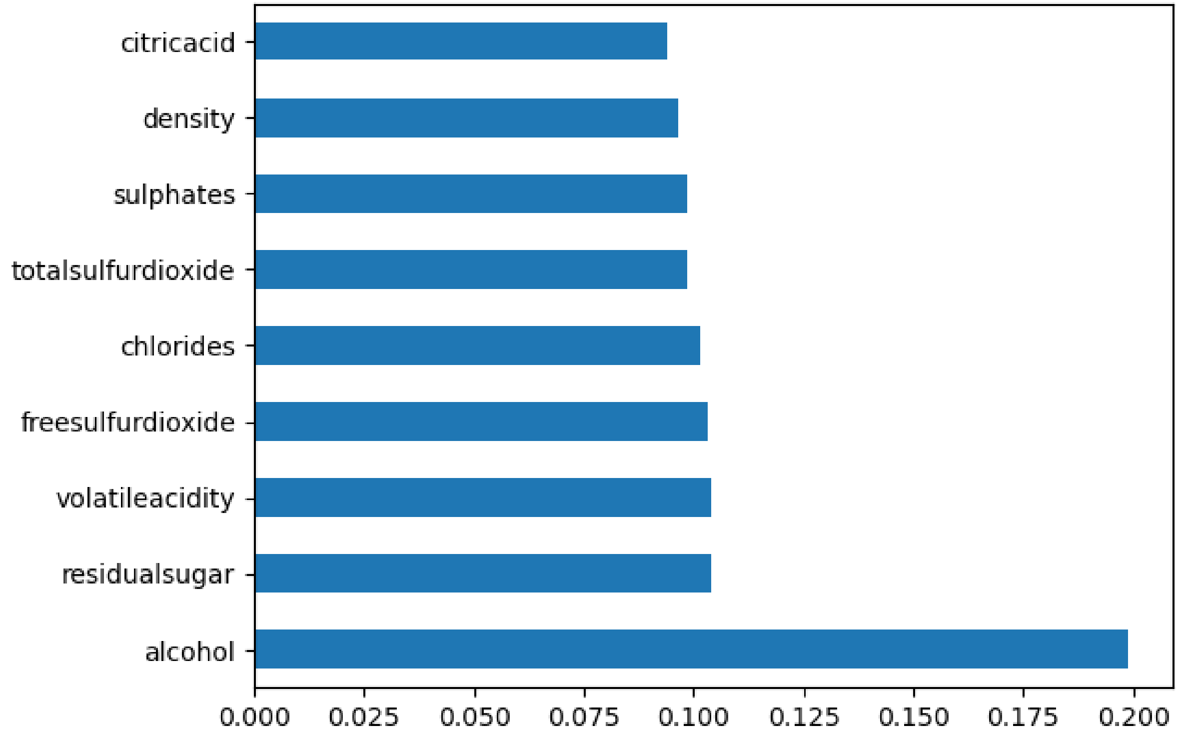
In [74]:
```
from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier()
model.fit(train_x, train_y)
```

Out[74]:
```
▾ ExtraTreesClassifier
ExtraTreesClassifier()
```

In [75]:
```
print(model.feature_importances_)
```

```
[0.10391241 0.09376059 0.1042028  0.10170526 0.10340185 0.09875117
 0.09661143 0.09858443 0.19907007]
```

In [76]:
```
feat_importance = pd.Series(model.feature_importances_, index=x.columns)
feat_importance.nlargest(9).plot(kind="barh")
plt.show()
```



In [77]:
```
fe_model = RandomForestClassifier(random_state=50)

fe_model.fit(train_x, train_y)
```

Out[77]:
```
RandomForestClassifier
RandomForestClassifier(random_state=50)
```

In [78]:
```
feature_scores = pd.Series(fe_model.feature_importances_, index=train_x.column
s).sort_values(ascending=False)
```

In [79]:
```
feature_scores
```

Out[79]:
```
alcohol               0.188746
residualsugar         0.113954
totalsulfurdioxide    0.107905
chlorides             0.107803
freesulfurdioxide     0.107724
volatileacidity       0.107276
sulphates             0.097564
citricacid            0.093311
density               0.075716
dtype: float64
```

After the all Feature Selection method use then we decide to drop a density column for best accuracy. So, start with second time generate Model

Start Whole Process of Model training in second time

In [80]:
```
df1.head()
```

Out[80]:

