Wine prediction project by Group 3

In [1]:
```
import pandas as pd
import numpy as np
import sklearn
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score
```

Loading the data set into the project.

In [2]:
```
winedata = pd.read_csv("winequality-red.csv")
winedata.head()
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

In [3]:
```
winedata.columns
```

Out[3]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

Checking the correlation for each of the fields

In [4]:
```
winedata.corr
```

Out[4]:
```
<bound method DataFrame.corr of       fixed acidity  volatile acidity  citric acid  resi
dual sugar  chlorides  \
0               7.4             0.700         0.00             1.9      0.076
1               7.8             0.880         0.00             2.6      0.098
2               7.8             0.760         0.04             2.3      0.092
3              11.2             0.280         0.56             1.9      0.075
4               7.4             0.700         0.00             1.9      0.076
...             ...               ...          ...             ...        ...
1594            6.2             0.600         0.08             2.0      0.090
1595            5.9             0.550         0.10             2.2      0.062
1596            6.3             0.510         0.13             2.3      0.076
1597            5.9             0.645         0.12             2.0      0.075
1598            6.0             0.310         0.47             3.6      0.067

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    11.0                  34.0  0.99780  3.51       0.56
```

```
1                       25.0                    67.0  0.99680  3.20         0.68
2                       15.0                    54.0  0.99700  3.26         0.65
3                       17.0                    60.0  0.99800  3.16         0.58
4                       11.0                    34.0  0.99780  3.51         0.56
...                      ...                     ...      ...   ...          ...
1594                    32.0                    44.0  0.99490  3.45         0.58
1595                    39.0                    51.0  0.99512  3.52         0.76
1596                    29.0                    40.0  0.99574  3.42         0.75
1597                    32.0                    44.0  0.99547  3.57         0.71
1598                    18.0                    42.0  0.99549  3.39         0.66

      alcohol  quality
0         9.4        5
1         9.8        5
2         9.8        5
3         9.8        6
4         9.4        5
...       ...      ...
1594     10.5        5
1595     11.2        6
1596     11.0        6
1597     10.2        5
1598     11.0        6

[1599 rows x 12 columns]>
```

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

Bivariate analysis/Graphs

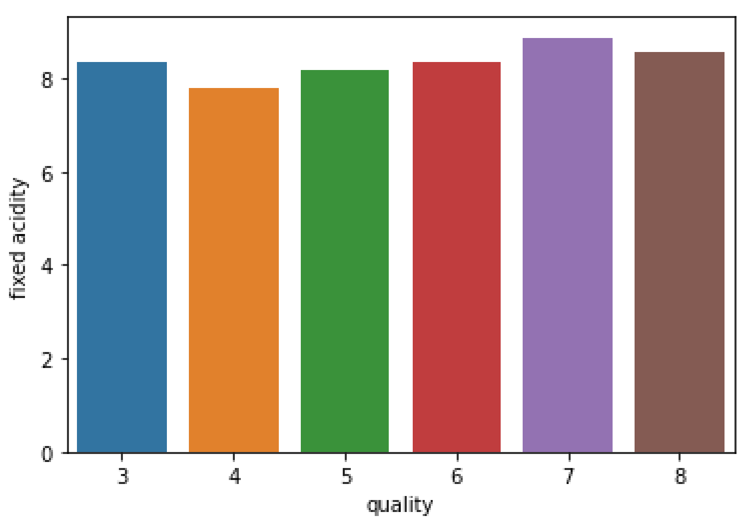
In [5]:
```python
f = plt.figure(figsize = (10,6))
```

