In [1]:

```
import pandas as pd
```

In [2]:

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

Out[2]:

	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 [3]:

```
df.head()
```

Out[3]:

	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 [4]:

```
df.shape
```

Out[4]:

```
(1599, 12)
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [6]:

```
df.isnull().sum()
```

Out[6]:

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

In [7]:

```
df.describe()
```

Out[7]:

	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.(
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.(
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.(
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.(
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.(
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.(

qualtity vs fixed acidity

In [8]:

```
df.columns
```

Out[8]:

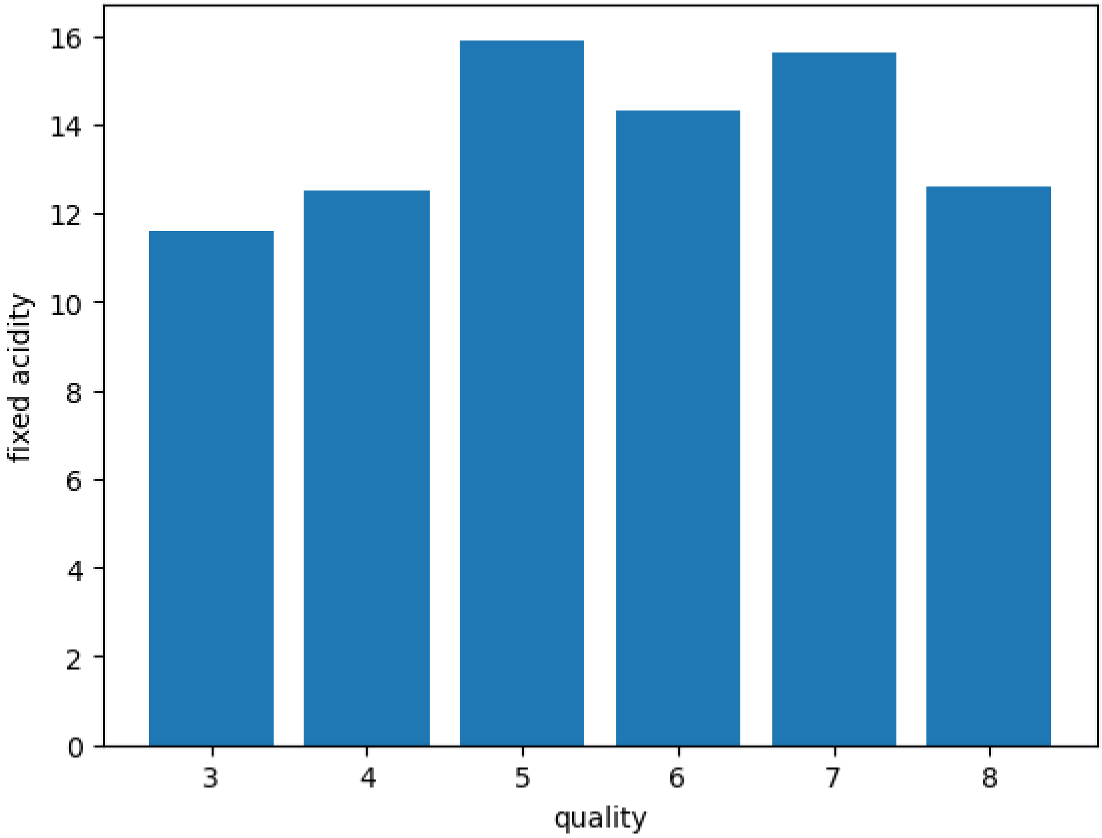
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual suga
r',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'densit
y',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [9]:

```
import matplotlib.pyplot as plt
```

In [10]:

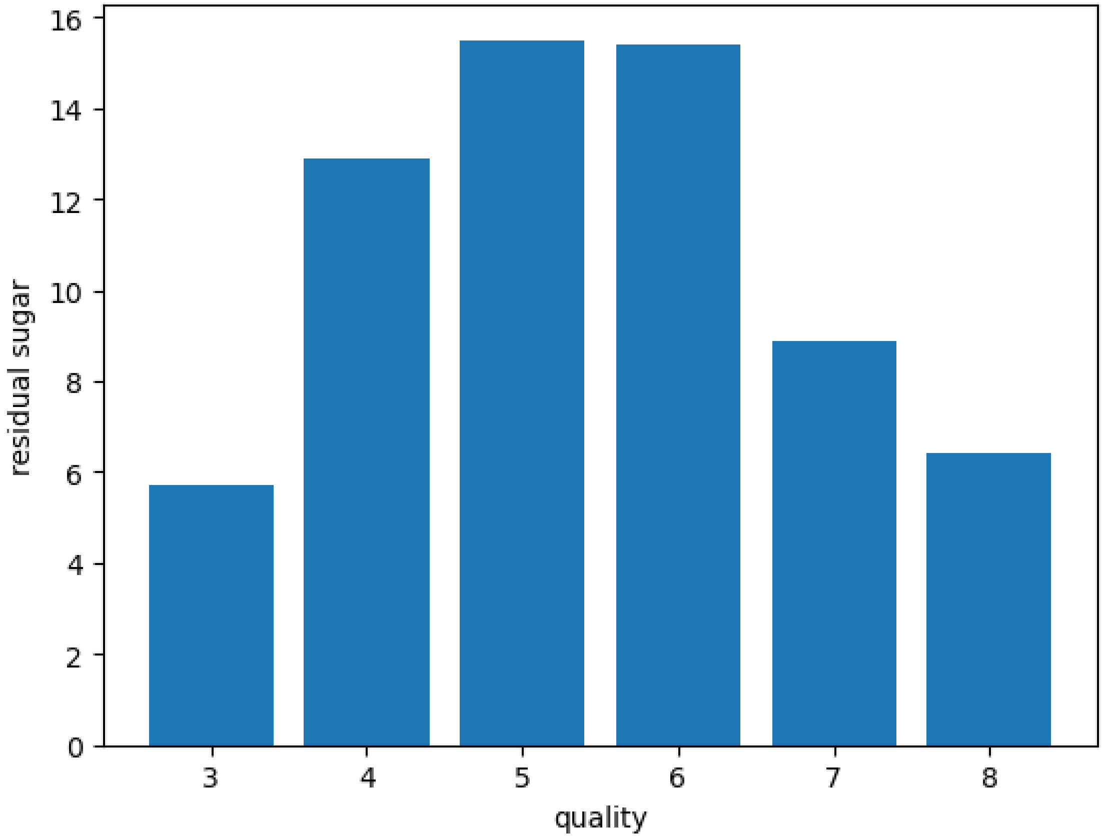
```
plt.bar(df['quality'],df['fixed acidity'])
plt.xlabel('quality')
plt.ylabel('fixed acidity')
plt.show()
```



residual sugar vs quality

In [11]:

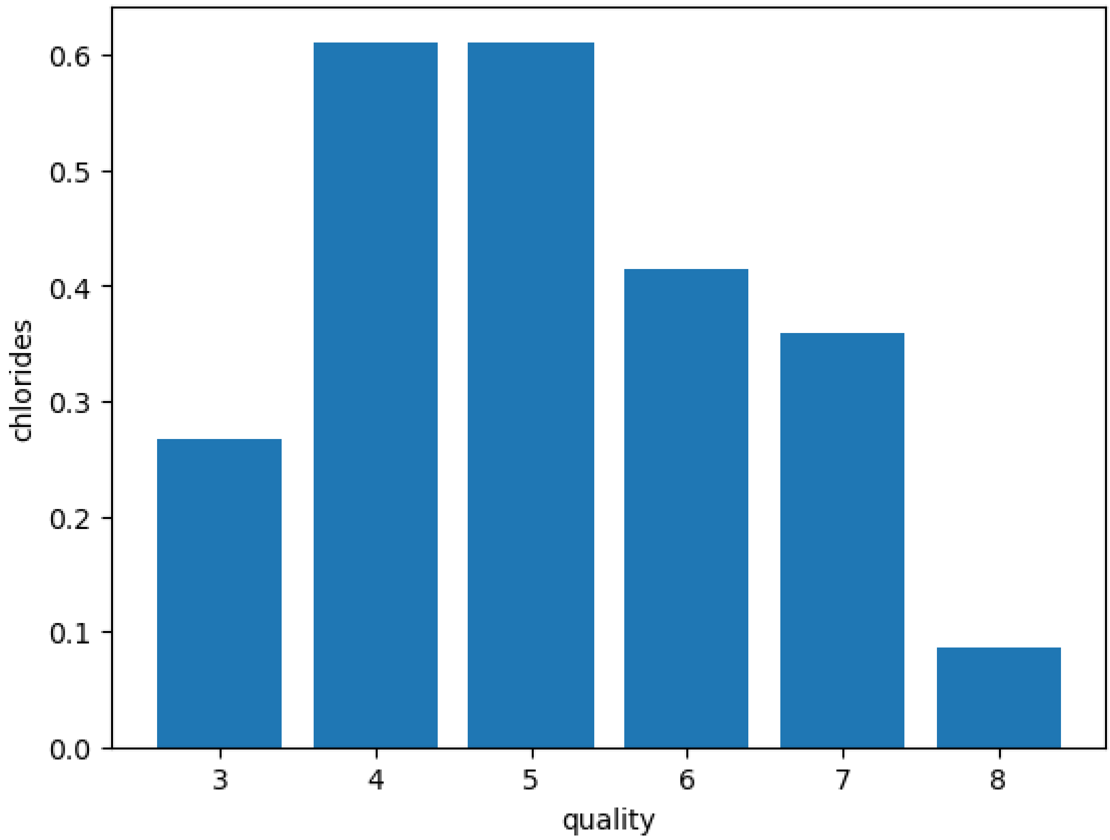
```
plt.bar(df['quality'],df['residual sugar'])
plt.xlabel('quality')
plt.ylabel('residual sugar')
plt.show()
```



choride vs quality

In [12]:

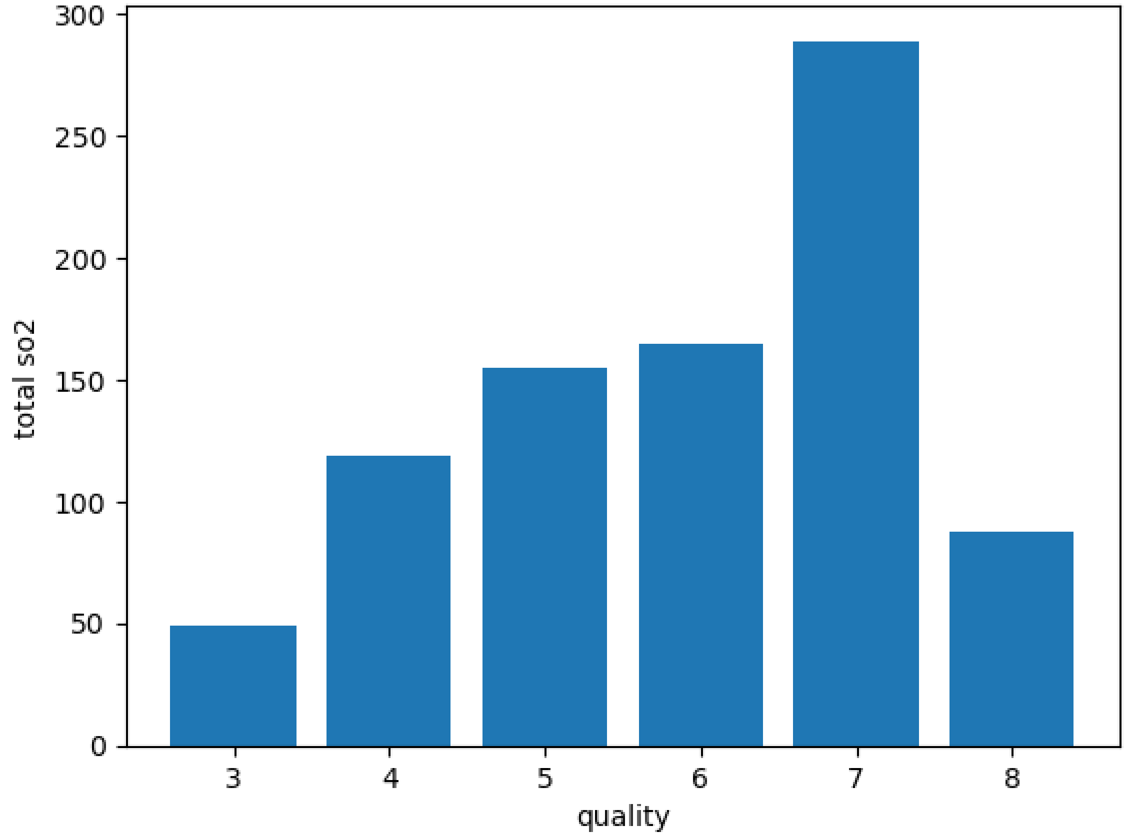
```
plt.bar(df['quality'],df['chlorides'])
plt.xlabel('quality')
plt.ylabel('chlorides')
plt.show()
```



total sulphur dioxide vs quality

In [13]:

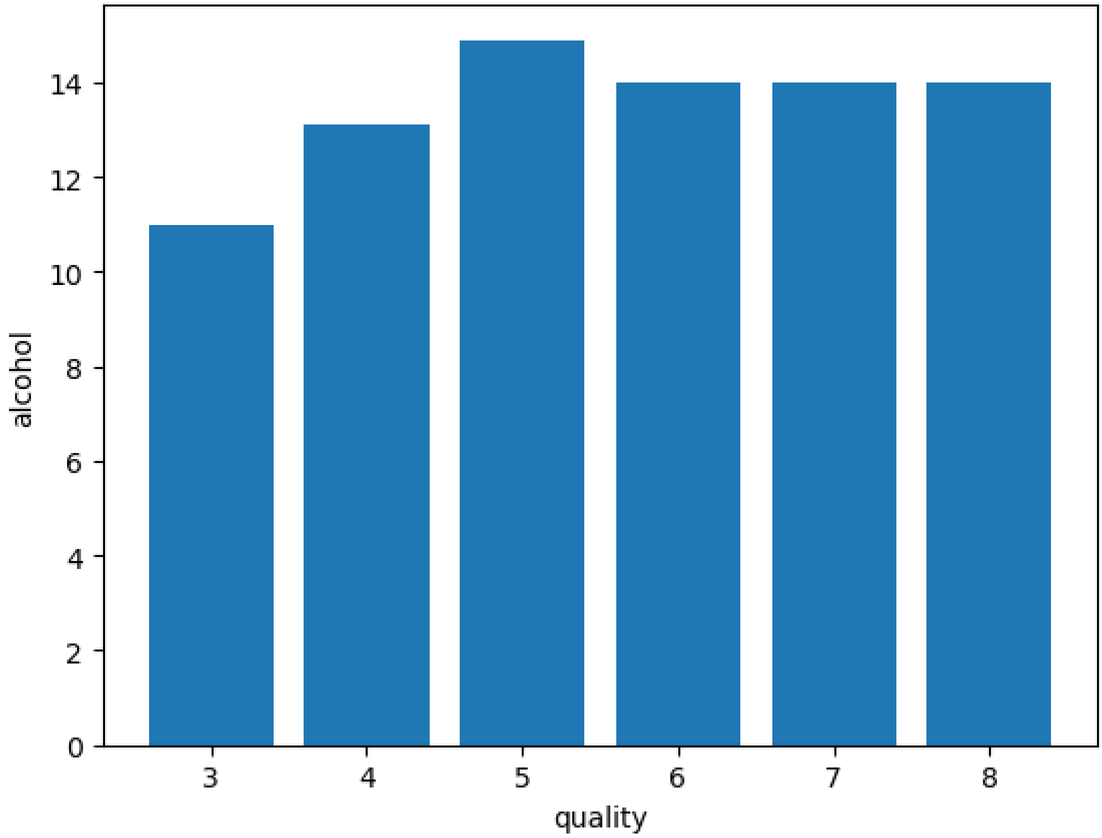
```
plt.bar(df['quality'],df['total sulfur dioxide'])
plt.xlabel('quality')
plt.ylabel('total so2')
plt.show()
```



alcohol vs quality

In [14]:

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



correlation matrix

In [15]:

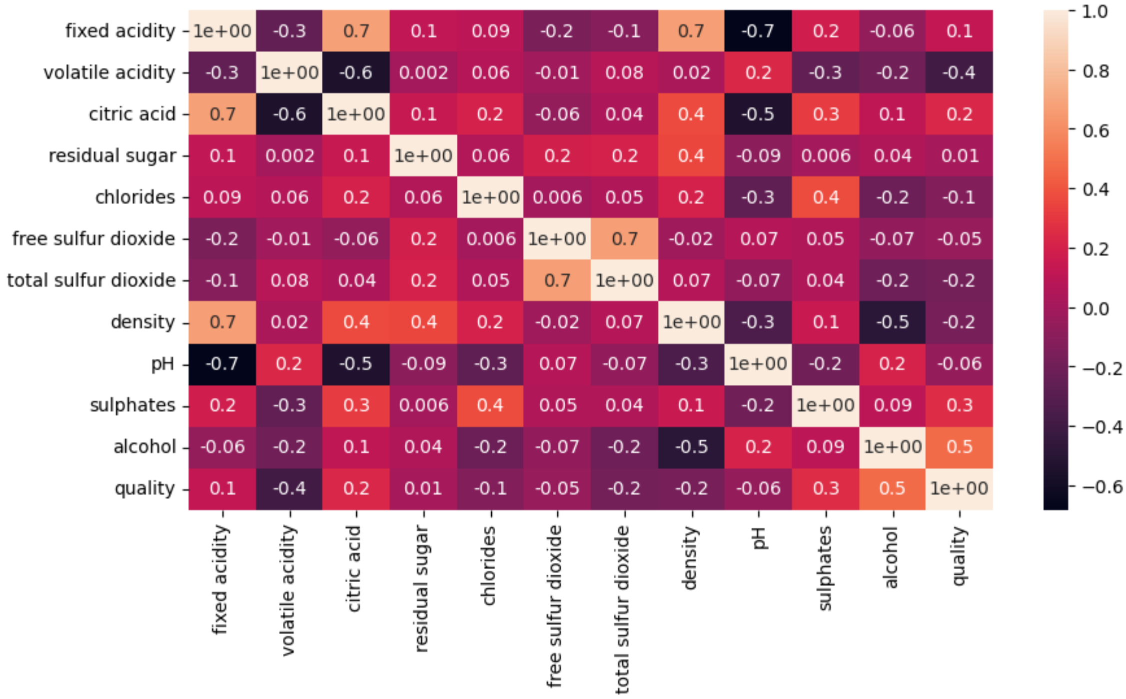
```
import seaborn as sns
```

In [16]:

```
plt.figure(figsize=(10,5))
sns.heatmap(df.corr(),annot=True,fmt='0.1')
```

Out[16]:

```
<AxesSubplot:>
```



Binarization of target variable

In [17]:

```
df['quality'].unique()
```

Out[17]:

```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

In [18]:

```
df['quality']=[1 if x>=7 else 0 for x in df['quality']]
```

In [19]:

```
df['quality'].unique()
```

Out[19]:

```
array([0, 1], dtype=int64)
```

Not handling imbalanced

In [20]:

```
df['quality'].value_counts()
```

Out[20]:

```
0    1382
1     217
Name: quality, dtype: int64
```

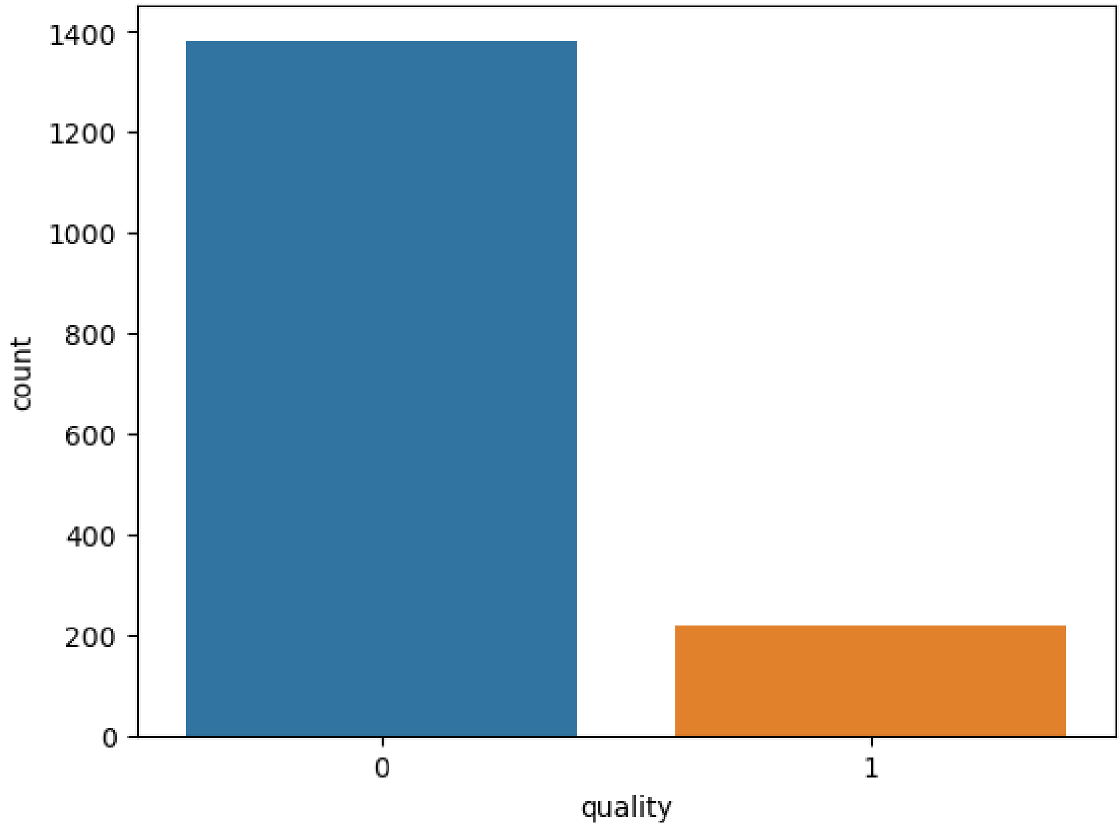
In [21]:

```
sns.countplot(df['quality'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: Futu
reWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterp
retation.
  warnings.warn(
```

Out[21]:

```
<AxesSubplot:xlabel='quality', ylabel='count'>
```



handling imbalanced Dataset

In [22]:

```
from imblearn.over_sampling import SMOTE
```

```
---------------------------------------------------------------------------
-
ModuleNotFoundError                       Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_12656\793248694.py in <module>
----> 1 from imblearn.over_sampling import SMOTE

ModuleNotFoundError: No module named 'imblearn'
```

In [ ]:

```
X_res,y_res=SMOTE().fit_resample(X,y)
```

In [ ]:

```
y_res.value_counts()
```

In [ ]:

```

```

In [ ]:

```

```

store features matrix in X and Response in vector y

In [ ]:

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

split data into train and test

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X_res,y_res,test_size=0.2,random_state=24
```

feature scaling

In [ ]:

```
from sklearn.preprocessing import StandardScaler
```

In [ ]:

```
st=StandardScaler()
X_train=st.fit_transform(X_train)
X_test=st.transform(X_test)
```

In [ ]:

```
X_train
```

Applying PCA

In [ ]:

```
from sklearn.decomposition import PCA
```

In [ ]:

```
pca=PCA(n_components=0.9)
```

In [ ]:

```
X_train=pca.fit_transform(X_train)
X_test=pca.transform(X_test)
```

In [ ]:

```
sum(pca.explained_variance_ratio_)
```

In [ ]:

```
pca.explained_variance_ratio_
```

Logistic Reg

In [ ]:

```
from sklearn.linear_model import LogisticRegression
```

In [ ]:

```
log=LogisticRegression()
log.fit(X_train,y_train)
```

In [ ]:

```
y_pred1=log.predict(X_test)
```

In [ ]:

```
from sklearn.metrics import accuracy_score
```

In [ ]:

```
accuracy_score(y_test,y_pred1)
```

In [ ]:

```
from sklearn.metrics import precision_score,recall_score,f1_score
```

In [ ]:

```
precision_score(y_test,y_pred1)
```

In [ ]:

```
recall_score(y_test,y_pred1)
```

In [ ]:

```
f1_score(y_test,y_pred1)
```

SVC

In [ ]:

```
from sklearn import svm
```

In [ ]:

```
svm=svm.SVC()
```

In [ ]:

```
svm.fit(X_train,y_train)
```

In [ ]:

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

In [ ]:

```
accuracy_score(y_test,y_pred2)
```

In [ ]:

```
precision_score(y_test,y_pred2)
```

In [ ]:

```
f1_score(y_test,y_pred1)
```

Kneighbour Classifier

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [ ]:

```
knn=KNeighborsClassifier()
```

In [ ]:

```
knn.fit(X_train,y_train)
```

In [ ]:

```
y_pred3=knn.predict(X_test,)
```

In [ ]:

```
accuracy_score(y_test,y_pred3)
```

In [ ]:

```
precision_score(y_test,y_pred3)
```

In [ ]:

```
recall_score(y_test,y_pred3)
```

In [ ]:

```
f1_score(y_test,y_pred3)
```

Decision Tree Classifier

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [ ]:

```
dt=DecisionTreeClassifier()
```

In [ ]:

```
dt.fit(X_train,y_train)
```

In [ ]:

```
y_pred4=dt.predict(X_test)
```

In [ ]:

```
accuracy_score(y_test,y_pred4)
```

In [ ]:

```
precision_score(y_test,y_pred4)
```

In [ ]:

```
f1_score(y_test,y_pred4)
```

Random Forest Classifier

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:

```
rf=RandomForestClassifier()
rf.fit(X_train,y_train)
```

In [ ]:

```
y_pred5=rf.predict(X_test)
```

In [ ]:

```
accuracy_score(y_test,y_pred5)
```

In [ ]:

```
precision_score(y_test,y_pred5)
```

In [ ]:

```
f1_score(y_test,y_pred5)
```

Gradient Boosting Classifer

In [ ]:

```
from sklearn.ensemble import GradientBoostingClassifier
```

In [ ]:

```
gbc=GradientBoostingClassifier()
gbc.fit(X_train,y_train)
```

In [ ]:

```
y_pred6=gbc.predict(X_test)
```

In [ ]:

```
accuracy_score(y_test,y_pred6)
```

In [ ]:

```
precision_score(y_test,y_pred6)
```

In [ ]:

```
f1_score(y_test,y_pred6)
```

In [ ]:

```
final_data=pd.DataFrame({'Models':['LR','SVC','KNN','DT','RF','GBC'],
             'ACC':[accuracy_score(y_test,y_pred1)*100,
                    accuracy_score(y_test,y_pred2)*100,
                    accuracy_score(y_test,y_pred3)*100,
                    accuracy_score(y_test,y_pred4)*100,
                    accuracy_score(y_test,y_pred5)*100,
                    accuracy_score(y_test,y_pred6)*100]})

final_data
```

In [ ]:

```
sns.barplot(final_data['Models'],final_data['ACC'])
```

Save the model

In [ ]:

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

In [ ]:

```
from imblearn.over_sampling import SMOTE
X_res,y_res=SMOTE().fit_resample(X,y)
```

In [ ]:

```
from sklearn.preprocessing import StandardScaler
st=StandardScaler()
X=st.fit_transform(X_res)
```

In [ ]:

```
X=pca.fit_transform(X)
```

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf.fit(X,y_res)
```

save using joblib

In [ ]:

```
import joblib
```

In [ ]:

```
joblib.dump(rf,'wine_quality_pred')
```

In [ ]:

```
model=joblib.load('wine_quality_pred')
model
```

prediction on new data

In [ ]:

```
import pandas as pd
new_data=pd.DataFrame({
    'fixed acidity':7.3,
    'volatile acidity':0.65,
    'citric acid':0.00,
    'residual sugar':1.2,
    'chlorides':0.065,
    'free sulfur dioxide':15.0,
    'total sulfur dioxide':21.0,
    'density':0.9946,
    'pH':3.39,
    'sulphates':0.47,
    'alcohol':10.0,

},index=[0])
```

In [ ]:

```
new_data
```

feature scaling

In [ ]:

```
test=pca.transform(st.transform(new_data))
```

In [ ]:

```
p=model.predict(test)
```

In [ ]:

```
if p[0]==1:
    print("good quality wine")
else:
    print("bad quality wine")
```

GUI

In [ ]:

```
from tkinter import *
from sklearn.preprocessing import StandardScaler
import joblib
```

In [ ]:

```
def show_entry_fields():
    p1=float(e1.get())
    p2=float(e2.get())
    p3=float(e3.get())
    p4=float(e4.get())
    p5=float(e5.get())
    p6=float(e6.get())
    p7=float(e7.get())
    p8=float(e8.get())
    p9=float(e9.get())
    p10=float(e10.get())
    p11=float(e11.get())
    model = joblib.load('wine_quality_prediction')
    result=model.predict(pca.transform(st.transform([[p1,p2,p3,p4,p5,p6,
                           p7,p8,p9,p10,p11]])))

    if result[0] == 0:
        Label(master, text="Bad Quality Wine").grid(row=31)
    else:
        Label(master, text="Good Quality Wine").grid(row=31)


master = Tk()
master.title("Wine Quality Prediction Using Machine Learning")


label = Label(master, text = "Wine Quality Prediction Using ML"
                          , bg = "black", fg = "white"). \
                               grid(row=0,columnspan=2)


Label(master, text="fixed acidity").grid(row=1)
Label(master, text="volatile acidity").grid(row=2)
Label(master, text="citric acid").grid(row=3)
Label(master, text="residual sugar").grid(row=4)
Label(master, text="chlorides").grid(row=5)
Label(master, text="free sulfur dioxide").grid(row=6)
Label(master, text="total sulfur dioxide").grid(row=7)
Label(master, text="density").grid(row=8)
Label(master, text="pH").grid(row=9)
Label(master, text="sulphates").grid(row=10)
Label(master,text="alcohol").grid(row=11)


e1 = Entry(master)
e2 = Entry(master)
e3 = Entry(master)
e4 = Entry(master)
e5 = Entry(master)
e6 = Entry(master)
e7 = Entry(master)
e8 = Entry(master)
e9 = Entry(master)
e10 = Entry(master)
e11 = Entry(master)



e1.grid(row=1, column=1)
```

```
e2.grid(row=2, column=1)
e3.grid(row=3, column=1)
e4.grid(row=4, column=1)
e5.grid(row=5, column=1)
e6.grid(row=6, column=1)
e7.grid(row=7, column=1)
e8.grid(row=8, column=1)
e9.grid(row=9, column=1)
e10.grid(row=10,column=1)
e11.grid(row=11,column=1)


Button(master, text='Predict', command=show_entry_fields).grid()

mainloop()
```

In [ ]:

```
import os
cwd = os.getcwd()
print(cwd)
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```