	Quality_Category	volatileacidity	citricacid	residualsugar	chlorides	freesulfurdioxide	totalsulf
0	0	0.30	0.34	1.6	0.049	14	
1	0	0.23	0.32	8.5	0.058	47	
2	0	0.28	0.40	6.9	0.050	30	
3	0	0.32	0.16	7.0	0.045	30	
4	0	0.27	0.36	20.7	0.045	45	

In [81]:
```
X = df1.drop(["Quality_Category", "density"], axis=1)
Y = df1["Quality_Category"]
```

In [82]:
```
train_x1, test_x1, train_y1, test_y1 = train_test_split(X,Y, random_state=50,
test_size=0.2, stratify=y)
```

In [83]:
```
train_x1
```

Out[83]:

	volatileacidity	citricacid	residualsugar	chlorides	freesulfurdioxide	totalsulfurdioxide	sulp
3073	0.25	0.38	7.9	0.045	54	208	
2483	0.27	0.27	1.7	0.034	25	122	
2155	0.24	0.37	1.8	0.031	6	61	
4235	0.28	0.36	1.8	0.041	38	90	
635	0.34	0.14	5.8	0.046	49	197	
...	...	...	...	...	...	...	
3851	0.18	0.29	4.6	0.032	68	137	
2015	0.32	0.22	16.7	0.046	38	133	
3337	0.17	0.27	4.6	0.050	23	98	
1978	0.20	0.30	14.2	0.056	53	213	
1475	0.16	0.49	2.0	0.056	20	124	

3915 rows × 8 columns

In [84]:
```
#create reset index
train_x1.reset_index(inplace=True, drop=True)
test_x1.reset_index(inplace=True, drop=True)

train_y1.reset_index(inplace=True, drop=True)
test_y1.reset_index(inplace=True, drop=True)
```

In [85]:
```
scaler.fit(train_x1)
train_x1 = pd.DataFrame(scaler.transform(train_x1), columns=train_x1.columns)
test_x1 = pd.DataFrame(scaler.transform(test_x1), columns=test_x1.columns)
```

In [86]:
```
train_x1
```

Out[86]:

	volatileacidity	citricacid	residualsugar	chlorides	freesulfurdioxide	totalsulfurdioxide	sulp
0	0.166667	0.228916	0.111963	0.106825	0.181185	0.461717	0.27
1	0.186275	0.162651	0.016871	0.074184	0.080139	0.262181	0.33
2	0.156863	0.222892	0.018405	0.065282	0.013937	0.120650	0.13
3	0.196078	0.216867	0.018405	0.094955	0.125436	0.187935	0.88
4	0.254902	0.084337	0.079755	0.109792	0.163763	0.436195	0.56
...	...	...	...	...	...	...	
3910	0.098039	0.174699	0.061350	0.068249	0.229965	0.296984	0.18
3911	0.235294	0.132530	0.246933	0.109792	0.125436	0.287703	0.52
3912	0.088235	0.162651	0.061350	0.121662	0.073171	0.206497	0.29
3913	0.117647	0.180723	0.208589	0.139466	0.177700	0.473318	0.27
3914	0.078431	0.295181	0.021472	0.139466	0.062718	0.266821	0.31

3915 rows × 8 columns

In [87]:
```
#KNeighborsClassifier
knn = KNeighborsClassifier(algorithm="auto", n_neighbors=25, weights="distanc
e")
knn.fit(train_x1, train_y1)
pred2 = knn.predict(test_x1)
print(classification_report(test_y1, pred2))
print(accuracy_score(test_y1, pred2))
```

```
              precision    recall  f1-score   support

           0       0.90      0.94      0.92       767
           1       0.74      0.64      0.69       212

    accuracy                           0.87       979
   macro avg       0.82      0.79      0.80       979
weighted avg       0.87      0.87      0.87       979

0.874361593462717
```

In [88]:
```
#LogisticRegression
log = LogisticRegression(C=1, solver="liblinear", tol=0.5)
log.fit(train_x1, train_y1)
pred1 = log.predict(test_x1)
print(classification_report(test_y1,pred1))
print(accuracy_score(test_y1, pred1))
```

```
              precision    recall  f1-score   support

           0       0.78      1.00      0.88       767
           1       0.00      0.00      0.00       212

    accuracy                           0.78       979
   macro avg       0.39      0.50      0.44       979
weighted avg       0.61      0.78      0.69       979

0.7834525025536262
```

