Wine manufacturing company

Problem statement

A wine manufacturing company is planning to create a new brand and needs to determine the quality of their wine by analyzing several chemical parameters, such as acidity, citric acid content, and others. The goal is to assess whether the wine quality is good or not based on these factors.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
import xgboost as xgb
from sklearn.metrics import accuracy_score, confusion_matrix
import warnings
warnings.filterwarnings('ignore')
```

DATA COLLECTION

In [2]:
```
# Loading the dataset to a usnig Pandas DataFrame
df=pd.read_csv('winequality-red.csv')
```

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

Out[3]:
```
(1599, 12)
```

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

Out[4]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
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
```

DATA ANALYSIS AND VISUALIZATION

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

Out[6]:

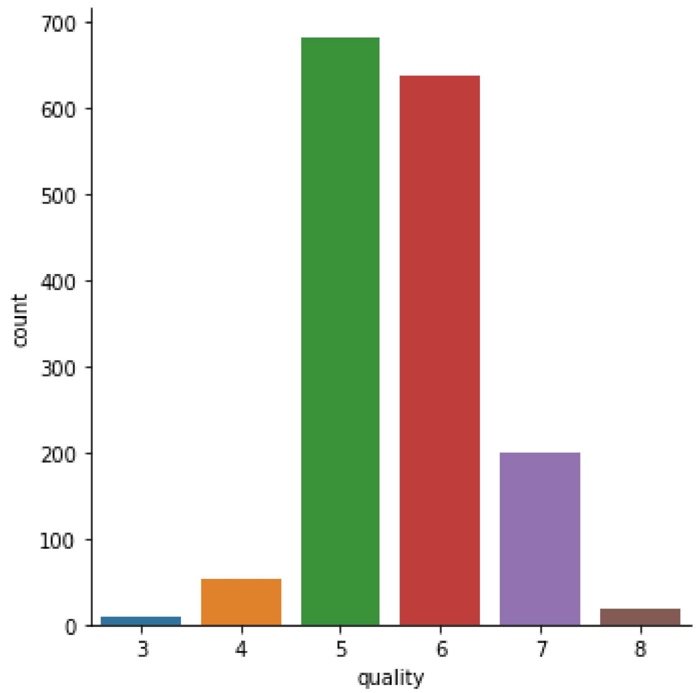
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfu dioxid
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.00000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.4677
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.89532
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.00000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.00000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.00000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.00000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.00000

In [7]:
```
df['quality'].unique()
```

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

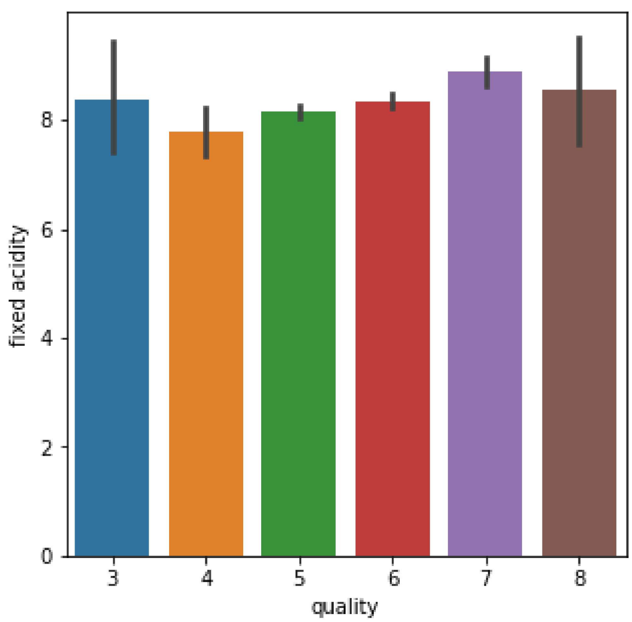
In [8]:
```
# Number of values for each quality
sns.catplot(x='quality',data=df,kind='count')
```

Out[8]: <seaborn.axisgrid.FacetGrid at 0x1cb97f06048>



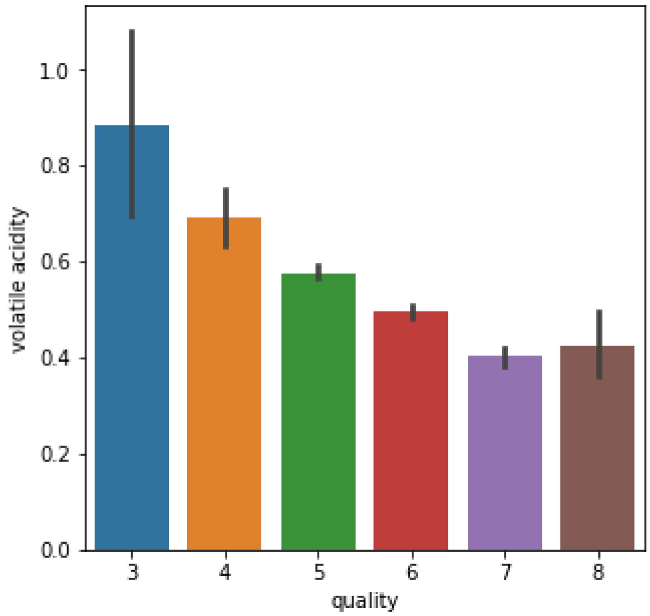
In [9]:
```
#fixed acidity vs quality
plot=plt.figure(figsize=(5,5))
sns.barplot(x='quality',data=df,y='fixed acidity')
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb9867e648>



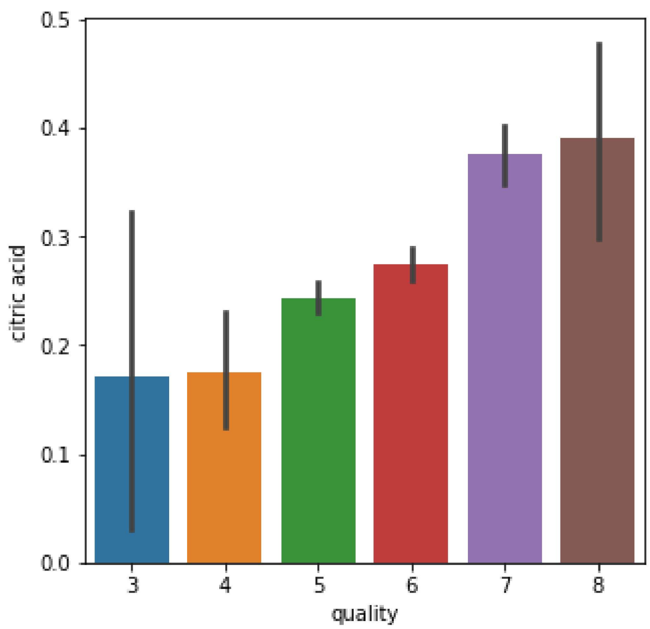
In [10]:
```
# volatile acidity vs quality
plot=plt.figure(figsize=(5,5))
sns.barplot(x='quality',data=df,y='volatile acidity')
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb9867ea48>



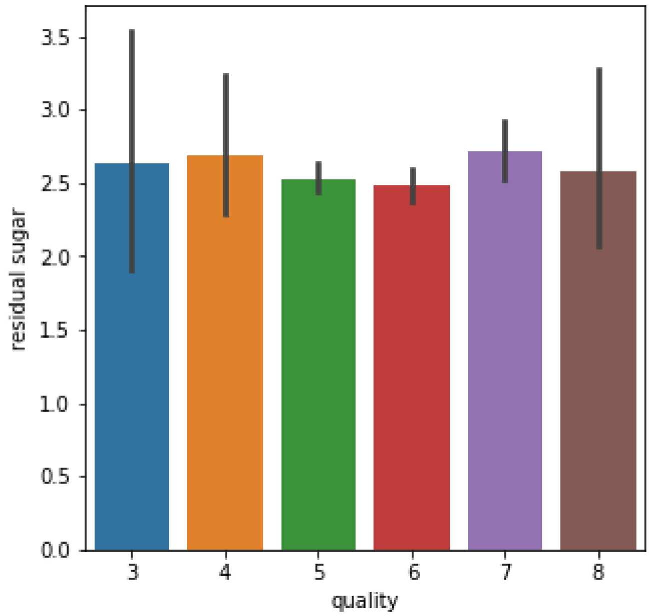
In [11]:
```
# citric acid vs quality
plot=plt.figure(figsize=(5,5))
sns.barplot(x='quality',data=df,y='citric acid')
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb987b7248>



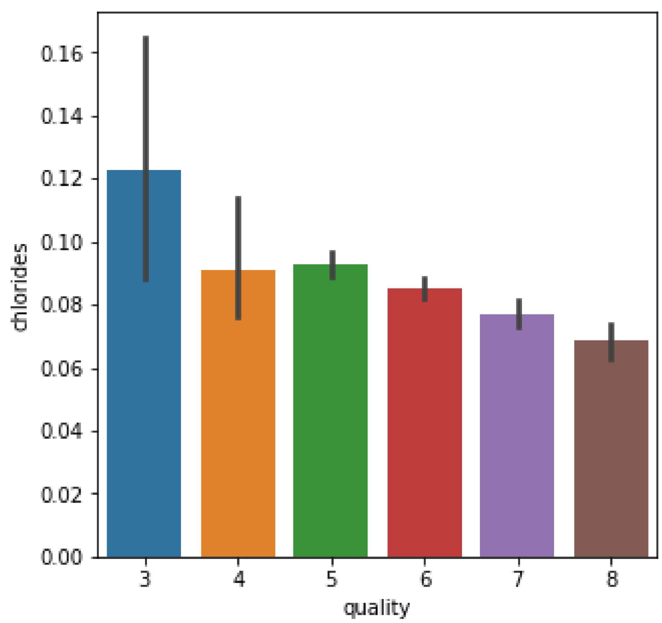
In [12]:
```
# residual sugar vs quality
plot=plt.figure(figsize=(5,5))
sns.barplot(x='quality',data=df,y='residual sugar')
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb987a7dc8>



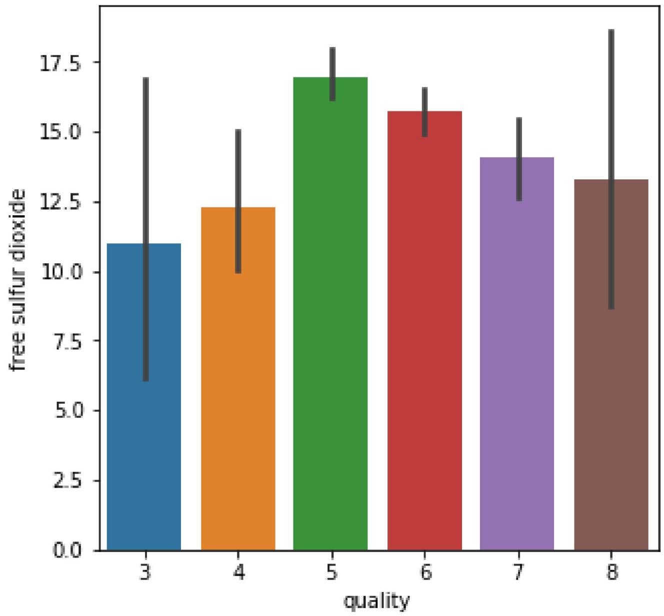
In [13]:
```
# chlorides vs quality
plot=plt.figure(figsize=(5,5))
sns.barplot(x='quality',data=df,y='chlorides')
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb98831a88>



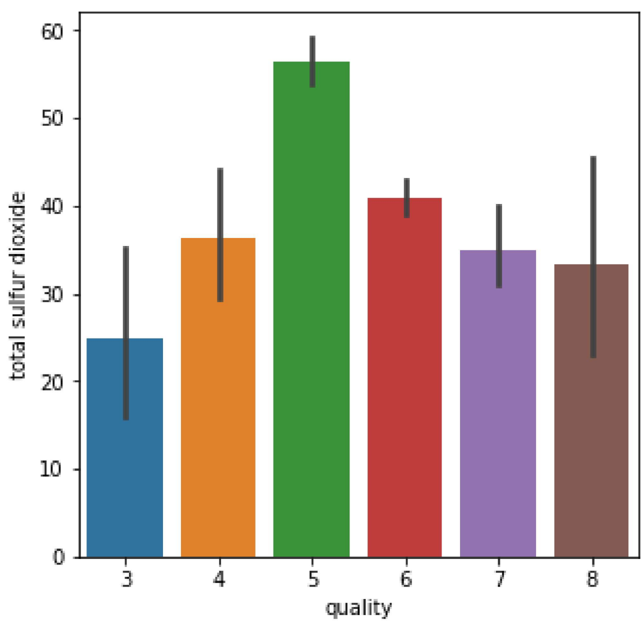
In [14]:
```
# free sulfur dioxide vs quality
plot=plt.figure(figsize=(5,5))
sns.barplot(x='quality',data=df,y='free sulfur dioxide')
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb98928488>



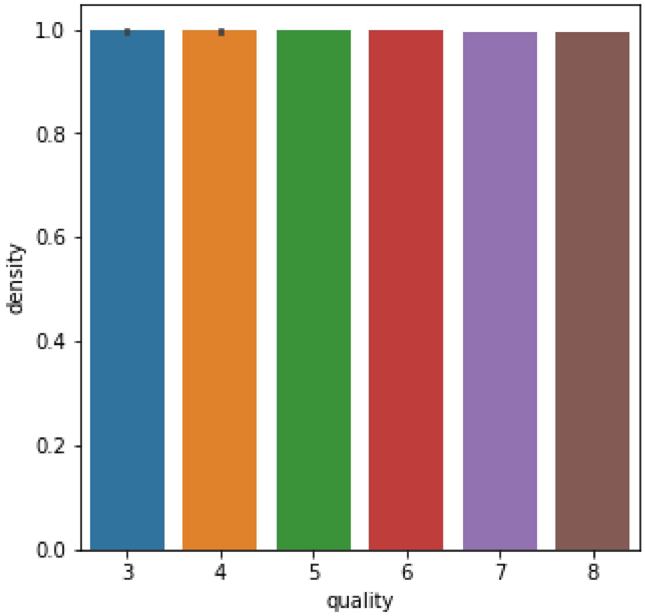
In [15]:
```
#total sulfur dioxide vs quality
plot=plt.figure(figsize=(5,5))
sns.barplot(x='quality',data=df,y='total sulfur dioxide')
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb989b8108>



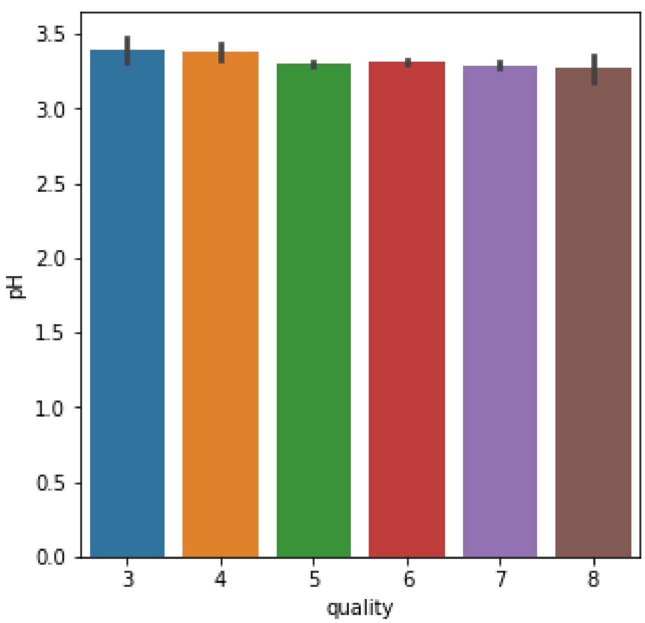
In [16]:
```
#density vs quality
plot=plt.figure(figsize=(5,5))
sns.barplot(x='quality',data=df,y='density')
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb98a23b48>



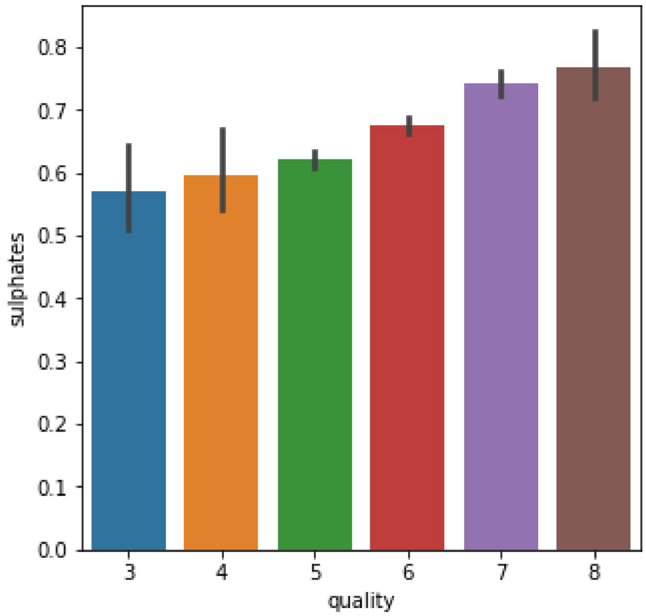
In [17]:
```
# ph vs quality
plot=plt.figure(figsize=(5,5))
sns.barplot(x='quality',data=df,y='pH')
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb98a239c8>



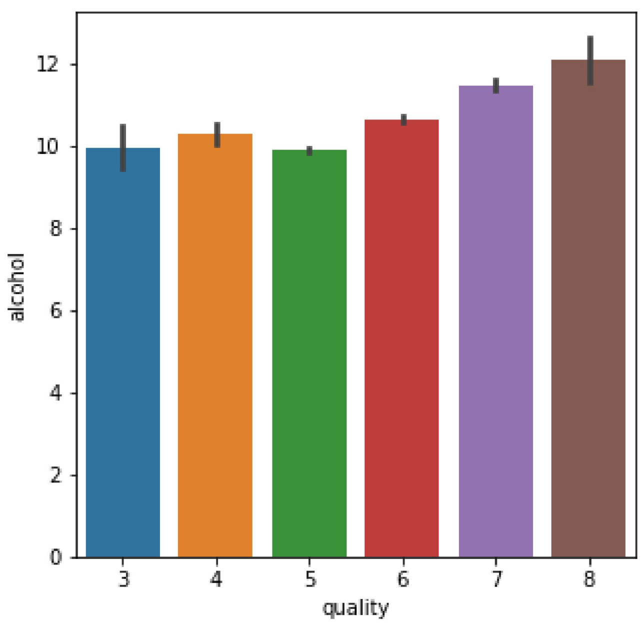
In [18]:
```
# quality vs sulphates
plot=plt.figure(figsize=(5,5))
sns.barplot(x='quality',data=df,y='sulphates')
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb98697888>



In [19]:
```
# quality vs alcohol
plot=plt.figure(figsize=(5,5))
sns.barplot(x='quality',data=df,y='alcohol')
```

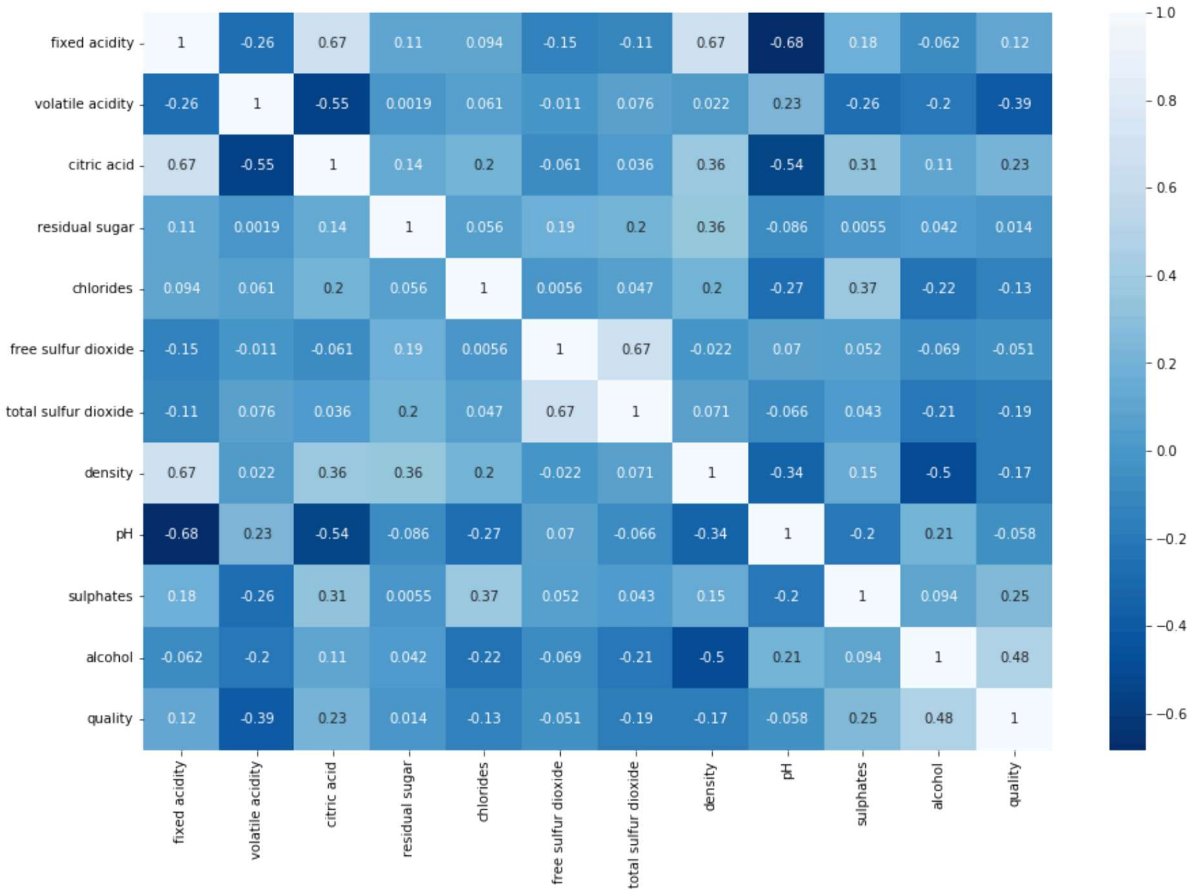
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb98bd3448>



In [20]:
```
corr=df.corr()
```

In [21]:
```
plot=plt.figure(figsize=(15,10))
sns.heatmap(corr,cmap="Blues_r",annot=True)
```

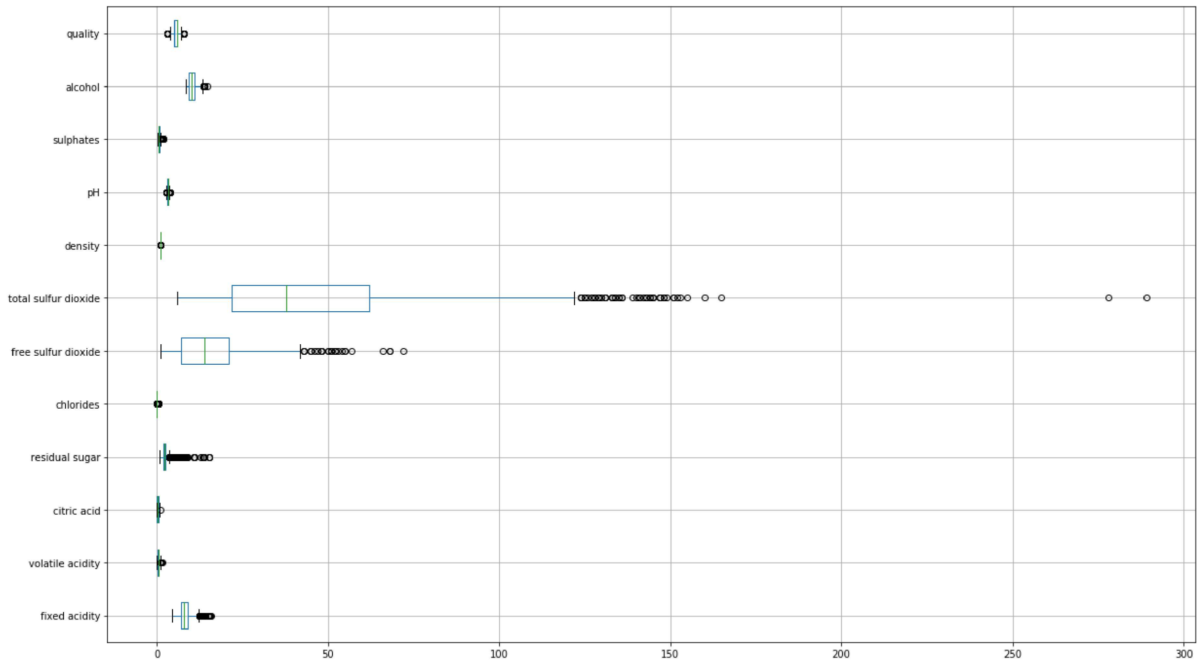
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb986610c8>



DATA PREPROCESSING

In [22]:
```
#identifing the outliers using boxplot
plt.figure(figsize=(20,12))
df.boxplot(vert=0)
```

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb9a19df88>



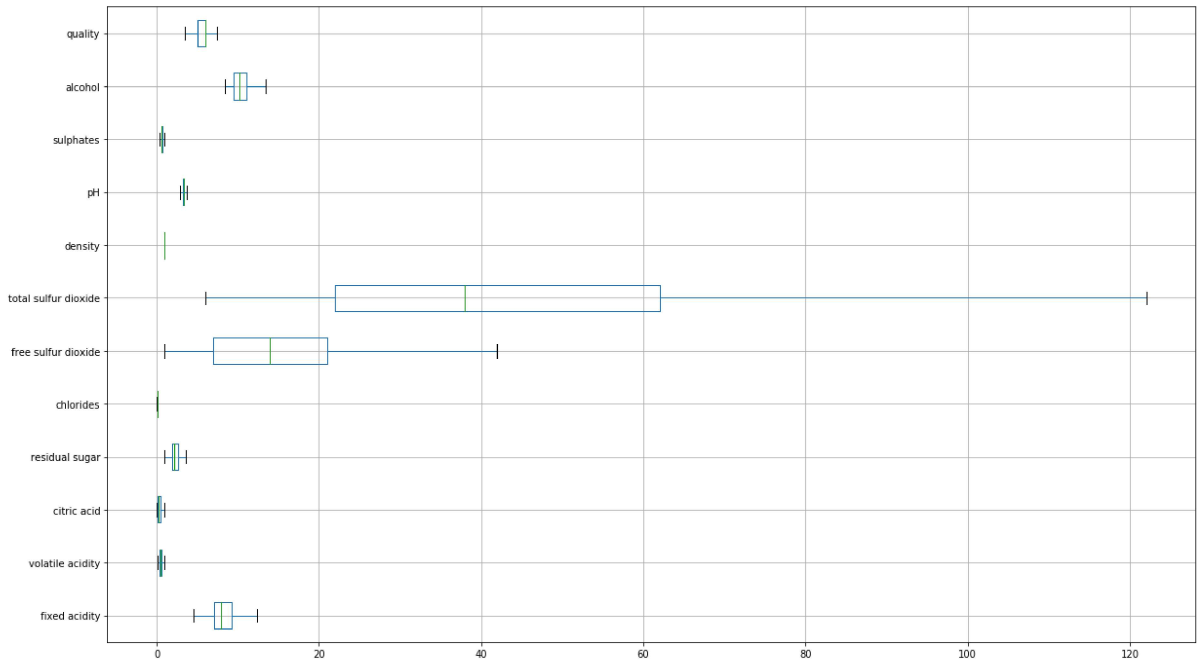
In [23]:
```
# Building user definrd functions for removing outliers

def remove_outlier(col):
    sorted(col)
    Q1,Q3 = np.percentile(col,[25,75])
    IQR = Q3 - Q1
    lower_range = Q1 - (1.5 * IQR)
    upper_range = Q3 + (1.5 * IQR)
    return lower_range, upper_range
```

In [24]:
```
#Using for loop for removing outliers in all the columns
for column in df.columns:
    lower,upper = remove_outlier(df[column])
    df[column] = np.where(df[column]>upper,upper,df[column])
    df[column] = np.where(df[column]<lower,lower,df[column])
```

In [25]:
```
# Identification of Outliers using boxplot
plt.figure(figsize=(20,12))
df.boxplot(vert = 0)
```

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb98ebd308>



In [26]:
```
X=df.drop('quality',axis=1)
```

In [27]:
```
X.head()
```

Out[27]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4

Label binarization

In [28]:
```
Y=df['quality'].apply(lambda y_value: 1 if y_value>=7 else 0)
```

In [29]:
```
Y.head(10)
```

Out[29]:
```
0    0
1    0
2    0
3    0
4    0
5    0
6    0
7    1
8    1
9    0
Name: quality, dtype: int64
```

Feature Importance

In [30]:
```
classifier=ExtraTreesClassifier()
classifier.fit(X,Y)
score=classifier.feature_importances_
print(score)
```

```
[0.07565615 0.10427869 0.09061692 0.07319236 0.07534924 0.06795276
 0.08347502 0.08340898 0.06572693 0.11620645 0.16413649]
```

Train test split

In [31]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=
7)
```

In [32]:
```
print(Y.shape,Y_train.shape,Y_test.shape)
```

```
(1599,) (1119,) (480,)
```

In [33]:
```
print(X.shape, X_train.shape,X_test.shape)
```

```
(1599, 11) (1119, 11) (480, 11)
```

Model Training

Random Forest Classifier

In [34]:
```
RFC=RandomForestClassifier()
RFC.fit(X_train, Y_train)
Y_pred=RFC.predict(X_test)
```

In [35]:
```
CM=confusion_matrix(Y_test,Y_pred)
CM
```

Out[35]:
```
array([[400,  17],
       [ 33,  30]], dtype=int64)
```

In [36]:
```
Accuracy=accuracy_score(Y_test,Y_pred)
print('Accuracy:',Accuracy)
```

```
Accuracy: 0.8958333333333334
```

Logistic Regression

In [37]:
```
LR=LogisticRegression()
LR.fit(X_train, Y_train)
Y_pred=LR.predict(X_test)
```

In [38]:
```
CM=confusion_matrix(Y_test,Y_pred)
CM
```

Out[38]:
```
array([[401,  16],
       [ 44,  19]], dtype=int64)
```

In [39]:
```
Accuracy=accuracy_score(Y_test,Y_pred)
print('Accuracy:',Accuracy)
```

```
Accuracy: 0.875
```

KNN

In [40]:
```
KNN=KNeighborsClassifier()
KNN.fit(X_train,Y_train)
Y_pred=KNN.predict(X_test)
```

In [41]:
```
CM=confusion_matrix(Y_test,Y_pred)
CM
```

Out[41]:
```
array([[399,  18],
       [ 47,  16]], dtype=int64)
```

In [42]:
```
Accuracy=accuracy_score(Y_test,Y_pred)
print('Accuracy:',Accuracy)
```

```
Accuracy: 0.8645833333333334
```

SVC

In [43]:
```
SVC=SVC()
SVC.fit(X_train,Y_train)
Y_pred=SVC.predict(X_test)
```

In [44]:
```
CM=confusion_matrix(Y_test,Y_pred)
CM
```

Out[44]:
```
array([[417,   0],
       [ 63,   0]], dtype=int64)
```

In [45]:
```
Accuracy=accuracy_score(Y_test,Y_pred)
print('Accuracy:',Accuracy)
```

```
Accuracy: 0.86875
```

Decision Tree

In [46]:
```
DTC=DecisionTreeClassifier()
DTC.fit(X_train,Y_train)
Y_pred=DTC.predict(X_test)
```

In [47]:
```
CM=confusion_matrix(Y_test,Y_pred)
CM
```

Out[47]:
```
array([[378,  39],
       [ 20,  43]], dtype=int64)
```

In [48]:
```
Accuracy=accuracy_score(Y_test,Y_pred)
print('Accuracy:',Accuracy)
```

```
Accuracy: 0.8770833333333333
```

GaussianNB

In [49]:
```
GNB=GaussianNB()
GNB.fit(X_train,Y_train)
Y_pred=GNB.predict(X_test)
```

In [50]:
```
CM=confusion_matrix(Y_test,Y_pred)
CM
```

Out[50]:
```
array([[362,  55],
       [ 20,  43]], dtype=int64)
```

In [51]:
```
Accuracy=accuracy_score(Y_test,Y_pred)
print('Accuracy:',Accuracy)
```

```
Accuracy: 0.84375
```

Xgboost

In [52]:
```
XGB=GaussianNB()
XGB.fit(X_train,Y_train)
Y_pred=XGB.predict(X_test)
```

In [53]:
```
CM=confusion_matrix(Y_test,Y_pred)
CM
```

Out[53]:
```
array([[362,  55],
       [ 20,  43]], dtype=int64)
```

In [54]:
```
Accuracy=accuracy_score(Y_test,Y_pred)
print('Accuracy:',Accuracy)
```

```
Accuracy: 0.84375
```

Building a predictive system

In [55]:
```
input_data=(7.4,0.7,0.0,1.9,0.076,11.0,34.0,0.9978,3.51,0.56,9.4)

# changing input data to a numpy array
input_data_as_numpy_array=np.asarray(input_data)

# Reshape the data as we are predicting the label for one instance
input_data_reshape=input_data_as_numpy_array.reshape(1,-1)

prediction=RFC.predict(input_data_reshape)

if(prediction[0]==1):
    print('Good Quality Wine')
else:
    print('Bad Quality Wine')
```

```
Bad Quality Wine
```

In [ ]:
```

```