imports

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import roc_curve,confusion_matrix, classification_report , accuracy_sc
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import GridSearchCV, cross_val_score
```

In [2]:

```
data=pd.read_csv('QualityPrediction.csv')
data
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	al
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

EDA

In [3]:

```
data.describe()
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total su dio
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000

In [4]:

```
data.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 [5]:

```
data['quality'].value_counts
```

Out[5]:

```
<bound method IndexOpsMixin.value_counts of 0       5
1       5
2       5
3       6
4       5
       ..
1594    5
1595    6
1596    6
1597    5
1598    6
Name: quality, Length: 1599, dtype: int64>
```

Data Visualization

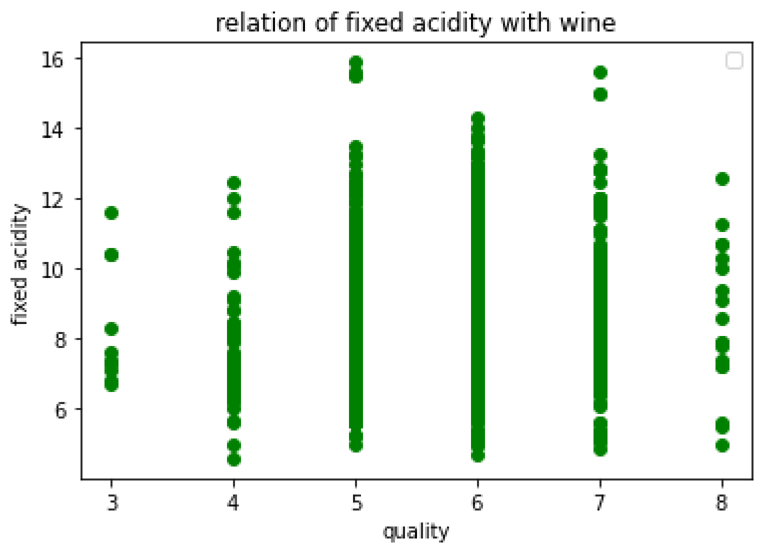
Bivariate Analysis

In [6]:

```
# checking the variation of fixed acidity in the different qualities of wine

plt.scatter(data['quality'], data['fixed acidity'], color = 'green')
plt.title('relation of fixed acidity with wine')
plt.xlabel('quality')
plt.ylabel('fixed acidity')
plt.legend()
plt.show()
```

```
No handles with labels found to put in legend.
```

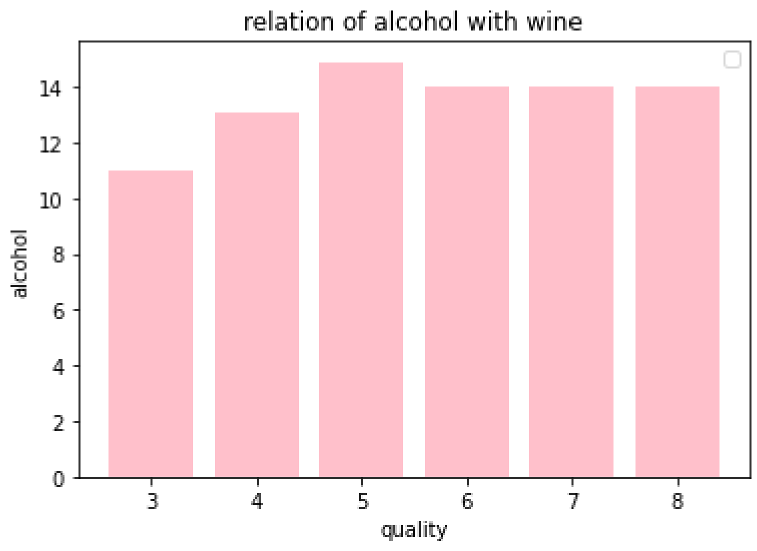


In [7]:

```
# checking the variation of fixed acidity in the different qualities of wine

plt.bar(data['quality'], data['alcohol'], color = 'pink')
plt.title('relation of alcohol with wine')
plt.xlabel('quality')
plt.ylabel('alcohol')
plt.legend()
plt.show()
```

```
No handles with labels found to put in legend.
```