In [89]:
```
#GaussianNB
nb = GaussianNB(var_smoothing=0.5)
nb.fit(train_x1, train_y1)
pred3 = nb.predict(test_x1)
print(classification_report(test_y1,pred3))
print(accuracy_score(test_y1, pred3))
```

```
              precision    recall  f1-score   support

           0       0.82      0.95      0.88       767
           1       0.56      0.25      0.34       212

    accuracy                           0.79       979
   macro avg       0.69      0.60      0.61       979
weighted avg       0.76      0.79      0.76       979

0.7946884576098059
```

In [90]:
```
#SVC
svm = SVC(gamma="auto")
svm.fit(train_x1, train_y1)
pred4 = svm.predict(test_x1)
print(classification_report(test_y1,pred4))
print(accuracy_score(test_y1, pred4))
```

```
              precision    recall  f1-score   support

           0       0.78      1.00      0.88       767
           1       0.00      0.00      0.00       212

    accuracy                           0.78       979
   macro avg       0.39      0.50      0.44       979
weighted avg       0.61      0.78      0.69       979

0.7834525025536262
```

In [91]:
```
#DecisionTreeClassifier
dt = DecisionTreeClassifier(criterion="log_loss", max_depth=75, splitter="rand
om")
dt.fit(train_x1, train_y1)
pred5 = dt.predict(test_x1)
print(classification_report(test_y1,pred5))
print(accuracy_score(test_y1, pred5))
```

```
              precision    recall  f1-score   support

           0       0.90      0.85      0.87       767
           1       0.55      0.67      0.60       212

    accuracy                           0.81       979
   macro avg       0.73      0.76      0.74       979
weighted avg       0.83      0.81      0.82       979

0.8089887640449438
```

In [92]:
```
dt.score(train_x1, train_y1)
```

Out[92]:
```
1.0
```

In [93]:
```
#RandomForestClassifier
rfc = RandomForestClassifier(max_depth=48 ,n_estimators= 125)
rfc.fit(train_x1, train_y1)
pred6 = rfc.predict(test_x1)
print(classification_report(test_y1,pred6))
print(accuracy_score(test_y1, pred6))
```

```
              precision    recall  f1-score   support

           0       0.90      0.96      0.93       767
           1       0.81      0.61      0.70       212

    accuracy                           0.88       979
   macro avg       0.85      0.79      0.81       979
weighted avg       0.88      0.88      0.88       979

0.8845760980592441
```

In [94]:
```
rfc.score(train_x1, train_y1)
```

Out[94]:
```
1.0
```

In [95]:
```
#XGBClassifier
xgb = Xgb.XGBClassifier(eta=0.2 ,max_depth=9 ,n_estimators= 100)
xgb.fit(train_x1, train_y1)
pred7 = xgb.predict(test_x1)
print(classification_report(test_y1,pred7))
print(accuracy_score(test_y1, pred7))
```

```
              precision    recall  f1-score   support

           0       0.90      0.94      0.92       767
           1       0.74      0.64      0.69       212

    accuracy                           0.87       979
   macro avg       0.82      0.79      0.80       979
weighted avg       0.87      0.87      0.87       979

0.8733401430030644
```

In [96]:
```
xgb.score(train_x1, train_y1)
```

Out[96]: 1.0

In [97]:
```
#AdaBoostClassifier
adb = AdaBoostClassifier(n_estimators= 100)
adb.fit(train_x1, train_y1)
pred8 = adb.predict(test_x1)
print(classification_report(test_y1,pred8))
print(accuracy_score(test_y1, pred8))
```

```
              precision    recall  f1-score   support

           0       0.84      0.93      0.88       767
           1       0.57      0.33      0.42       212

    accuracy                           0.80       979
   macro avg       0.70      0.63      0.65       979
weighted avg       0.78      0.80      0.78       979

0.8018386108273748
```

Score After Hyper Parameter tuning and Feature Selection of all models

In [98]:
```
print('LogisticRegression score is ', accuracy_score(test_y1, pred1))
print('KNeighborsClassifier score is', accuracy_score(test_y1, pred2))
print('GaussianNB score is', accuracy_score(test_y1, pred3))
print('Support Vector Machine score is', accuracy_score(test_y1, pred4))
print('DecisionTreeClassifier score is', accuracy_score(test_y1, pred5))
print('RandomForestClassifier score is', accuracy_score(test_y1, pred6))
print('XGBClassifier score is', accuracy_score(test_y1, pred7))
print('AdaBoostClassifier score is', accuracy_score(test_y1, pred8))
```

```
LogisticRegression score is  0.7834525025536262
KNeighborsClassifier score is 0.874361593462717
GaussianNB score is 0.7946884576098059
Support Vector Machine score is 0.7834525025536262
DecisionTreeClassifier score is 0.8089887640449438
RandomForestClassifier score is 0.8845760980592441
XGBClassifier score is 0.8733401430030644
AdaBoostClassifier score is 0.8018386108273748
```