```
<Figure size 720x432 with 0 Axes>
```

Quality vs fixed acidity

In [6]:
```python
sns.barplot(x = 'quality', y = 'fixed acidity', ci=None, data = winedata)
```

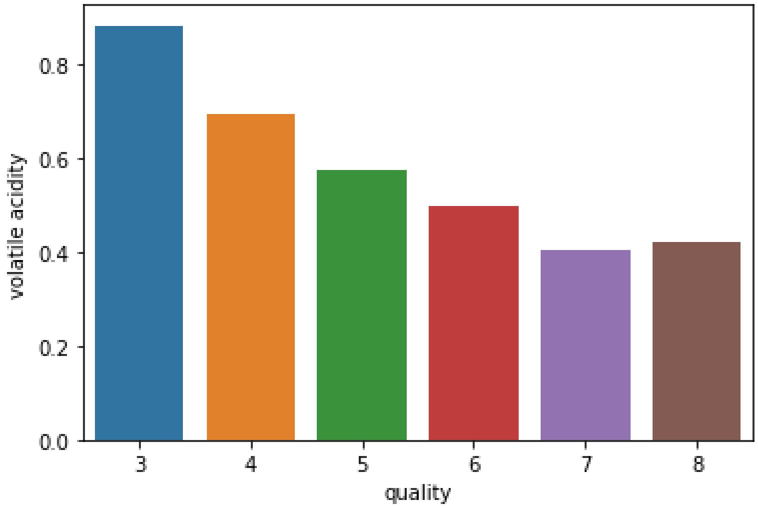
Out[6]:
```
<AxesSubplot:xlabel='quality', ylabel='fixed acidity'>
```



Quality vs Volatile acidity

```
In [7]: sns.barplot(x = 'quality', y = 'volatile acidity', ci=None, data = winedata)
```

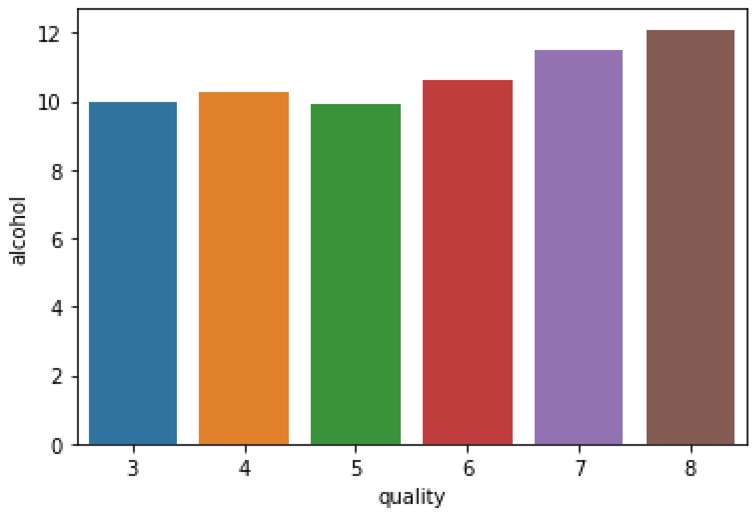
Out[7]: <AxesSubplot:xlabel='quality', ylabel='volatile acidity'>



Quality vs Alcohol

```
In [8]: sns.barplot(x = 'quality', y = 'alcohol', ci=None, data = winedata)
```

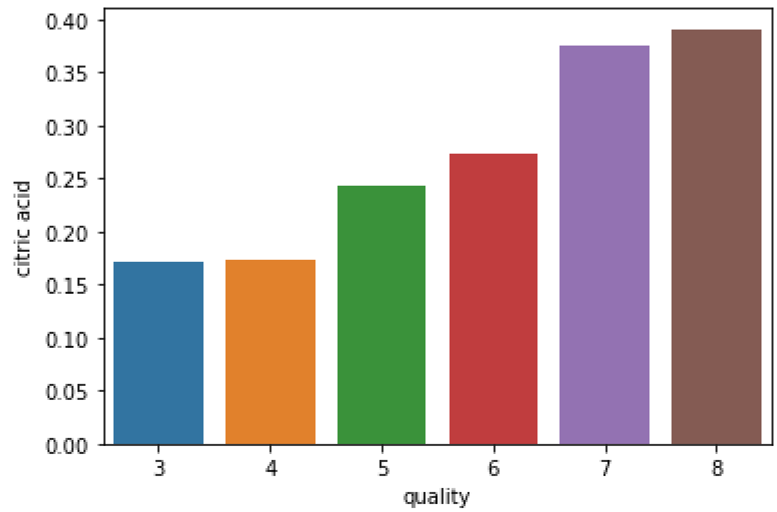
Out[8]: <AxesSubplot:xlabel='quality', ylabel='alcohol'>



Quality vs Citric acid

```
In [9]: sns.barplot(x = 'quality', y = 'citric acid', ci=None, data = winedata)
```

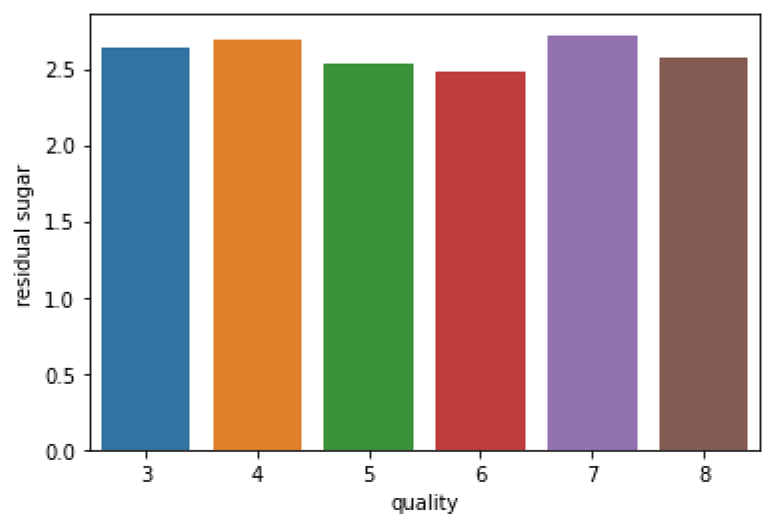
Out[9]: <AxesSubplot:xlabel='quality', ylabel='citric acid'>



Quality vs Residual Sugar

In [10]:
```
sns.barplot(x = 'quality', y = 'residual sugar', ci=None, data = winedata)
```

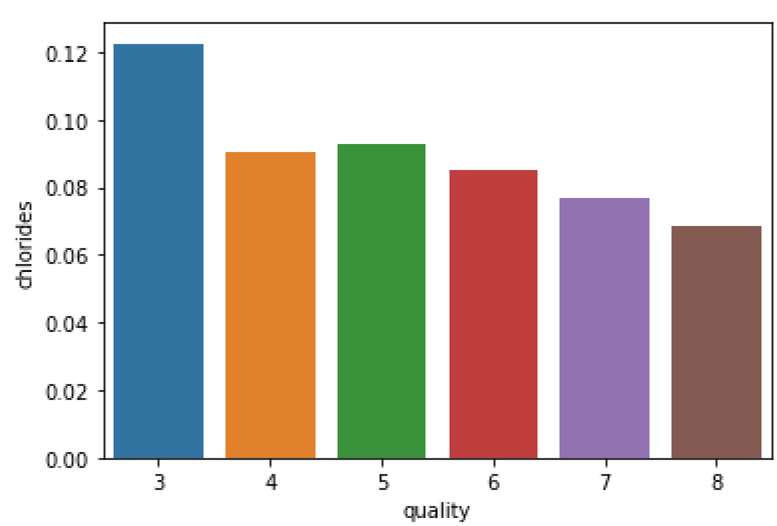
Out[10]:
```
<AxesSubplot:xlabel='quality', ylabel='residual sugar'>
```



Quality vs Chlorides

In [11]:
```
sns.barplot(x = 'quality', y = 'chlorides', ci=None, data = winedata)
```

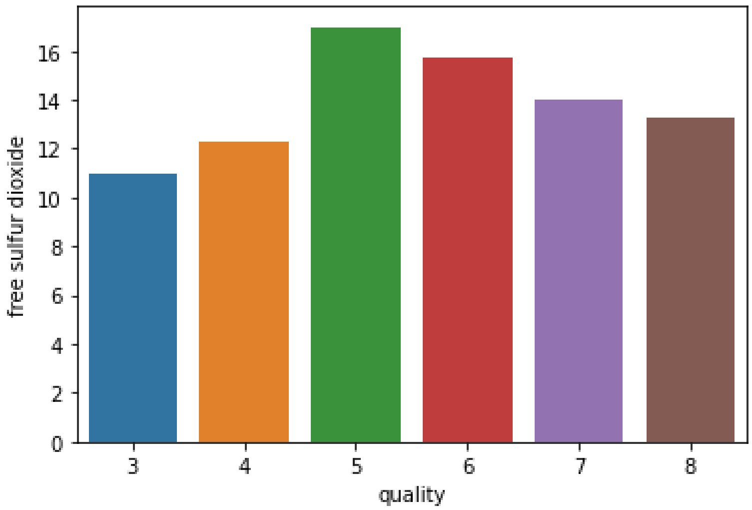
Out[11]:
```
<AxesSubplot:xlabel='quality', ylabel='chlorides'>
```



Quality vs Free Sulfur Dioxide

In [12]:
```
sns.barplot(x = 'quality', y = 'free sulfur dioxide', ci=None, data = winedata)
```

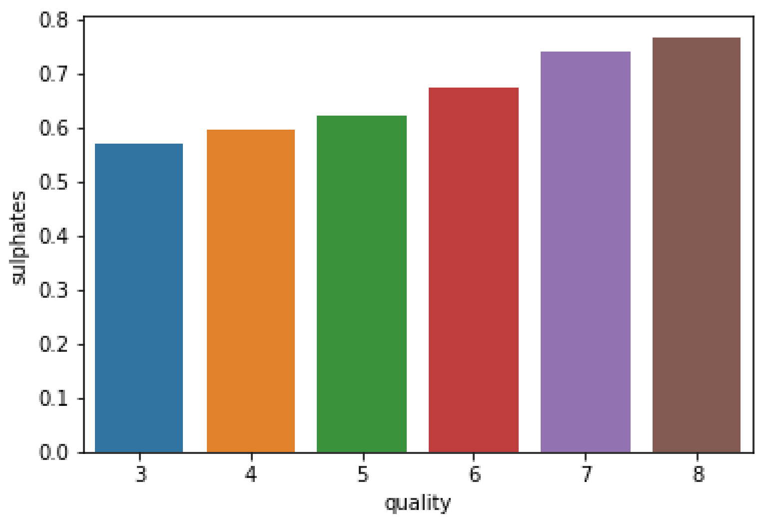
Out[12]: <AxesSubplot:xlabel='quality', ylabel='free sulfur dioxide'>



Quality vs Sulphates

In [13]:
```
sns.barplot(x = 'quality', y = 'sulphates',ci=None, data = winedata)
```

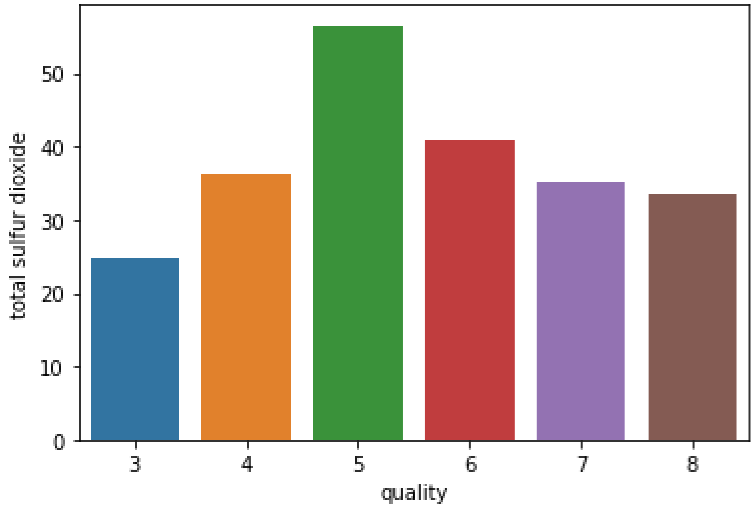
Out[13]: <AxesSubplot:xlabel='quality', ylabel='sulphates'>



Quality vs Total Sulfur Dioxide

In [14]:
```
sns.barplot(x = 'quality', y = 'total sulfur dioxide',ci=None, data = winedata)
```

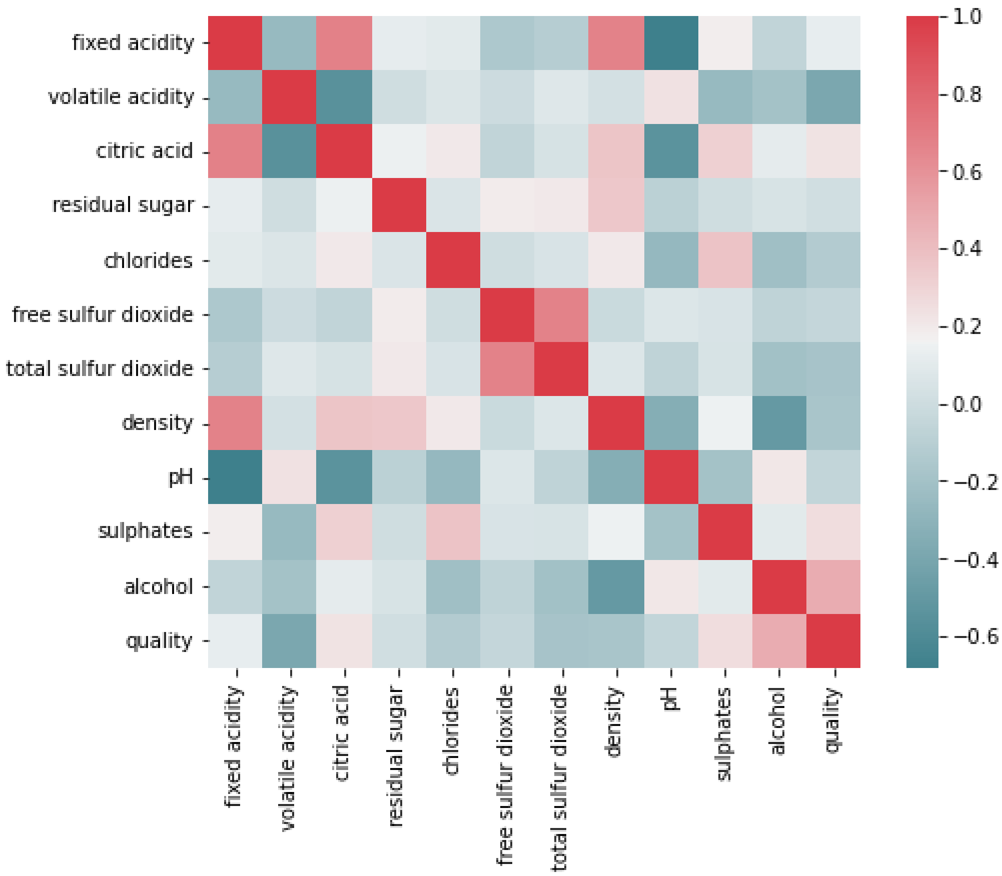
Out[14]: <AxesSubplot:xlabel='quality', ylabel='total sulfur dioxide'>



Checking correlation between attributes using a heat map

In [15]:
```
f, ax = plt.subplots(figsize=(8, 6))
corr = winedata.corr()
sns.heatmap(corr, cmap=sns.diverging_palette(210, 10, as_cmap=True),
            square=True, ax=ax)
```

Out[15]:
```
<AxesSubplot:>
```



From the above correlation plot for the given dataset for wine quality prediction, we can easily see which items are related strongly with each other and which items are related weekly with each other.

The strongly correlated items are :

1.fixed acidity and citric acid. 2.free sulfur dioxide and total sulfur dioxide. 3.fixed acidity and density. 4.

alcohol and quality.

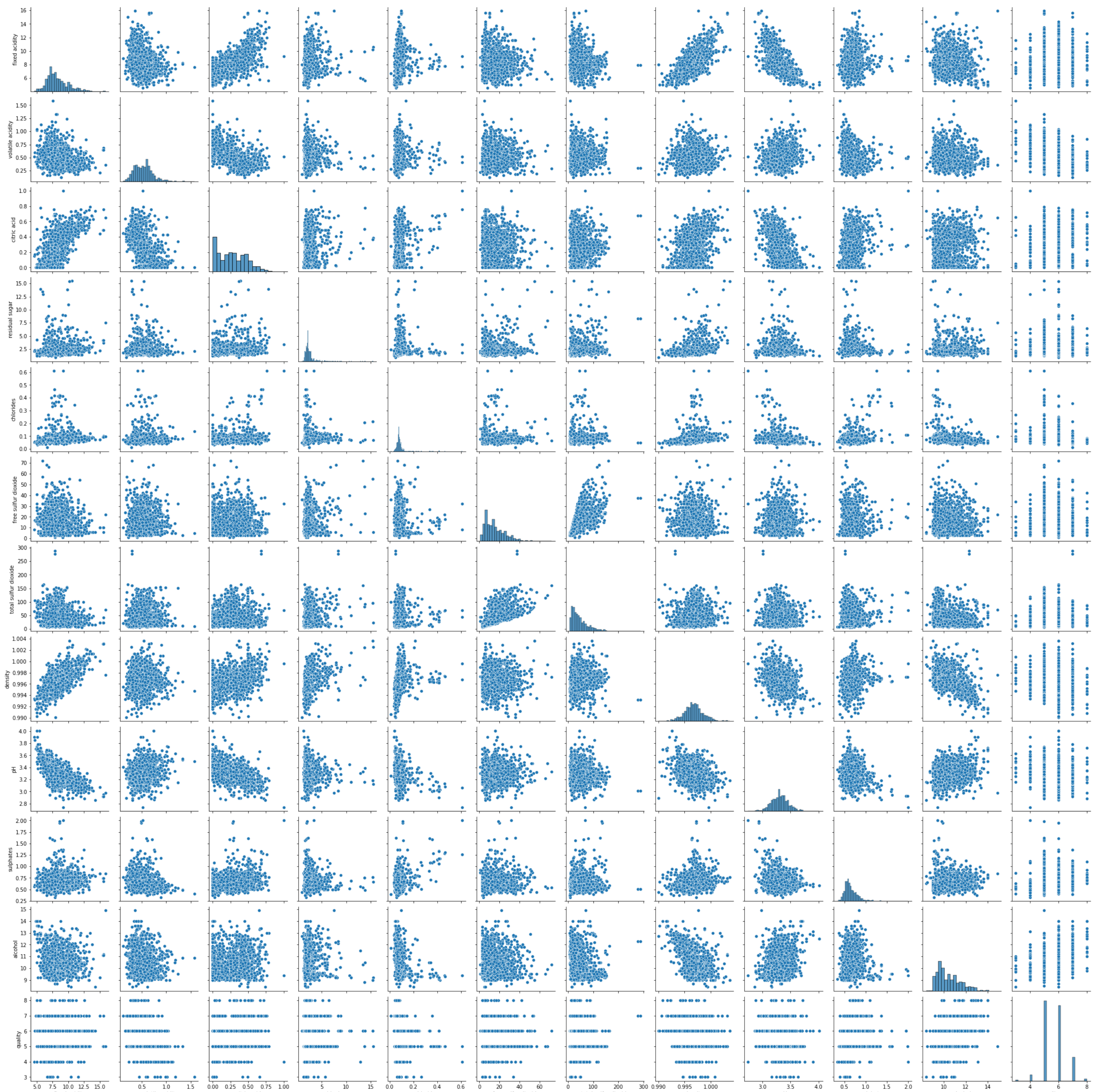
From the above holistic picture of heatmap, it is clearly evident that Alcohol is the most important characteristic of any wine taken

The weak correlated items are :

1.citric acid and volatile acidity. 2.fixed acidity and ph. 3.density and alcohol.

In [16]:
```
sns.pairplot(winedata)
```

Out[16]:
```
<seaborn.axisgrid.PairGrid at 0x1f08205f490>
```



Understanding the data and data pre-processing

In [17]:
```
winedata.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 [18]:
```
winedata.shape
```

Out[18]:
```
(1599, 12)
```

In [19]:
```
winedata.describe()
```

Out[19]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690

In [20]:
```
winedata['quality'].value_counts()
```

Out[20]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

Removing Unnecassary columns from the dataset

As we saw that **volatile acidity, total sulphor dioxide, chlorides, density** are very less related to the dependent variable

quality so even if we remove these columns the accuracy won't be affected that much.

In [21]:
```
#winedata = winedata.drop(['volatile acidity', 'total sulfur dioxide', 'chlorides', 'den
```

In [22]:
```
# checking the shape of the dataset
winedata.shape
```

Out[22]:
```
(1599, 12)
```

In [23]:
```
winedata.describe()
```

Out[23]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690

In [24]:
```
winedata['quality'] = winedata['quality'].map({3 : 'bad', 4 :'bad', 5: 'bad',
                                                6: 'good', 7: 'good', 8: 'good'})

# analyzing the different values present in the dependent variable(quality column)
winedata['quality'].value_counts()
```

Out[24]:
```
good    855
bad     744
Name: quality, dtype: int64
```

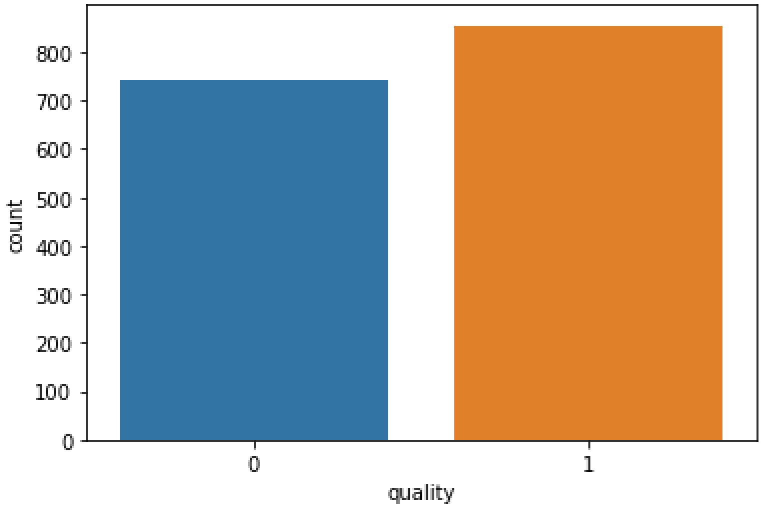
In [25]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

winedata['quality'] = le.fit_transform(winedata['quality'])

winedata['quality'].value_counts
```

Out[25]:
```
<bound method IndexOpsMixin.value_counts of 0       0
1       0
2       0
3       1
4       0
       ..
1594    0
1595    1
1596    1
1597    0
1598    1
Name: quality, Length: 1599, dtype: int32>
```

In [26]:
```
sns.countplot(winedata['quality'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
s the following variable as a keyword arg: x. From version 0.12, the only valid position
al argument will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

Out[26]:
```
<AxesSubplot:xlabel='quality', ylabel='count'>
```



In [27]:
```
# dividing the dataset into dependent and independent variables

x = winedata.iloc[:,:11]
y = winedata.iloc[:,11]

# determining the shape of x and y.
print(x.shape)
print(y.shape)
```

```
(1599, 11)
(1599,)
```

In [94]:
```
# dividing the dataset in training and testing set

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test= train_test_split(x,y,test_size=0.20,random_state=1)

# determining the shapes of training and testing sets
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(1279, 11)
(1279,)
(320, 11)
(320,)
```

In [95]:
```
# standard scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

Logistic Regression

In [96]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# creating the model
model = LogisticRegression()

# feeding the training set into the model
model.fit(x_train, y_train)
```

```python
# predicting the results for the test set
y_pred = model.predict(x_test)

# calculating the training and testing accuracies
print("Training accuracy :", model.score(x_train, y_train))
print("Testing accuracy :", model.score(x_test, y_test))

# classification report
print(classification_report(y_test, y_pred))

# confusion matrix
print(confusion_matrix(y_test, y_pred))
```

```
Training accuracy : 0.7513682564503519
Testing accuracy : 0.740625
              precision    recall  f1-score   support

           0       0.74      0.70      0.72       154
           1       0.74      0.78      0.76       166

    accuracy                           0.74       320
   macro avg       0.74      0.74      0.74       320
weighted avg       0.74      0.74      0.74       320

[[108  46]
 [ 37 129]]
```

Support Vector Machine

In [97]:
```python
from sklearn.svm import SVC

# creating the model
model = SVC()

# feeding the training set into the model
model.fit(x_train, y_train)

# predicting the results for the test set
y_pred = model.predict(x_test)

# calculating the training and testing accuracies
print("Training accuracy :", model.score(x_train, y_train))
print("Testing accuracy :", model.score(x_test, y_test))

# classification report
print(classification_report(y_test, y_pred))

# confusion matrix
print(confusion_matrix(y_test, y_pred))
```

```
Training accuracy : 0.8029710711493354
Testing accuracy : 0.75
              precision    recall  f1-score   support

           0       0.74      0.74      0.74       154
           1       0.76      0.76      0.76       166

    accuracy                           0.75       320
   macro avg       0.75      0.75      0.75       320
weighted avg       0.75      0.75      0.75       320

[[114  40]
 [ 40 126]]
```

Decision Tree

In [98]:
```
from sklearn.tree import DecisionTreeClassifier

# creating the model
model = DecisionTreeClassifier()

# feeding the training set into the model
model.fit(x_train, y_train)

# predicting the results for the test set
y_pred = model.predict(x_test)

# calculating the training and testing accuracies
print("Training accuracy :", model.score(x_train, y_train))
print("Testing accuracy :", model.score(x_test, y_test))

# classification report
print(classification_report(y_test, y_pred))

# confusion matrix
print(confusion_matrix(y_test, y_pred))
```

```
Training accuracy : 1.0
Testing accuracy : 0.728125
              precision    recall  f1-score   support

           0       0.73      0.69      0.71       154
           1       0.73      0.76      0.74       166

    accuracy                           0.73       320
   macro avg       0.73      0.73      0.73       320
weighted avg       0.73      0.73      0.73       320

[[107  47]
 [ 40 126]]
```

Random Forest

In [99]:
```
from sklearn.ensemble import RandomForestClassifier

# creating the model
model = RandomForestClassifier()

# feeding the training set into the model
model.fit(x_train, y_train)

# predicting the results for the test set
y_pred = model.predict(x_test)

# calculating the training and testing accuracies
print("Training accuracy :", model.score(x_train, y_train))
print("Testing accuracy :", model.score(x_test, y_test))

# classification report
print(classification_report(y_test, y_pred))

# confusion matrix
print(confusion_matrix(y_test, y_pred))
```

```
Training accuracy : 1.0
Testing accuracy : 0.821875
              precision    recall  f1-score   support
```

```
           0       0.82      0.81      0.81       154
           1       0.82      0.84      0.83       166

    accuracy                           0.82       320
   macro avg       0.82      0.82      0.82       320
weighted avg       0.82      0.82      0.82       320

[[124  30]
 [ 27 139]]
```

Naive Bayes

In [100…

```python
from sklearn.naive_bayes import GaussianNB

# creating the model
model = GaussianNB()

# feeding the training set into the model
model.fit(x_train, y_train)

# predicting the results for the test set
y_pred = model.predict(x_test)

# calculating the training and testing accuracies
print("Training accuracy :", model.score(x_train, y_train))
print("Testing accuracy :", model.score(x_test, y_test))

# classification report
print(classification_report(y_test, y_pred))

# confusion matrix
print(confusion_matrix(y_test, y_pred))
```

```
Training accuracy : 0.7310398749022674
Testing accuracy : 0.73125
              precision    recall  f1-score   support

           0       0.72      0.71      0.72       154
           1       0.74      0.75      0.74       166

    accuracy                           0.73       320
   macro avg       0.73      0.73      0.73       320
weighted avg       0.73      0.73      0.73       320

[[110  44]
 [ 42 124]]
```

Multi Layer Perceptron

In [101…

```python
from sklearn.neural_network import MLPClassifier

# creating the model
model = MLPClassifier(hidden_layer_sizes = (100, 100), max_iter = 150)

# feeding the training data to the model
model.fit(x_train, y_train)

# calculating the accuracies
print("training accuracy :", model.score(x_train, y_train))
print("testing accuracy :", model.score(x_test, y_test))

# classification report
print(classification_report(y_test, y_pred))
```

```
# confusion matrix
print(confusion_matrix(y_test, y_pred))
```

```
training accuracy : 0.9374511336982018
testing accuracy : 0.753125
              precision    recall  f1-score   support

           0       0.72      0.71      0.72       154
           1       0.74      0.75      0.74       166

    accuracy                           0.73       320
   macro avg       0.73      0.73      0.73       320
weighted avg       0.73      0.73      0.73       320

[[110  44]
 [ 42 124]]
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (150) reached and the optimization hasn't converged yet.
  warnings.warn(
```

Stochastic Gradient Descent Classifier

In [102…

```
from sklearn.linear_model import SGDClassifier

# creating the model
model = SGDClassifier(penalty=None)

# feeding the training data to the model
model.fit(x_train, y_train)

# calculating the accuracies
print("training accuracy :", model.score(x_train, y_train))
print("testing accuracy :", model.score(x_test, y_test))

# classification report
print(classification_report(y_test, y_pred))

# confusion matrix
print(confusion_matrix(y_test, y_pred))
```

```
training accuracy : 0.72869429241595
testing accuracy : 0.746875
              precision    recall  f1-score   support

           0       0.72      0.71      0.72       154
           1       0.74      0.75      0.74       166

    accuracy                           0.73       320
   macro avg       0.73      0.73      0.73       320
weighted avg       0.73      0.73      0.73       320

[[110  44]
 [ 42 124]]
```

In [ ]: