In [1]:
```
import pandas as pd
```

In [2]:
```
df=pd.read_csv("D:\winequalityN.csv")
df.head(3)
```

Out[2]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
0	white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45
1	white	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49
2	white	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44

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

Out[3]: (6497, 13)

In [4]:
```
df.info()
```

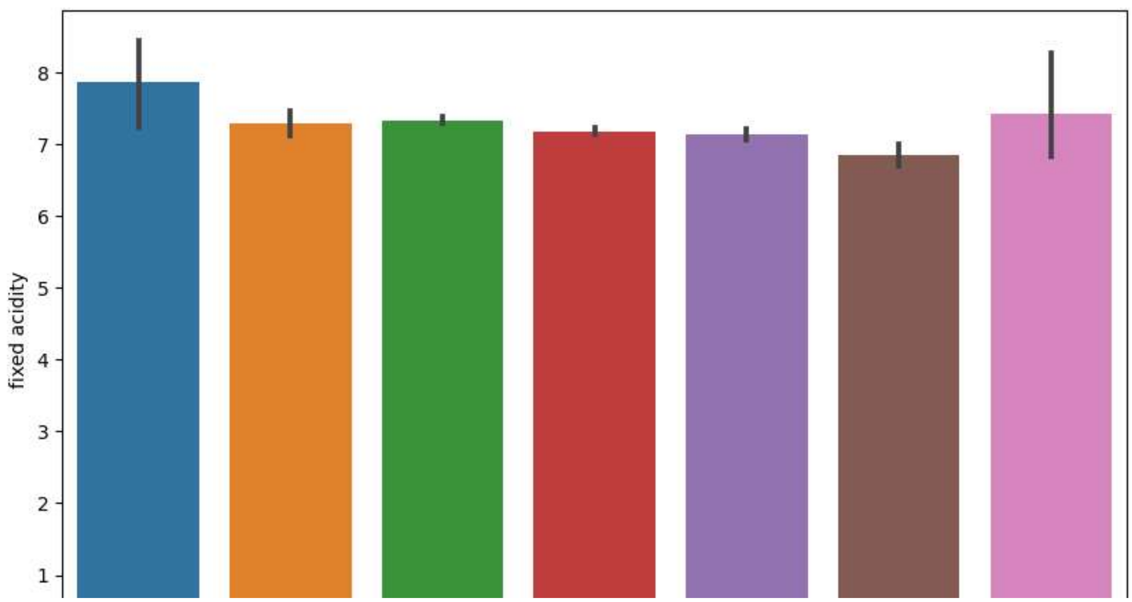
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   type                  6497 non-null   object 
 1   fixed acidity         6487 non-null   float64
 2   volatile acidity      6489 non-null   float64
 3   citric acid           6494 non-null   float64
 4   residual sugar        6495 non-null   float64
 5   chlorides             6495 non-null   float64
 6   free sulfur dioxide   6497 non-null   float64
 7   total sulfur dioxide  6497 non-null   float64
 8   density               6497 non-null   float64
 9   pH                    6488 non-null   float64
 10  sulphates             6493 non-null   float64
 11  alcohol               6497 non-null   float64
 12  quality               6497 non-null   int64  
dtypes: float64(11), int64(1), object(1)
memory usage: 660.0+ KB
```

Let's do some plotting to know how the data columns are distributed in the dataset¶

In [5]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

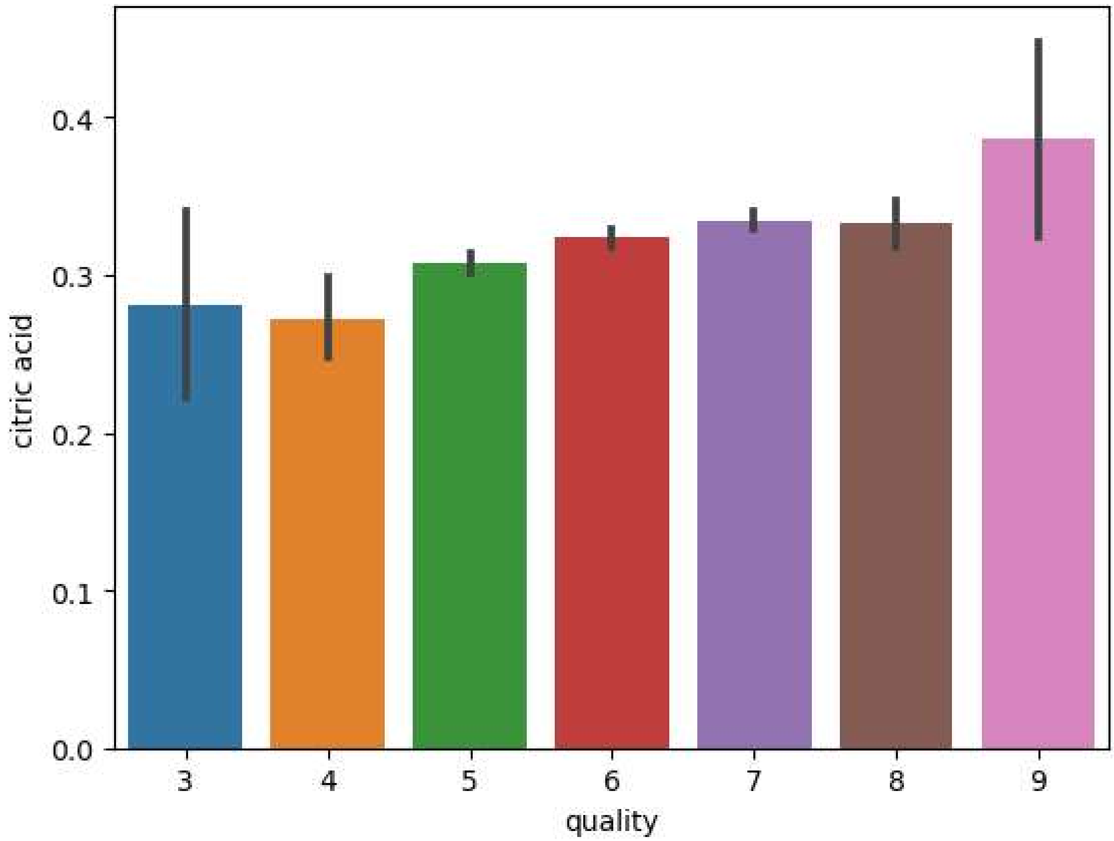
In [6]:
```