CONCLUSION :- IN ABOVE GENERATED MODEL IN RANDOM FOREST AND XGBOOST CLASSIFIER GIVE ACCURACY IN TRAIN SET IS 100% BUT TESTING SET ACCURACY GIVES 88.25% AND 87.33%, RESPECTIVELY. SO, IN THIS CASE MODEL PERFORMING OVERFITING. SO THAT WE DO A OVER SAMPLING BECAUSE THE DATASET HAVE INBALANCED SO WE DO IT AND CHECK THE ACCURACY OF MODEL.

OVER SAMPLING

In [99]:
```
#for over sampling import libraries.
```

In [100]:
```
from imblearn.over_sampling import SMOTE
```

In [101]:
```
os = SMOTE(random_state=50)
xos, yos = os.fit_resample(X, Y)
```

In [102]:
```
train_x11, test_x11, train_y11, test_y11 = train_test_split(xos,yos, random_st
ate=50, test_size=0.2)
```

In [103]:
```
scaler.fit(train_x11)
train_x11 = pd.DataFrame(scaler.transform(train_x11), columns=train_x11.column
s)
test_x11 = pd.DataFrame(scaler.transform(test_x11), columns=test_x11.columns)
```

In [104]:
```
train_x11
```

Out[104]:

	volatileacidity	citricacid	residualsugar	chlorides	freesulfurdioxide	totalsulfurdioxide	sulp
0	0.441176	0.096386	0.032209	0.080119	0.111498	0.276102	0.39
1	0.135439	0.218328	0.039572	0.090630	0.195122	0.385151	0.32
2	0.137255	0.301205	0.200920	0.118694	0.188153	0.417633	0.55
3	0.362745	0.162651	0.062883	0.077151	0.052265	0.164733	0.17
4	0.088235	0.192771	0.015337	0.130564	0.156794	0.327146	0.68
...	...	...	...	...	...	...	
6131	0.106997	0.168675	0.011740	0.107487	0.069686	0.185615	0.37
6132	0.098039	0.168675	0.147239	0.089021	0.094077	0.245940	0.29
6133	0.254902	0.198795	0.139571	0.080119	0.153310	0.378190	0.22
6134	0.137951	0.173579	0.010166	0.052204	0.101045	0.238979	0.36
6135	0.323529	0.132530	0.102761	0.077151	0.073171	0.250580	0.21

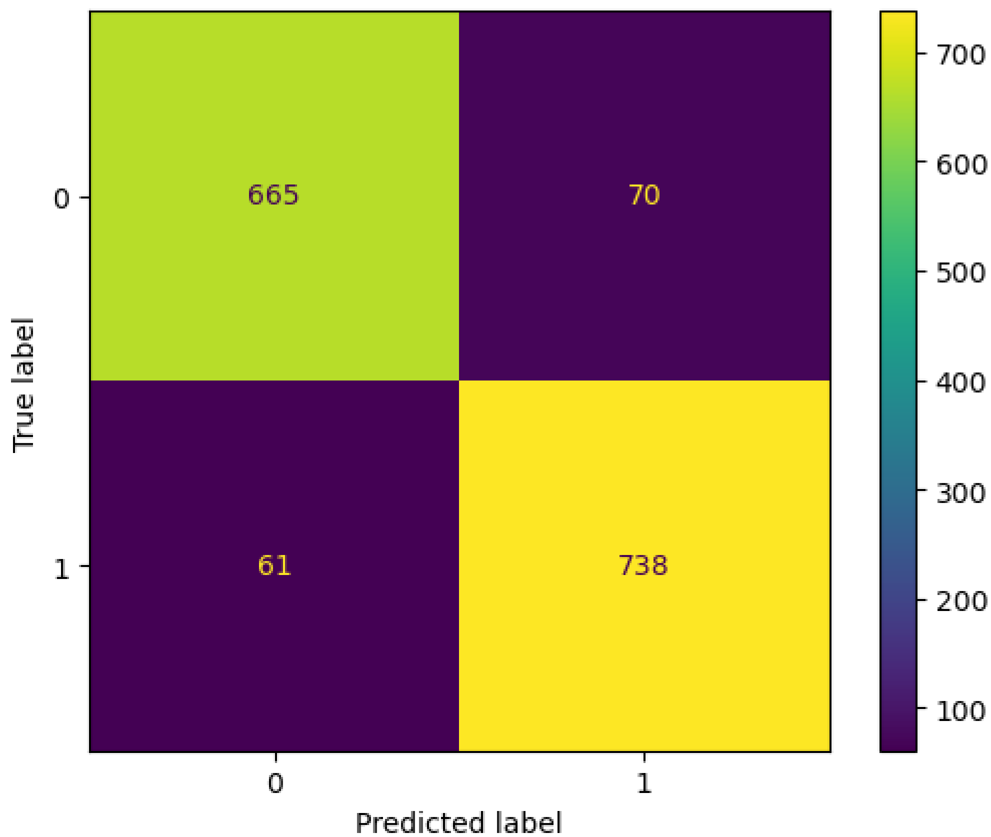
6136 rows × 8 columns

In [106]:
```
#XGBClassifier
xgb.fit(train_x11, train_y11)
pred_1x = xgb.predict(test_x11)
print(classification_report(test_y11,pred_1x))
print(accuracy_score(test_y11, pred_1x))
print(xgb.score(train_x11, train_y11))
```

```
              precision    recall  f1-score   support

           0       0.92      0.90      0.91       735
           1       0.91      0.92      0.92       799

    accuracy                           0.91      1534
   macro avg       0.91      0.91      0.91      1534
weighted avg       0.91      0.91      0.91      1534

0.9146023468057366
0.9998370273794003
```

In [107]:
```
print('XGBClassifier of confusion_matrix is:')
print(ConfusionMatrixDisplay.from_predictions(test_y11, pred_1x))
```

```
XGBClassifier of confusion_matrix is:
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x00
00026C36AEA500>
```

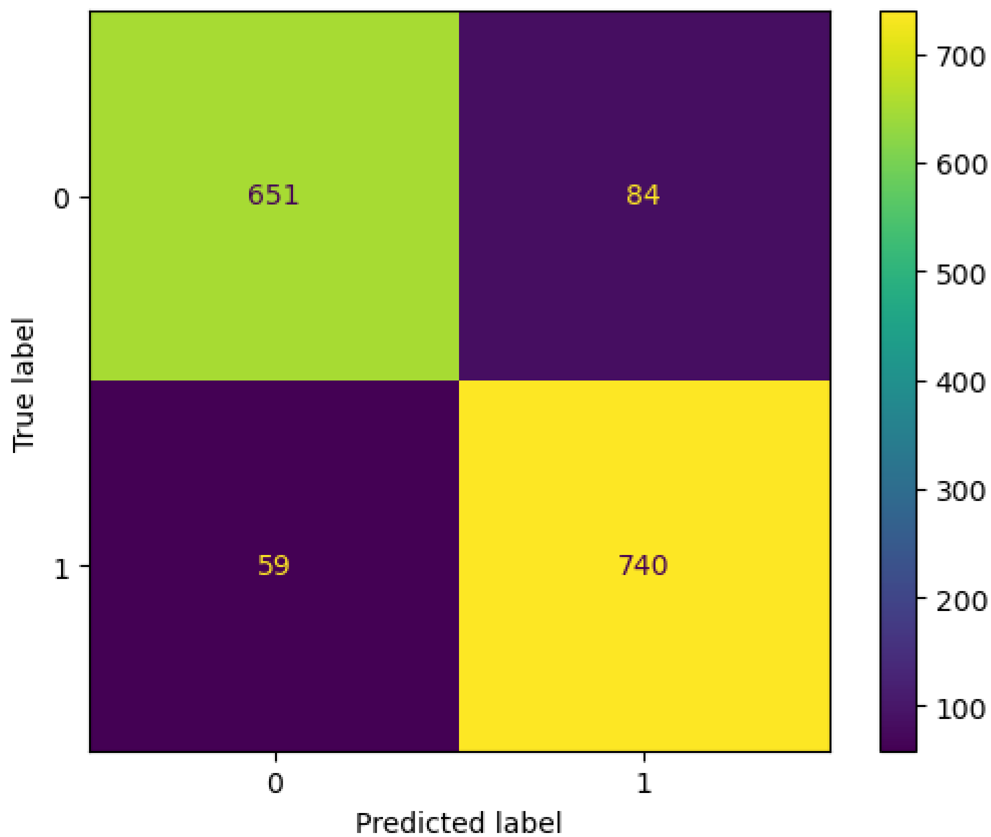


In [108]:
```
#RandomForestClassifier
rfc.fit(train_x11, train_y11)
pred_2r = rfc.predict(test_x11)
print(classification_report(test_y11,pred_2r))
print(accuracy_score(test_y11, pred_2r))
print(rfc.score(train_x11, train_y11))
```

```
              precision    recall  f1-score   support

           0       0.92      0.89      0.90       735
           1       0.90      0.93      0.91       799

    accuracy                           0.91      1534
   macro avg       0.91      0.91      0.91      1534
weighted avg       0.91      0.91      0.91      1534

0.9067796610169492
1.0
```

In [109]:
```
print('RandomForestClassifier of confusion_matrix is:')
print(ConfusionMatrixDisplay.from_predictions(test_y11, pred_2r))
```

```
RandomForestClassifier of confusion_matrix is:
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x00
00026C3EB97BE0>
```



Testing the new data for checking

In [110]:
```
x.tail()
```

Out[110]:

	volatileacidity	citricacid	residualsugar	chlorides	freesulfurdioxide	totalsulfurdioxide	dens
4889	0.21	0.29	1.6	0.039	24	92	0.9
4890	0.32	0.36	8.0	0.047	57	168	0.9
4891	0.24	0.19	1.2	0.041	30	111	0.9
4892	0.29	0.30	1.1	0.022	20	110	0.9
4893	0.21	0.38	0.8	0.020	22	98	0.9

In [111]:
```
new_df = {"volatileacidity": 0.30,'citricacid': 0.31, "residualsugar": 1.0,'ch
lorides': 0.20,
          'freesulfurdioxide': 19, 'totalsulfurdioxide': 105, 'sulphates' : 0.
37, "alcohol": 12.0}
```

In [112]:
```
index =[0]
```

In [113]:
```
new_df = pd.DataFrame(new_df, index=index)
new_df
```

Out[113]:

	volatileacidity	citricacid	residualsugar	chlorides	freesulfurdioxide	totalsulfurdioxide	sulphate
0	0.3	0.31	1.0	0.2	19	105	0.3

In [114]:
```
new_df = pd.DataFrame(scaler.transform(new_df), columns=new_df.columns)
```

In [115]:
```
new_df
```

Out[115]:

	volatileacidity	citricacid	residualsugar	chlorides	freesulfurdioxide	totalsulfurdioxide	sulphate
0	0.215686	0.186747	0.006135	0.566766	0.059233	0.222738	0.18987

In [116]:
```
prediction = xgb.predict(new_df)
prediction
```

Out[116]:
```
array([0])
```

In [117]:
```
prediction2 = rfc.predict(new_df)
prediction2
```

Out[117]:
```
array([0], dtype=int64)
```

In [118]:
```
new_df1 = {"volatileacidity": 0.29,'citricacid': 0.30, "residualsugar": 1.1,'c
hlorides': 0.022,
          'freesulfurdioxide': 20, 'totalsulfurdioxide': 110, 'sulphates' : 0.
38, "alcohol": 12.8}
```

In [119]:
```
new_df1 = pd.DataFrame(new_df1, index=index)
new_df1
```

Out[119]:

	volatileacidity	citricacid	residualsugar	chlorides	freesulfurdioxide	totalsulfurdioxide	sulphate
0	0.29	0.3	1.1	0.022	20	110	0.3

In [120]:
```
new_df1 = pd.DataFrame(scaler.transform(new_df1), columns=new_df1.columns)
new_df1
```

Out[120]:

	volatileacidity	citricacid	residualsugar	chlorides	freesulfurdioxide	totalsulfurdioxide	sulphate
0	0.205882	0.180723	0.007669	0.038576	0.062718	0.234339	0.20253

In [121]:
```
prediction3 = xgb.predict(new_df1)
prediction3
```

Out[121]:
```
array([1])
```

In [122]:
```
prediction4 = rfc.predict(new_df1)
prediction4
```

Out[122]:
```
array([1], dtype=int64)
```

In [123]:
```
print('RandomForestClassifier score is ', accuracy_score(test_y11, pred_2r))
print('Xgboost Classifier score is', accuracy_score(test_y11, pred_1x))
```

```
RandomForestClassifier score is  0.9067796610169492
Xgboost Classifier score is 0.9146023468057366
```

CONCLUSION : From the above all Different Model's Random Forest Classification and Xgboost classifier have generated the model with higher accuracy in both defulat model and Hyperperameter tuning. But, in both model train score is higher then testing score. So, After Over sampling the dataset Xgboost give higher score and score is : 0.9146023468057366