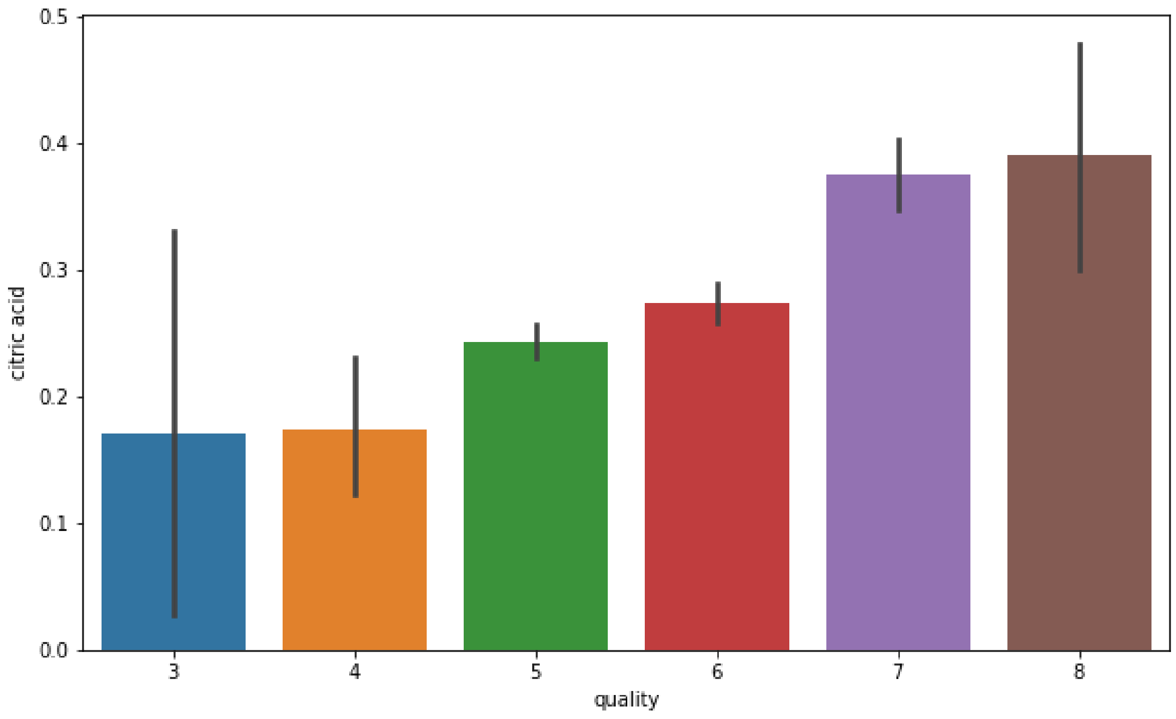
In [8]:

```
import seaborn as sns

fig = plt.figure(figsize = (10,6))
sns.barplot(x = 'quality', y = 'citric acid', data = data)
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x227364308e0>
```