#Here we see that fixed acidity does not give any specification to classify th
fig=plt.figure(figsize=(10,6))
sns.barplot(x='quality',y='fixed acidity', data=df)
```

Out[6]:
```
<Axes: xlabel='quality', ylabel='fixed acidity'>
```



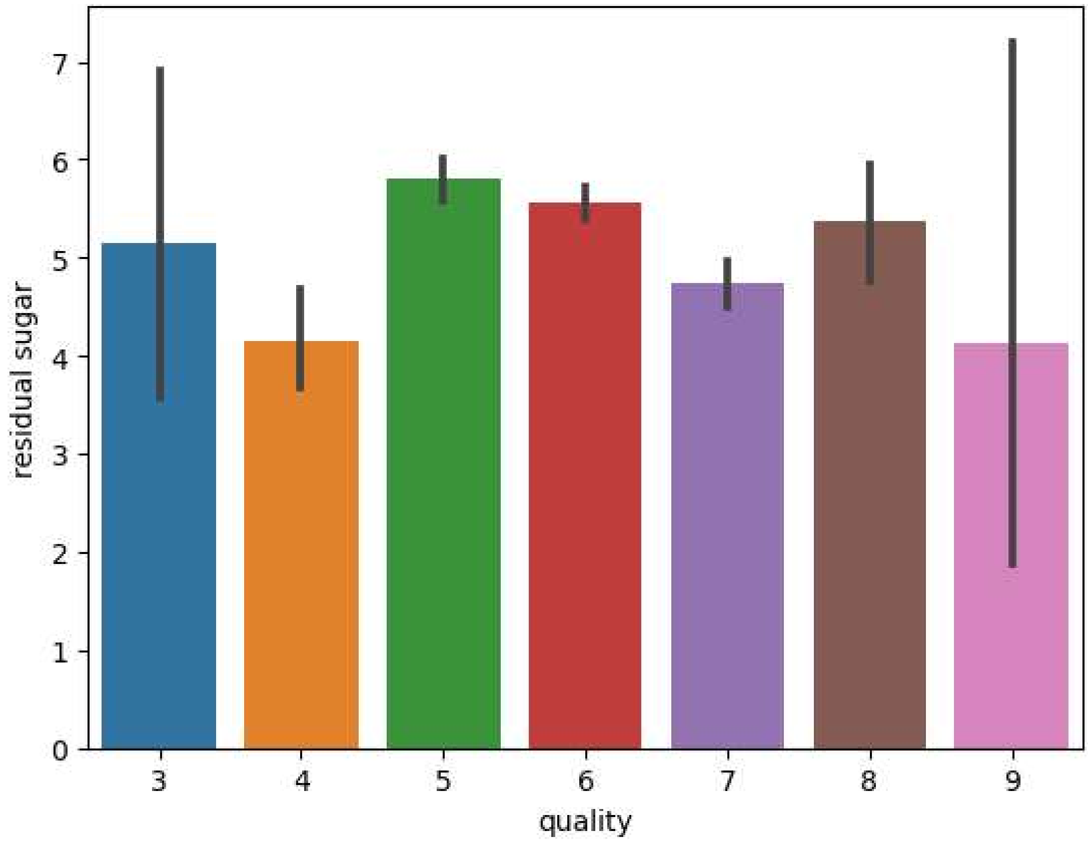
In [7]:
```
sns.barplot(x='quality',y='citric acid', data=df)
```

Out[7]: <Axes: xlabel='quality', ylabel='citric acid'>



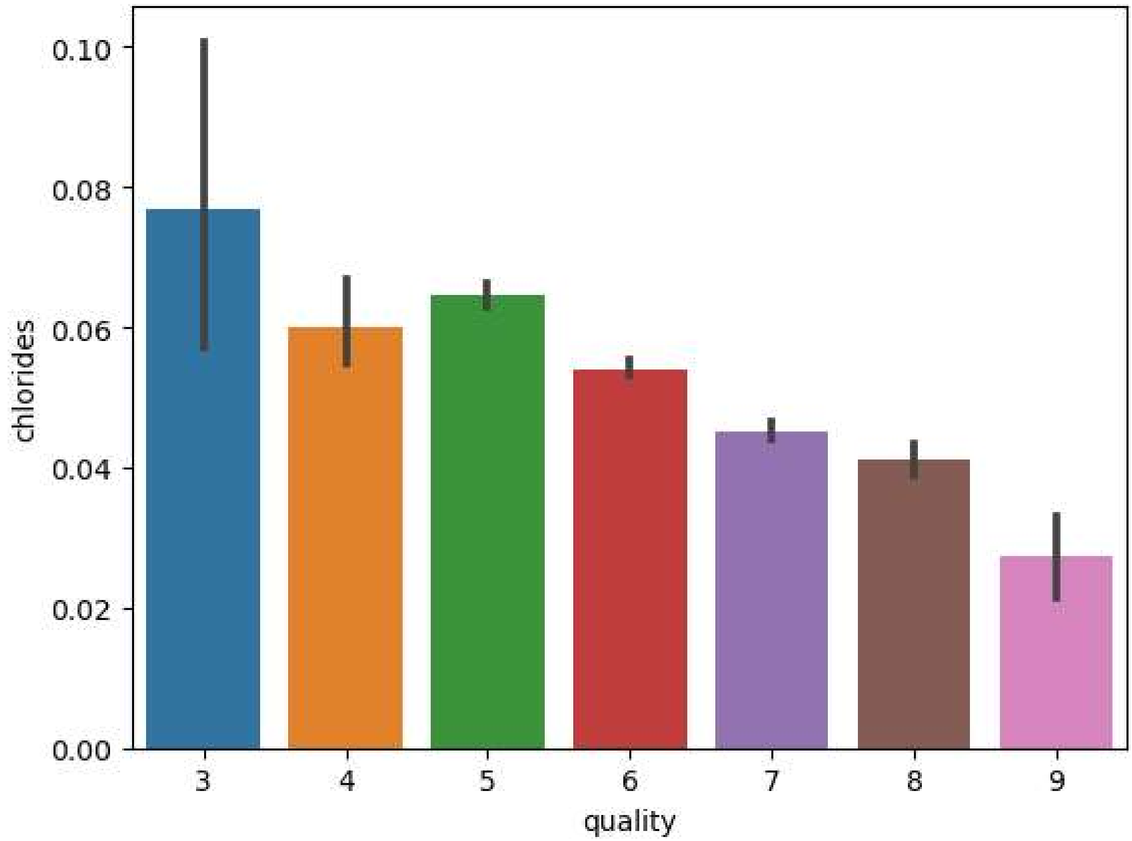
In [8]:
```
sns.barplot(x='quality',y='residual sugar', data=df)
```

Out[8]: <Axes: xlabel='quality', ylabel='residual sugar'>



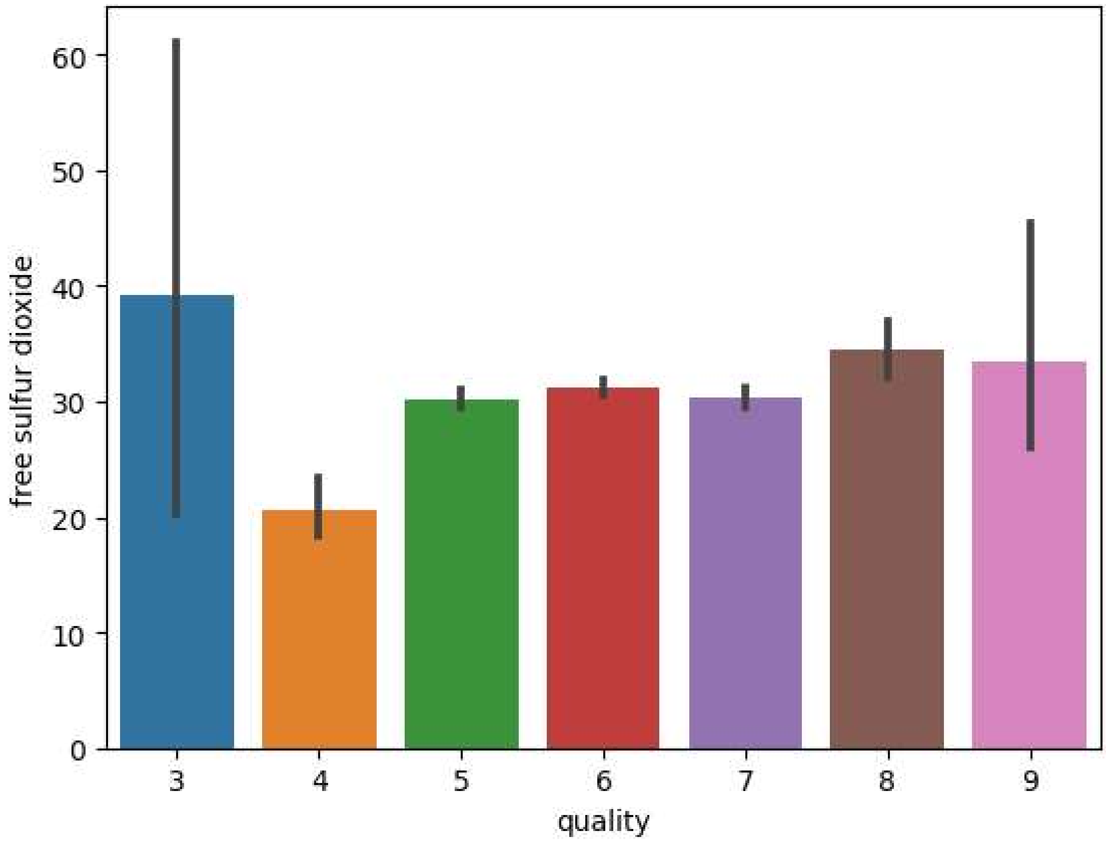
In [9]:
```
sns.barplot(x='quality',y='chlorides', data=df)
```

Out[9]: <Axes: xlabel='quality', ylabel='chlorides'>



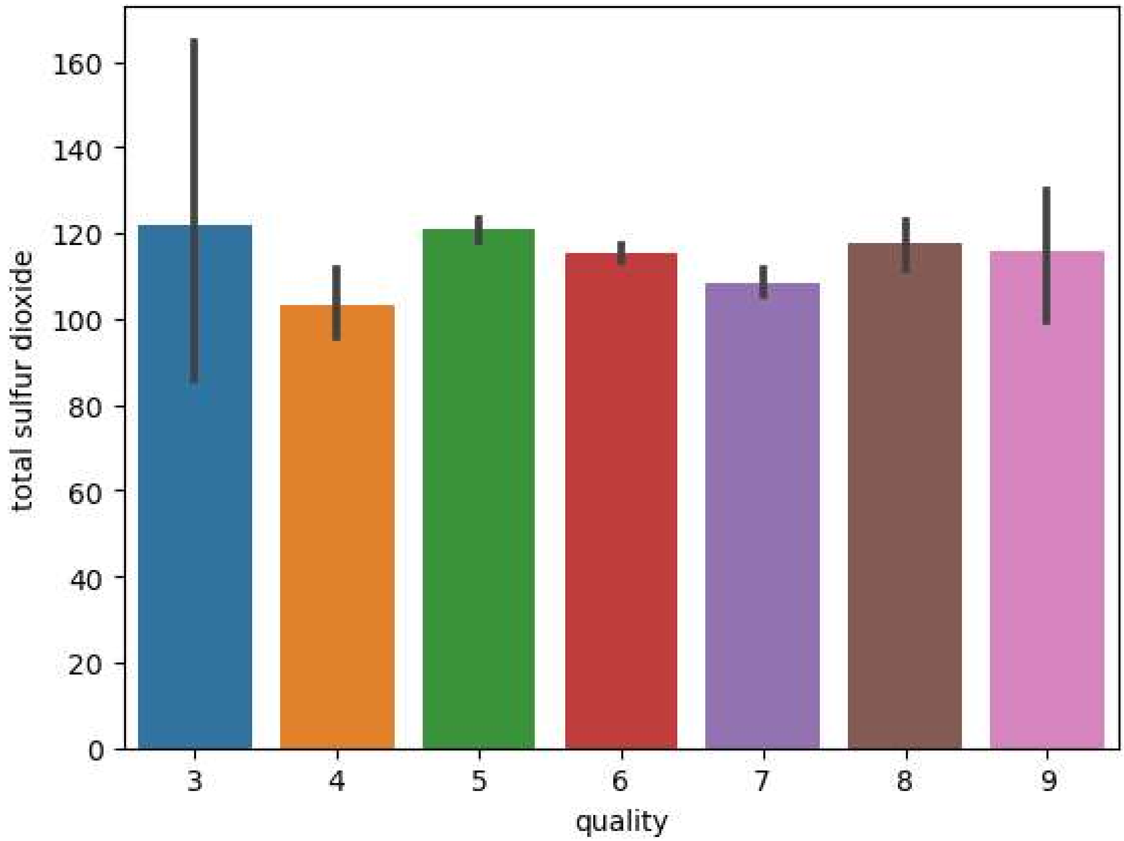
In [10]:
```
sns.barplot(x='quality',y='free sulfur dioxide', data=df)
```

Out[10]: <Axes: xlabel='quality', ylabel='free sulfur dioxide'>



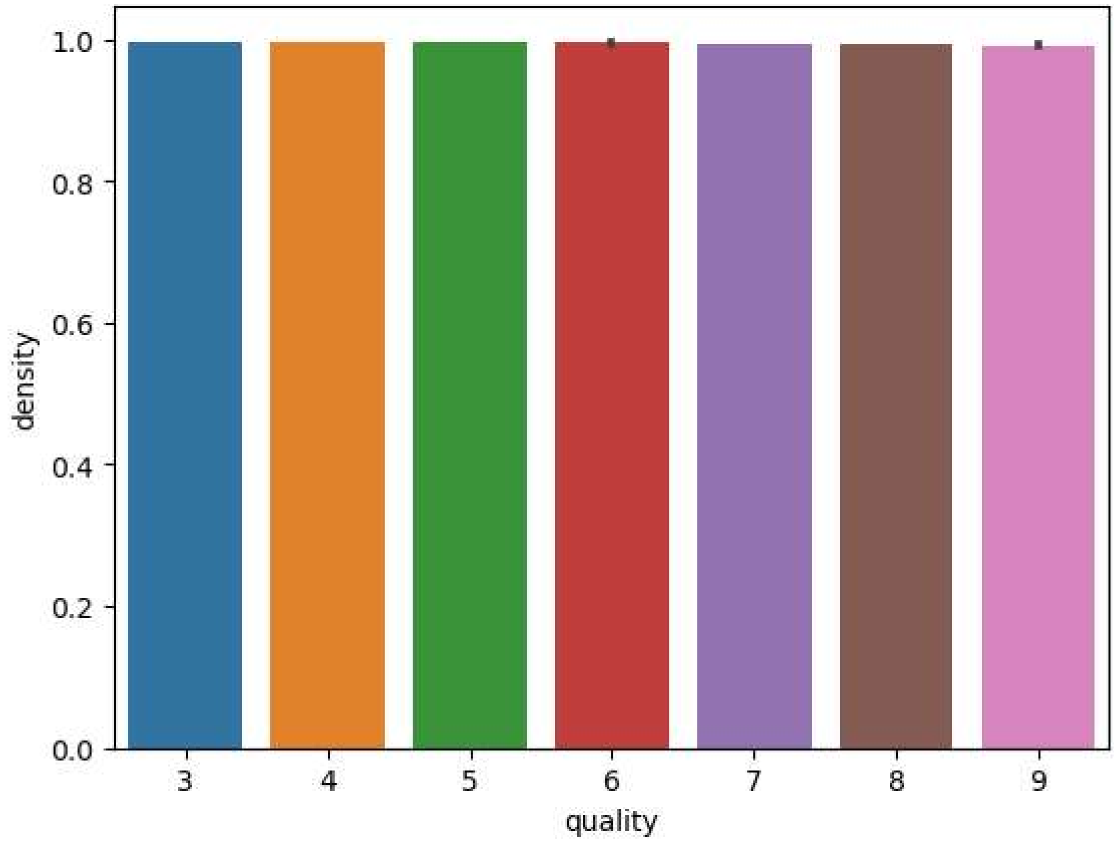
In [11]:
```
sns.barplot(x='quality',y='total sulfur dioxide', data=df)
```

Out[11]: <Axes: xlabel='quality', ylabel='total sulfur dioxide'>



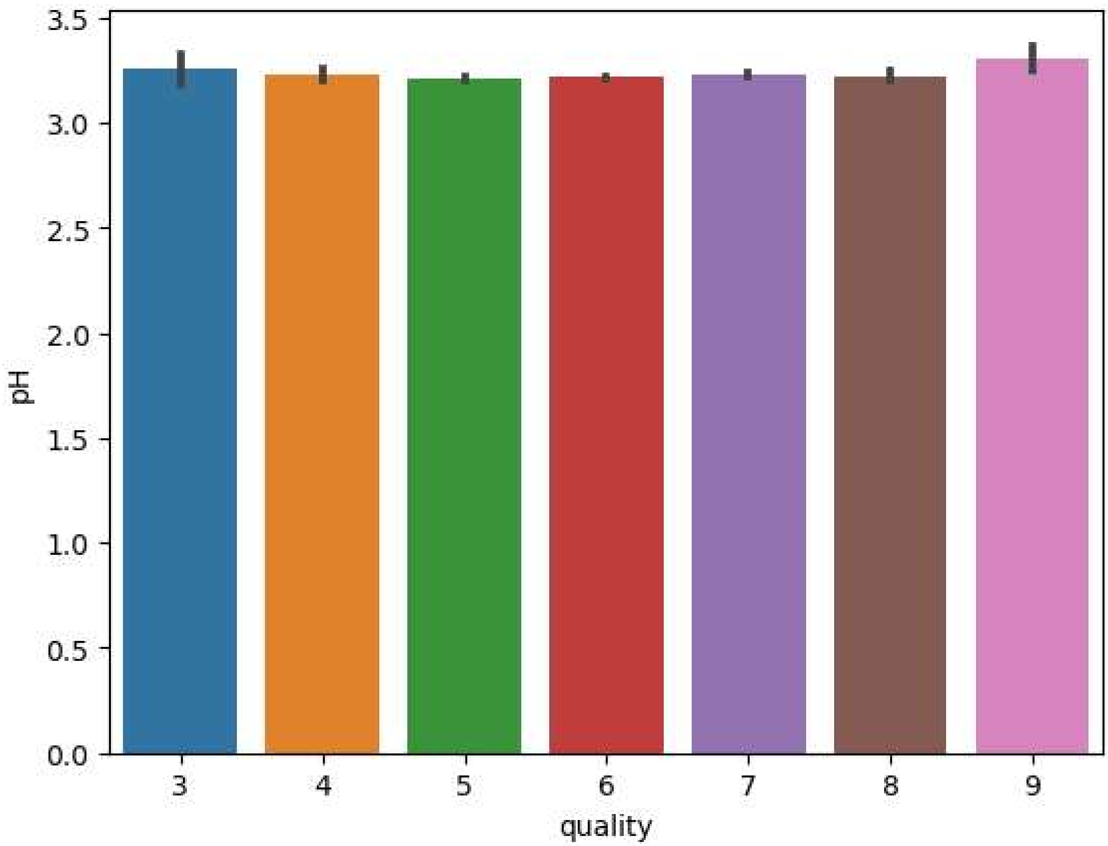
In [12]:
```
sns.barplot(x='quality',y='density', data=df)
```

Out[12]: <Axes: xlabel='quality', ylabel='density'>



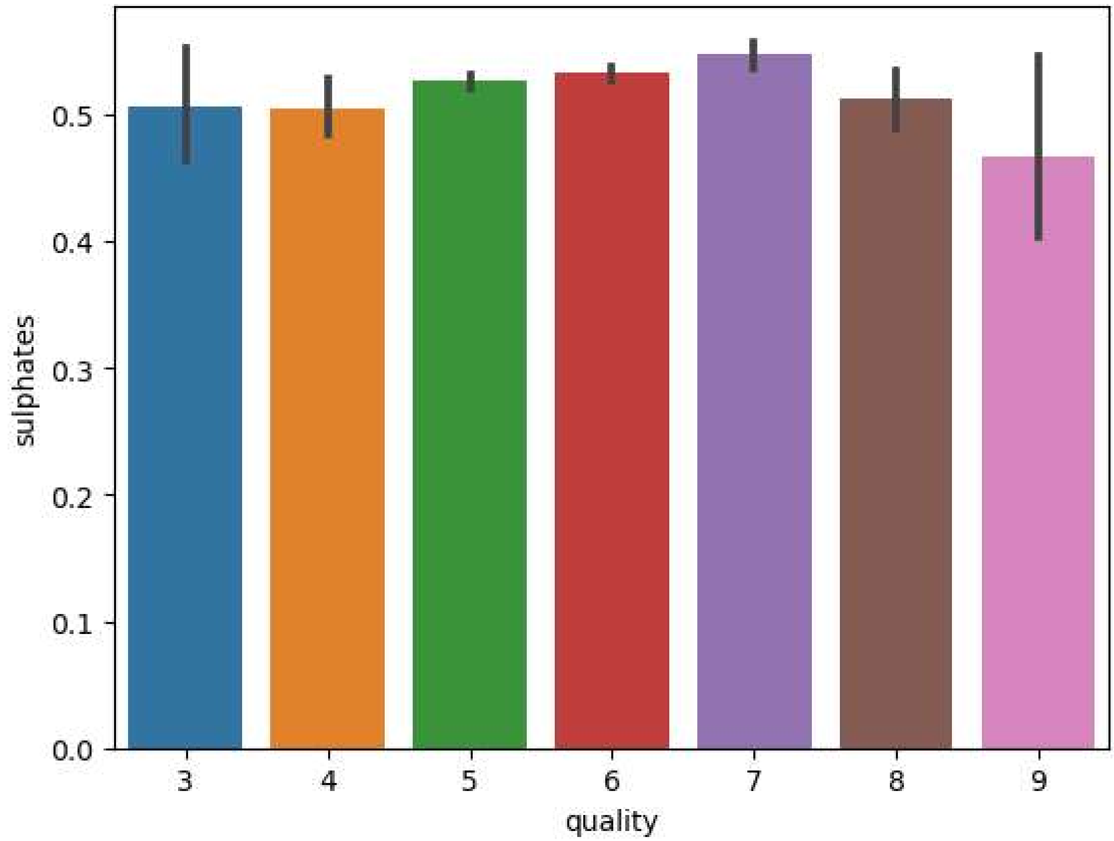
In [13]:
```
sns.barplot(x='quality',y='pH', data=df)
```

Out[13]:
```
<Axes: xlabel='quality', ylabel='pH'>
```



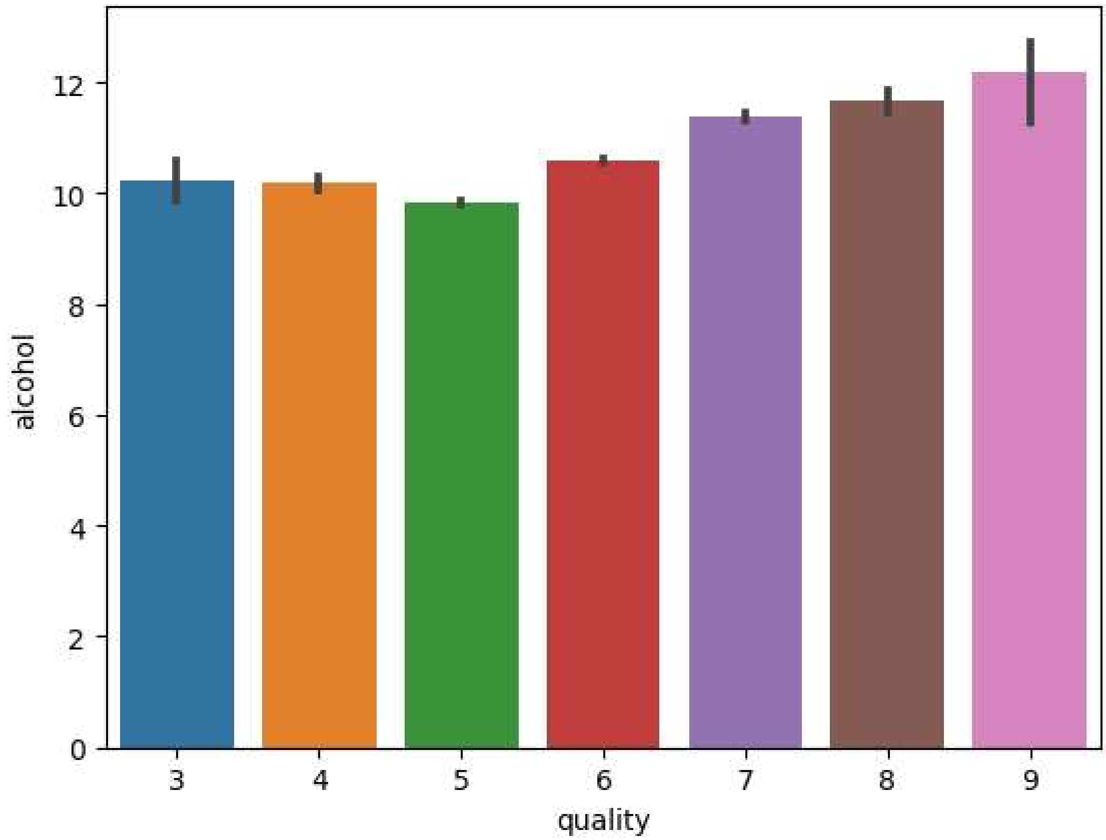
In [14]:
```
sns.barplot(x='quality',y='sulphates', data=df)
```

Out[14]: <Axes: xlabel='quality', ylabel='sulphates'>



In [15]:
```
sns.barplot(x='quality',y='alcohol', data=df)
```

Out[15]: <Axes: xlabel='quality', ylabel='alcohol'>



Remove Outlier and fill NaN value

In [16]:
```
df.head()
```

Out[16]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
0	white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45
1	white	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49
2	white	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44
3	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40
4	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40

In [17]:
```
df=df.drop(columns='type')
```

In [18]:
```
df.head(2)
```

Out[18]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.001	3.0	0.45	8.8
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.994	3.3	0.49	9.5

In [19]:
```
q1=df.quantile(0.25)
q3=df.quantile(0.75)
q1,q3
```

Out[19]:
```
(fixed acidity            6.40000
 volatile acidity         0.23000
 citric acid              0.25000
 residual sugar           1.80000
 chlorides                0.03800
 free sulfur dioxide     17.00000
 total sulfur dioxide    77.00000
 density                  0.99234
 pH                       3.11000
 sulphates                0.43000
 alcohol                  9.50000
 quality                  5.00000
 Name: 0.25, dtype: float64,
 fixed acidity             7.70000
 volatile acidity          0.40000
 citric acid               0.39000
 residual sugar            8.10000
 chlorides                 0.06500
 free sulfur dioxide      41.00000
 total sulfur dioxide    156.00000
 density                   0.99699
 pH                        3.32000
 sulphates                 0.60000
 alcohol                  11.30000
 quality                   6.00000
 Name: 0.75, dtype: float64)
```

In [20]:
```
IQR=q3-q1
IQR
```

Out[20]:
```
fixed acidity            1.30000
volatile acidity         0.17000
citric acid              0.14000
residual sugar           6.30000
chlorides                0.02700
free sulfur dioxide     24.00000
total sulfur dioxide    79.00000
density                  0.00465
pH                       0.21000
sulphates                0.17000
alcohol                  1.80000
quality                  1.00000
dtype: float64
```

In [21]:
```
lower_limit=q1-1.5*IQR
upper_limit=q3+1.5*IQR
lower_limit,upper_limit
```

Out[21]:
```
(fixed acidity            4.450000
 volatile acidity        -0.025000
 citric acid              0.040000
 residual sugar          -7.650000
 chlorides               -0.002500
 free sulfur dioxide    -19.000000
 total sulfur dioxide   -41.500000
 density                  0.985365
 pH                       2.795000
 sulphates                0.175000
 alcohol                  6.800000
 quality                  3.500000
 dtype: float64,
 fixed acidity             9.650000
 volatile acidity          0.655000
 citric acid               0.600000
 residual sugar           17.550000
 chlorides                 0.105500
 free sulfur dioxide      77.000000
 total sulfur dioxide    274.500000
 density                   1.003965
 pH                        3.635000
 sulphates                 0.855000
 alcohol                  14.000000
 quality                   7.500000
 dtype: float64)
```

In [22]:
```
df[(df<lower_limit)|(df>upper_limit)]
```

Out[22]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alco
0	NaN	NaN	NaN	20.7	NaN	NaN	NaN	NaN	NaN	NaN	I
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	I
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	I
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	I
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	I
...	...	...	...	...	...	...	...	...	...	...	
6492	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	I
6493	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	I
6494	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	I
6495	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	I
6496	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	I

6497 rows × 12 columns

In [23]:
```
df[(df>lower_limit)&(df<upper_limit)]
```

Out[23]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alco
0	7.0	0.270	0.36	NaN	0.045	45.0	170.0	1.00100	3.00	0.45	
1	6.3	0.300	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	
2	8.1	0.280	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	
3	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	
4	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	
...	...	...	...	...	...	...	...	...	...	...	
6492	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	
6493	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	NaN	
6494	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	
6495	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	
6496	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	

6497 rows × 12 columns

In [24]:
```
mean=df.mean()
mean
```

Out[24]:
```
fixed acidity             7.216579
volatile acidity          0.339691
citric acid               0.318722
residual sugar            5.444326
chlorides                 0.056042
free sulfur dioxide      30.525319
total sulfur dioxide    115.744574
density                   0.994697
pH                        3.218395
sulphates                 0.531215
alcohol                  10.491801
quality                   5.818378
dtype: float64
```

In [25]:
```
median=df.median()
median
```

Out[25]:
```
fixed acidity             7.00000
volatile acidity          0.29000
citric acid               0.31000
residual sugar            3.00000
chlorides                 0.04700
free sulfur dioxide      29.00000
total sulfur dioxide    118.00000
density                   0.99489
pH                        3.21000
sulphates                 0.51000
alcohol                  10.30000
quality                   6.00000
dtype: float64
```

In [26]:
```
import numpy as np
df=df.replace(np.NaN,median)
```

In [27]:

```
df.head(50)
```

Out[27]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
0	7.0	0.27	0.36	20.70	0.045	45.0	170.0	1.0010	3.00	0.45	8.
1	6.3	0.30	0.34	1.60	0.049	14.0	132.0	0.9940	3.30	0.49	9.
2	8.1	0.28	0.40	6.90	0.050	30.0	97.0	0.9951	3.26	0.44	10.
3	7.2	0.23	0.32	8.50	0.058	47.0	186.0	0.9956	3.19	0.40	9.
4	7.2	0.23	0.32	8.50	0.058	47.0	186.0	0.9956	3.19	0.40	9.
5	8.1	0.28	0.40	6.90	0.050	30.0	97.0	0.9951	3.26	0.44	10.
6	6.2	0.32	0.16	7.00	0.045	30.0	136.0	0.9949	3.18	0.47	9.
7	7.0	0.27	0.36	20.70	0.045	45.0	170.0	1.0010	3.00	0.45	8.
8	6.3	0.30	0.34	1.60	0.049	14.0	132.0	0.9940	3.30	0.49	9.
9	8.1	0.22	0.43	1.50	0.044	28.0	129.0	0.9938	3.22	0.45	11.
10	8.1	0.27	0.41	1.45	0.033	11.0	63.0	0.9908	2.99	0.56	12.
11	8.6	0.23	0.40	4.20	0.035	17.0	109.0	0.9947	3.14	0.53	9.
12	7.9	0.18	0.37	1.20	0.040	16.0	75.0	0.9920	3.18	0.63	10.
13	6.6	0.16	0.40	1.50	0.044	48.0	143.0	0.9912	3.54	0.52	12.
14	8.3	0.42	0.62	19.25	0.040	41.0	172.0	1.0002	2.98	0.67	9.
15	6.6	0.17	0.38	1.50	0.032	28.0	112.0	0.9914	3.25	0.55	11.
16	6.3	0.48	0.04	1.10	0.046	30.0	99.0	0.9928	3.24	0.36	9.
17	7.0	0.66	0.48	1.20	0.029	29.0	75.0	0.9892	3.33	0.39	12.
18	7.4	0.34	0.42	1.10	0.033	17.0	171.0	0.9917	3.12	0.53	11.
19	6.5	0.31	0.14	7.50	0.044	34.0	133.0	0.9955	3.22	0.50	9.
20	6.2	0.66	0.48	1.20	0.029	29.0	75.0	0.9892	3.33	0.39	12.
21	6.4	0.31	0.38	2.90	0.038	19.0	102.0	0.9912	3.17	0.35	11.
22	6.8	0.26	0.42	1.70	0.049	41.0	122.0	0.9930	3.47	0.48	10.
23	7.6	0.67	0.14	1.50	0.074	25.0	168.0	0.9937	3.05	0.51	9.
24	6.6	0.27	0.41	1.30	0.052	16.0	142.0	0.9951	3.42	0.47	10.
25	7.0	0.25	0.32	9.00	0.046	56.0	245.0	0.9955	3.25	0.50	10.
26	6.9	0.24	0.35	1.00	0.052	35.0	146.0	0.9930	3.45	0.44	10.
27	7.0	0.28	0.39	8.70	0.051	32.0	141.0	0.9961	3.38	0.53	10.
28	7.4	0.27	0.48	1.10	0.047	17.0	132.0	0.9914	3.19	0.49	11.
29	7.2	0.32	0.36	2.00	0.033	37.0	114.0	0.9906	3.10	0.71	12.
30	8.5	0.24	0.39	10.40	0.044	20.0	142.0	0.9974	3.20	0.53	10.
31	8.3	0.14	0.34	1.10	0.042	7.0	47.0	0.9934	3.47	0.40	10.
32	7.4	0.25	0.36	2.05	0.050	31.0	100.0	0.9920	3.19	0.44	10.
33	6.2	0.12	0.34	3.00	0.045	43.0	117.0	0.9939	3.42	0.51	9.

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
34	5.8	0.27	0.20	14.95	0.044	22.0	179.0	0.9962	3.37	0.37	10.
35	7.3	0.28	0.43	1.70	0.080	21.0	123.0	0.9905	3.19	0.42	12.
36	6.5	0.39	0.23	5.40	0.051	25.0	149.0	0.9934	3.24	0.35	10.
37	7.0	0.33	0.32	1.20	0.053	38.0	138.0	0.9906	3.13	0.28	11.
38	7.3	0.24	0.39	17.95	0.057	45.0	149.0	0.9999	3.21	0.36	8.
39	7.3	0.24	0.39	17.95	0.057	45.0	149.0	0.9999	3.21	0.36	8.
40	6.7	0.23	0.39	2.50	0.172	63.0	158.0	0.9937	3.11	0.36	9.
41	6.7	0.24	0.39	2.90	0.173	63.0	157.0	0.9937	3.10	0.34	9.
42	7.0	0.31	0.26	7.40	0.069	28.0	160.0	0.9954	3.13	0.46	9.
43	6.6	0.24	0.27	1.40	0.057	33.0	152.0	0.9934	3.22	0.56	9.
44	6.7	0.23	0.26	1.40	0.060	33.0	154.0	0.9934	3.24	0.56	9.
45	7.4	0.18	0.31	1.40	0.058	38.0	167.0	0.9931	3.16	0.53	10.
46	6.2	0.45	0.26	4.40	0.063	63.0	206.0	0.9940	3.27	0.52	9.
47	6.2	0.46	0.25	4.40	0.066	62.0	207.0	0.9939	3.25	0.52	9.
48	7.0	0.31	0.26	7.40	0.069	28.0	160.0	0.9954	3.13	0.46	9.
49	6.9	0.19	0.35	5.00	0.067	32.0	150.0	0.9950	3.36	0.48	9.

Preprocessing Data for performing Machine learning algorithms

In [28]:
```
from sklearn.preprocessing import LabelEncoder
```

In [29]:
```
#Dividing wine as good and bad by giving the limit for the quality
bins = (2, 6.5, 8)
group_names = ['bad', 'good']
df['quality'] = pd.cut(df['quality'], bins = bins, labels = group_names)
```

In [30]:
```
df['quality'].value_counts()
```

Out[30]:
```
bad     5220
good    1272
Name: quality, dtype: int64
```

In [31]:
```
#Now lets assign a labels to our quality variable
label_quality = LabelEncoder()
```

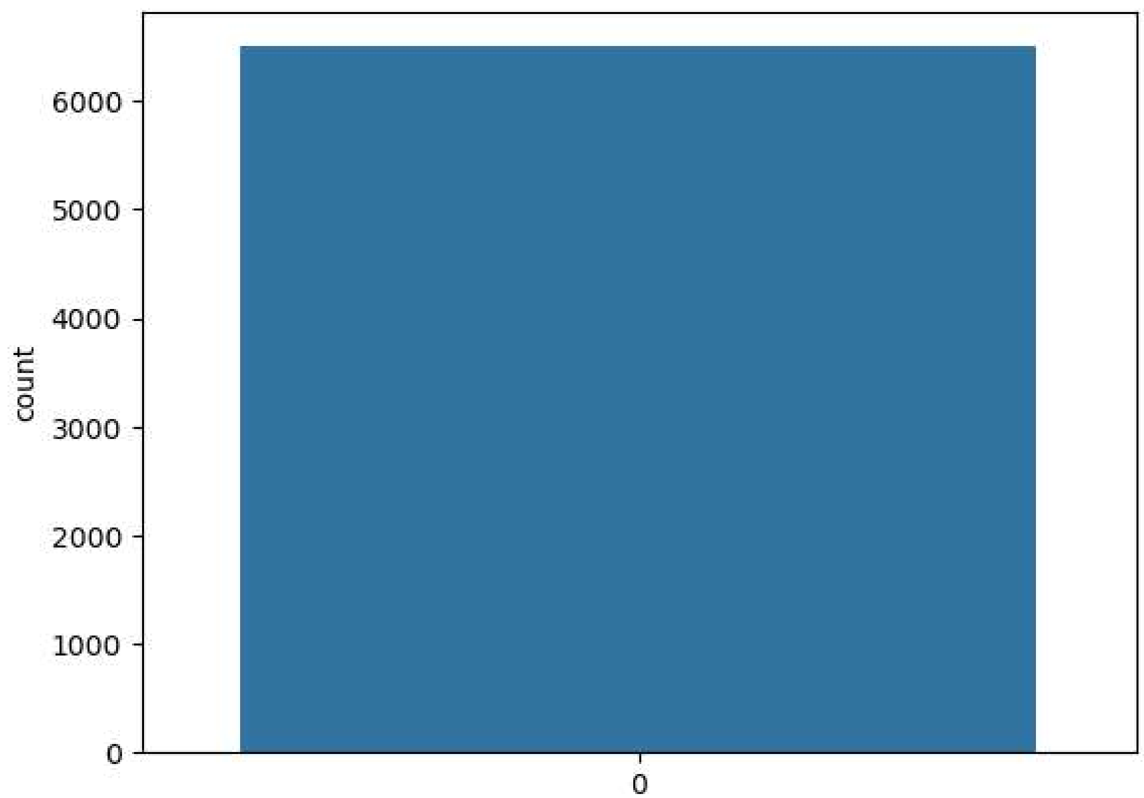
In [32]:
```
#Bad becomes 0 and good becomes 1
df['quality'] = label_quality.fit_transform(df['quality'])
```

In [33]:
```
df['quality'].value_counts()
```

Out[33]:
```
0    5220
1    1272
2       5
Name: quality, dtype: int64
```

In [34]:
```
sns.countplot(df['quality'])
```

Out[34]: <Axes: ylabel='count'>



In [35]:
```
#Now seperate the dataset as response varible and feature varibles
X = df.drop('quality', axis = 1)
y = df['quality']
```

In [36]:
```
#Train and Test solitting of data
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=
```

In [37]:
```
#Apply Standard scaling to get optimized result
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