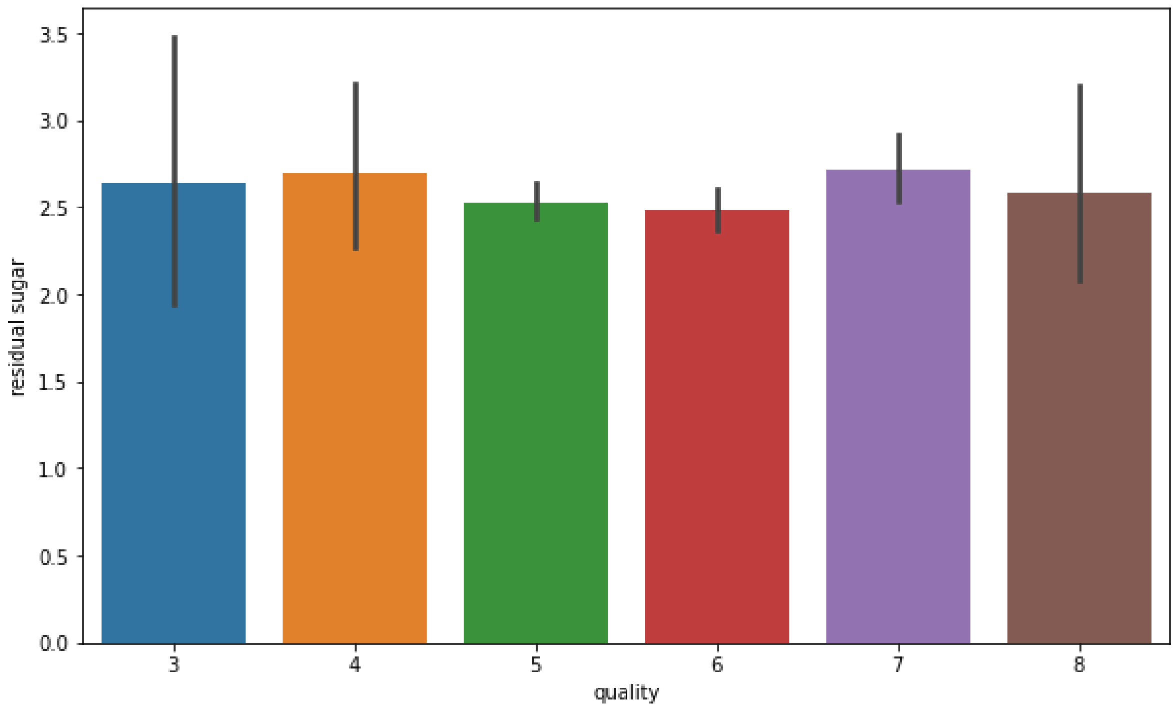


In [9]:

```
fig = plt.figure(figsize = (10,6))
sns.barplot(x = 'quality', y = 'residual sugar', data = data)
```

Out[9]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x22735b17a60>
```

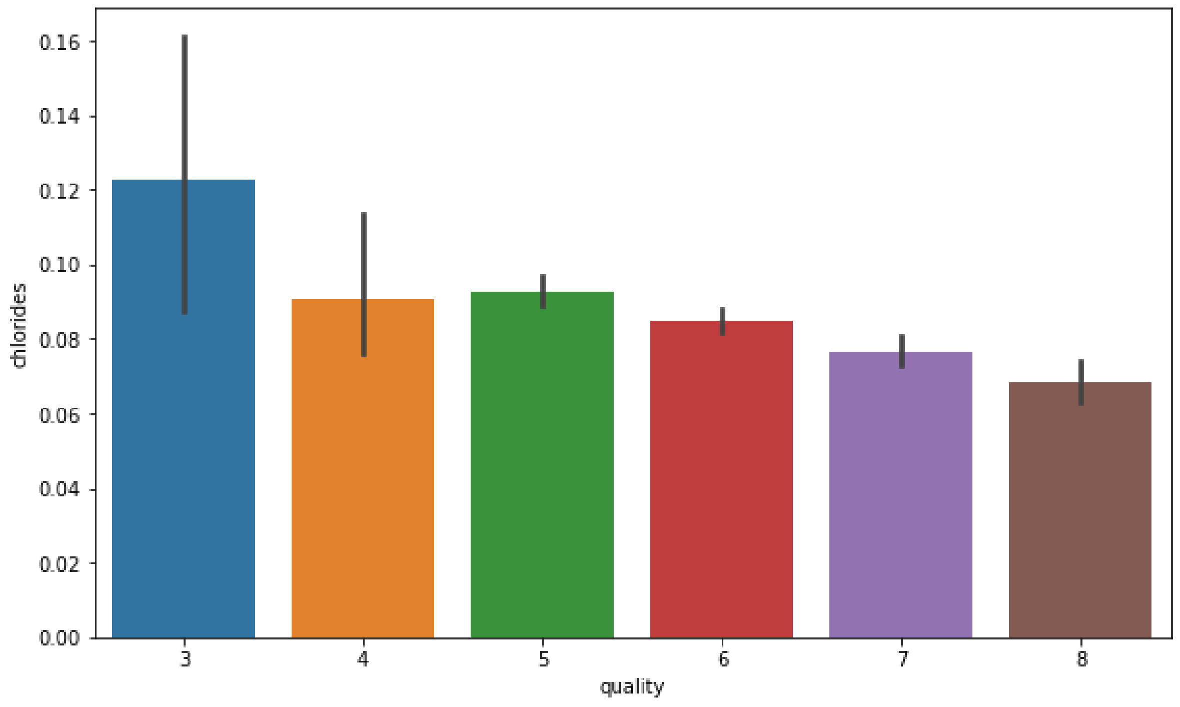


In [10]:

```
fig = plt.figure(figsize = (10,6))
sns.barplot(x = 'quality', y = 'chlorides', data = data)
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2273589a640>
```

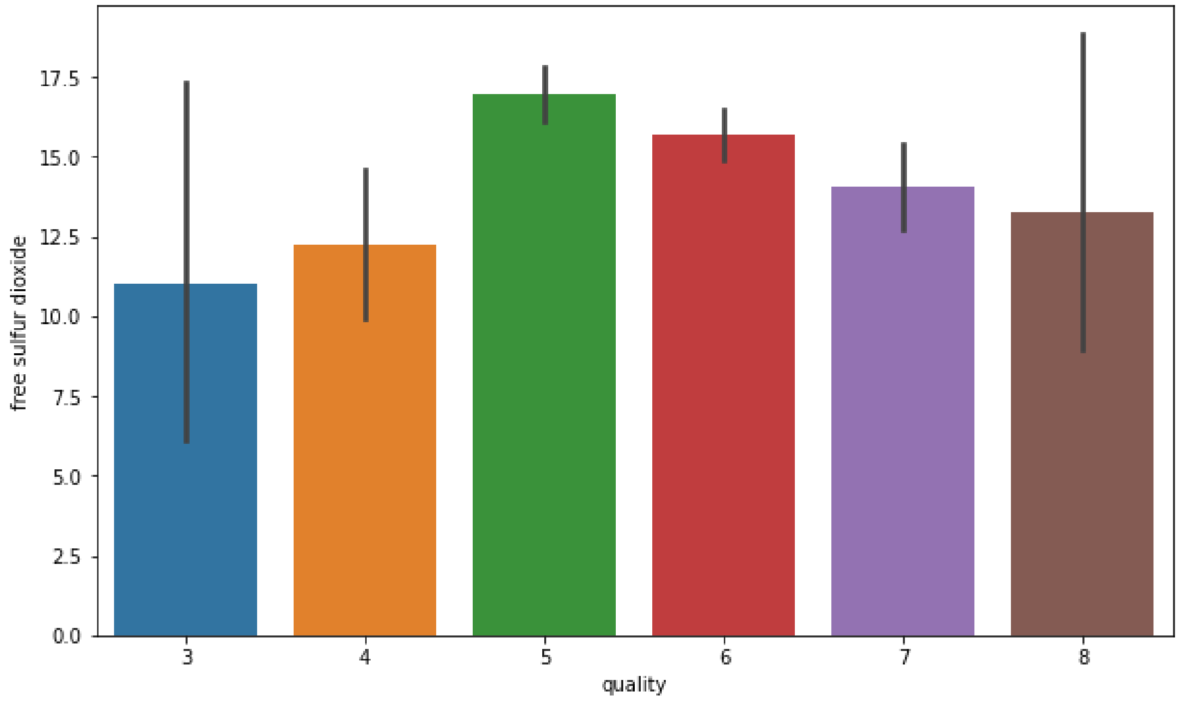


In [11]:

```
fig = plt.figure(figsize = (10,6))
sns.barplot(x = 'quality', y = 'free sulfur dioxide', data = data)
```

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x22736929f40>
```



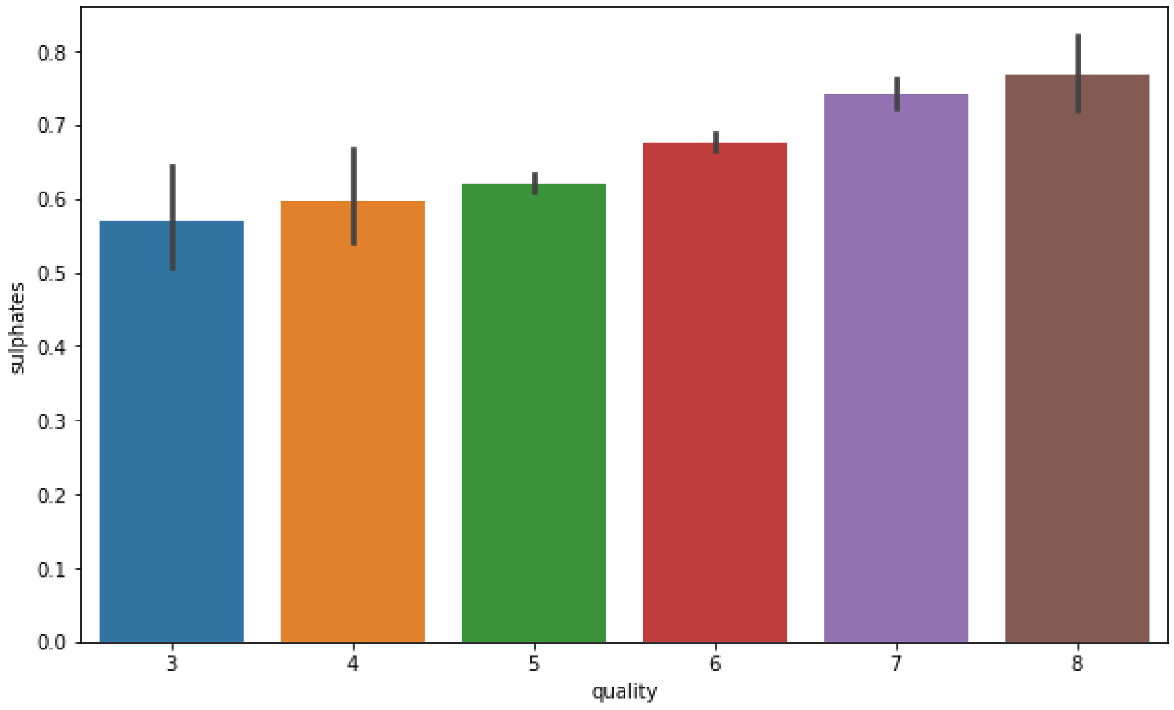
In [12]:

```
#Sulphates level goes higher with the quality of wine

fig = plt.figure(figsize = (10,6))
sns.barplot(x = 'quality', y = 'sulphates', data = data)
```

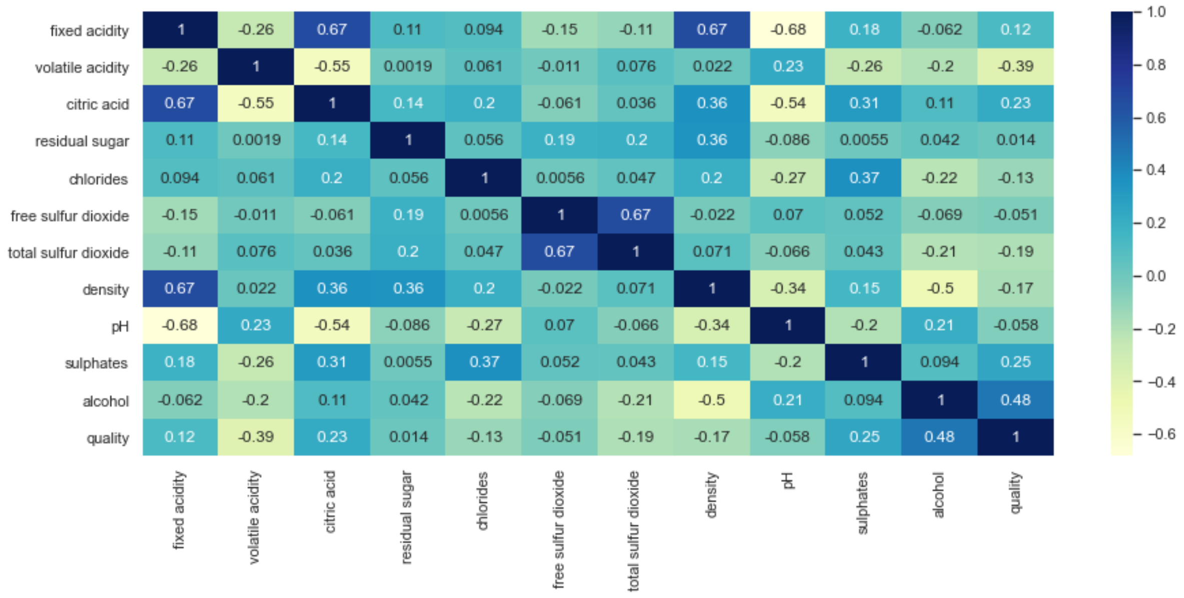
Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x227369f8df0>
```



In [13]:

```
sns.set(rc={'figure.figsize':(15,6)})
sns.heatmap(data.corr(), cmap = "YlGnBu", annot =True)
plt.show()
```



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. For Example, The strongly correlated items are : 1.fixed acidity and citric acid. 2.free sulphur dioxide and total sulphor dioxide. 3.fixed acidity and density.

The weekly correlated items are : 1.citric acid and volatile acidity. 2.fixed acidity and ph. 3.density and alcohol 4.ph and density

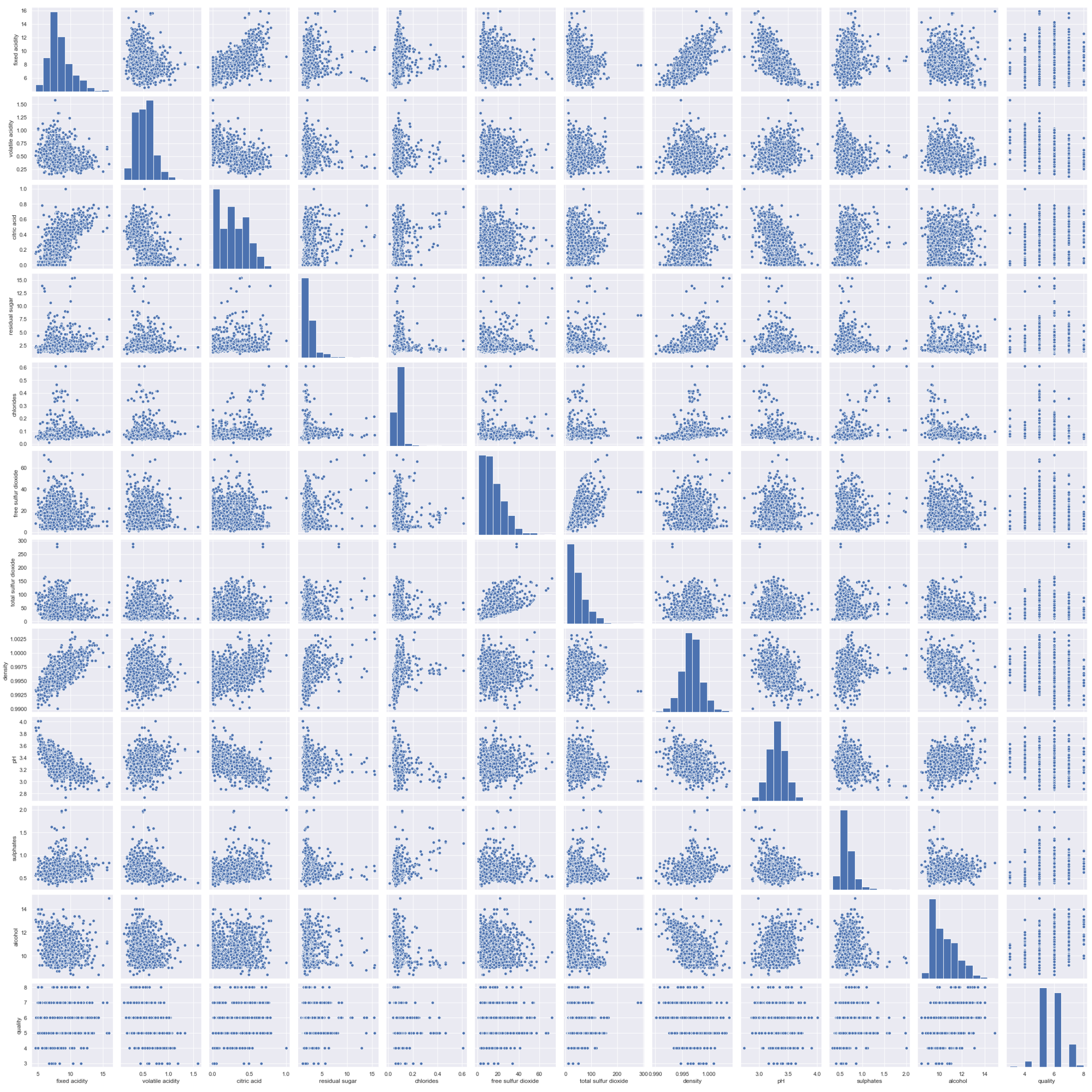
These are some relations which do not depend on each other at all.

In [14]:

```
sns.pairplot(data)
```

Out[14]:

```
<seaborn.axisgrid.PairGrid at 0x22737e1afa0>
```



In [15]:

```
data.columns
```

Out[15]:

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

Data preprocessing

In [16]:

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

In [17]:

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

Out[17]:

```
good    855
bad     744
Name: quality, dtype: int64
```

In [18]:

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
data['quality'].map({0:'bad',1:'good'})

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

data['quality'].value_counts
```

Out[18]:

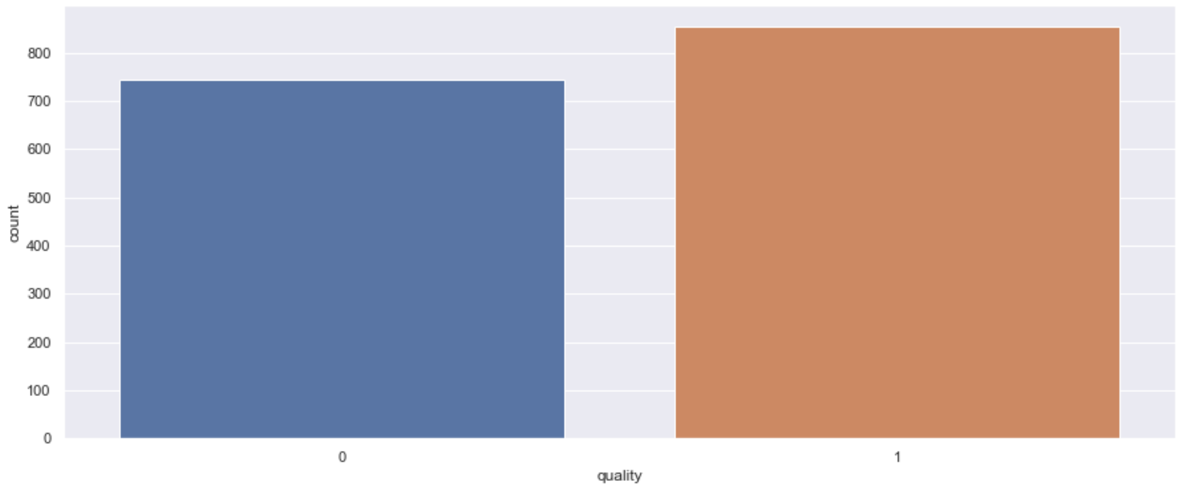
```
<bound method IndexOpsMixin.value_counts of 0       0
1       0
2       0
3       1
4       0
       ..
1594    0
1595    1
1596    1
1597    0
1598    1
Name: quality, Length: 1599, dtype: int32>
```

In [19]:

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

Out[19]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2273b5cb8e0>
```



In [20]:

```
# dividing the dataset into dependent and independent variables
X=data.iloc[:,:-1]
y=data.iloc[:,-1]
# determining the shape of x and y.
print(X.shape)
print(y.shape)
```

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

In [21]:

```
##Train Test Split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.4,train_size=0.6,random_stat
```

In [22]:

```
## Standardize the dataset
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
```

In [23]:

```
X_train=scaler.fit_transform(X_train)
X_train
```

Out[23]:

```
array([[ 0.99064185, -0.87252436,  0.34803642, ..., -0.9151259 ,
        -0.04638563,  0.15582671],
       [ 1.04727864,  0.79403337,  0.60383572, ..., -0.9151259 ,
        -1.15693875, -0.39958949],
       [-0.65182509,  0.93291319, -0.72632064, ..., -0.13271365,
        -0.72505698, -0.30702012],
       ...,
       [-0.65182509,  0.23851413, -1.13559952, ...,  0.45409553,
        -0.35487261, -0.12188139],
       [-0.76509867,  0.90513722, -1.39139882, ...,  0.25849247,
        -0.35487261, -0.67729759],
       [ 0.93400506, -1.15028398,  1.62703292, ..., -0.71952283,
         0.75568051, -0.76986695]])
```

In [24]:

```
X_test=scaler.transform(X_test)
```

Modelling

Logistic Regression

In [25]:

```
from sklearn.linear_model import LogisticRegression
# creating the model
model = LogisticRegression()

# 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 : 0.7372262773722628
Testing accuracy : 0.759375
              precision    recall  f1-score   support

           0       0.74      0.75      0.74       297
           1       0.78      0.77      0.77       343

    accuracy                           0.76       640
   macro avg       0.76      0.76      0.76       640
weighted avg       0.76      0.76      0.76       640

[[223  74]
 [ 80 263]]
```

Decision Tree

In [26]:

```
from sklearn.tree import DecisionTreeClassifier

# creating model
model = DecisionTreeClassifier(random_state=42)

# 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))
```

```
Training accuracy : 1.0
Testing accuracy : 0.7546875
```

In [27]:

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

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

```
              precision    recall  f1-score   support

           0       0.73      0.75      0.74       297
           1       0.78      0.76      0.77       343

    accuracy                           0.75       640
   macro avg       0.75      0.75      0.75       640
weighted avg       0.76      0.75      0.75       640

[[223  74]
 [ 83 260]]
```

In [28]:

```
#Now lets try to do some evaluation for decision tree model using cross validation.

model_eval = cross_val_score(estimator = model, X = X_train, y = y_train, cv = 10)
model_eval.mean()
```

Out[28]:

```
0.7382785087719299
```

Random forest

In [29]:

```
from sklearn.ensemble import RandomForestClassifier

# creating the model
model = RandomForestClassifier(n_estimators = 200,random_state=6)

# 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))
```

```
Training accuracy : 1.0
Testing accuracy : 0.815625
```

In [30]:

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

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

```
              precision    recall  f1-score   support

           0       0.79      0.82      0.81       297
           1       0.84      0.81      0.82       343

    accuracy                           0.82       640
   macro avg       0.81      0.82      0.82       640
weighted avg       0.82      0.82      0.82       640

[[244  53]
 [ 65 278]]
```

In [31]:

```
#Now lets try to do some evaluation for random forest model using cross validation.

model_eval = cross_val_score(estimator = model, X = X_train, y = y_train, cv = 8)
model_eval.mean()
```

Out[31]:

```
0.7883315826330533
```

Support Vector Machine

In [32]:

```
from sklearn.svm import SVC

# creating the model
model = SVC(random_state=4)

# 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))
```

```
Training accuracy : 0.8018769551616267
Testing accuracy : 0.7671875
```

In [33]:

```
# finding the best parameters for the SVC model

param = {
    'C': [0.8,0.9,1,1.1,1.2,1.3,1.4],
    'kernel':['linear', 'rbf'],
    'gamma' :[0.1,0.8,0.9,1,1.1,1.2,1.3,1.4]
}
grid_svc = GridSearchCV(model, param_grid = param, scoring = 'accuracy', cv = 8)
```

In [34]:

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

Out[34]:

```
GridSearchCV(cv=8, estimator=SVC(random_state=4),
             param_grid={'C': [0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4],
                         'gamma': [0.1, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4],
                         'kernel': ['linear', 'rbf']},
             scoring='accuracy')
```

In [35]:

```
grid_svc.best_params_
```

Out[35]:

```
{'C': 1.4, 'gamma': 0.1, 'kernel': 'rbf'}
```

In [36]:

```
# creating a new SVC model with these best parameters

model2 = SVC(C = 1.3, gamma = 0.8, kernel = 'rbf')
model2.fit(X_train, y_train)
y_pred = model2.predict(X_test)

print(classification_report(y_test, y_pred))
print("Training accuracy :", model2.score(X_train, y_train))
print("Testing accuracy :", model2.score(X_test, y_test))
```

```
              precision    recall  f1-score   support

           0       0.79      0.77      0.78       297
           1       0.80      0.82      0.81       343

    accuracy                           0.80       640
   macro avg       0.80      0.79      0.80       640
weighted avg       0.80      0.80      0.80       640

Training accuracy : 0.9676746611053181
Testing accuracy : 0.796875
```

In [ ]:

```
```

In [ ]:

```
```