In [38]:
```
X_train=sc.fit_transform(X_train)
X_test=sc.fit_transform(X_test)
```

Our training and testing data is ready now to perform machine learning algorithm

In [ ]:
```

```

Random Forest Classifier

In [39]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=100)
rfc.fit(X_train,y_train)
pred_rfc=rfc.predict(X_test)
```

In [40]:
```
#Let's see how our model performed
from sklearn.metrics import classification_report
print(classification_report(y_test,pred_rfc))
```

```
              precision    recall  f1-score   support

           0       0.88      0.97      0.92      1047
           1       0.79      0.47      0.59       253

    accuracy                           0.87      1300
   macro avg       0.84      0.72      0.76      1300
weighted avg       0.87      0.87      0.86      1300
```

```
87% accuracy obtained by random forest classifier
```

In [41]:
```
#draw Confusion matrix for the random forest classificatiion
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test,pred_rfc))
```

```
[[1016   31]
 [ 134  119]]
```

Stochastic Gradient Decent Classifier

In [42]:
```
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier(penalty=None)
sgd.fit(X_train, y_train)
pred_sgd= sgd.predict(X_test)
```

In [43]:
```
print(pred_sgd)
```

```
[0 0 0 ... 0 0 0]
```

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

```
              precision    recall  f1-score   support

           0       0.83      0.94      0.88      1047
           1       0.47      0.22      0.30       253

    accuracy                           0.80      1300
   macro avg       0.65      0.58      0.59      1300
weighted avg       0.76      0.80      0.77      1300
```

80% accuracy obtained by stochastic gredient decent classifier

In [45]:
```
print(confusion_matrix(y_test, pred_sgd))
```

```
[[985  62]
 [198  55]]
```

In [ ]:

Support Vector Classifier

In [46]:
```
from sklearn.svm import SVC
svc=SVC()
svc.fit(X_train, y_train)
pred_svc = svc.predict(X_test)
```

In [ ]:

In [47]:
```
print(classification_report(y_test, pred_svc))
```

```
              precision    recall  f1-score   support

           0       0.85      0.97      0.91      1047
           1       0.72      0.30      0.43       253

    accuracy                           0.84      1300
   macro avg       0.79      0.64      0.67      1300
weighted avg       0.83      0.84      0.81      1300
```

In [48]:
```
print(svc.predict(X_test))
```

```
[0 0 0 ... 0 0 0]
```

84 % got from support vector machine

cross validation Score For random forest and SGD

In [49]:
```
from sklearn.model_selection import cross_val_score
rfc_eval = cross_val_score(estimator = rfc, X=X_train, y = y_train)
rfc_eval.mean()
```

Out[49]:
```
0.8822390242096689
```

In [ ]:

In [ ]: