In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

```
data=pd.read_csv('WINE DATA\\winequality-white.csv',delimiter = ";")
```

In [3]:

```
data
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
...	...	...	...	...	...	...	...	...	...	...	...	...
4893	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2	6
4894	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6	5
4895	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4	6
4896	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8	7
4897	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8	6

4898 rows × 12 columns

In [4]:

```
data.columns
```

Out[4]:

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

In [5]:

```
data.describe()
```

Out[5]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulph
count	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.00
mean	6.854788	0.278241	0.334192	6.391415	0.045772	35.308085	138.360657	0.994027	3.188267	0.48
std	0.843868	0.100795	0.121020	5.072058	0.021848	17.007137	42.498065	0.002991	0.151001	0.11
min	3.800000	0.080000	0.000000	0.600000	0.009000	2.000000	9.000000	0.987110	2.720000	0.22
25%	6.300000	0.210000	0.270000	1.700000	0.036000	23.000000	108.000000	0.991723	3.090000	0.41
50%	6.800000	0.260000	0.320000	5.200000	0.043000	34.000000	134.000000	0.993740	3.180000	0.47

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulph
75%	7.300000	0.320000	0.390000	9.900000	0.050000	46.000000	167.000000	0.996100	3.280000	0.55
max	14.200000	1.100000	1.660000	65.800000	0.346000	289.000000	440.000000	1.038980	3.820000	1.08

In [6]:

```
print("Total quantity of wine quality 3 : ",len([a for a in data['quality'] if a==3]))
print("Total quantity of wine quality 4 : ",len([a for a in data['quality'] if a==4]))
print("Total quantity of wine quality 5 : ",len([a for a in data['quality'] if a==5]))
print("Total quantity of wine quality 6 : ",len([a for a in data['quality'] if a==6]))
print("Total quantity of wine quality 7 : ",len([a for a in data['quality'] if a==7]))
print("Total quantity of wine quality 8 : ",len([a for a in data['quality'] if a==8]))
print("Total quantity of wine quality 9 : ",len([a for a in data['quality'] if a==9]))
```

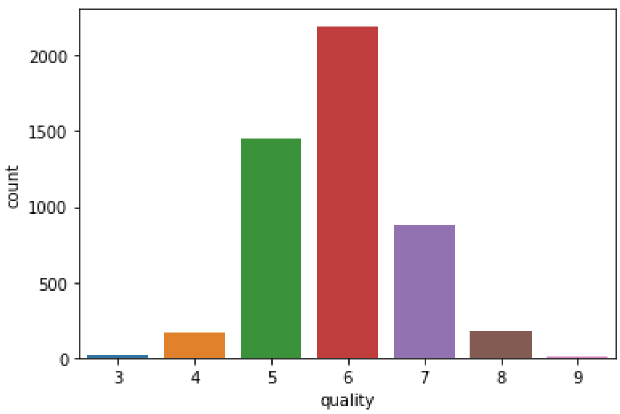
```
Total quantity of wine quality 3 :  20
Total quantity of wine quality 4 :  163
Total quantity of wine quality 5 :  1457
Total quantity of wine quality 6 :  2198
Total quantity of wine quality 7 :  880
Total quantity of wine quality 8 :  175
Total quantity of wine quality 9 :  5
```

In [7]:

```
sns.countplot(x='quality', data=data)
```

Out[7]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a5d7067da0>
```



In [11]:

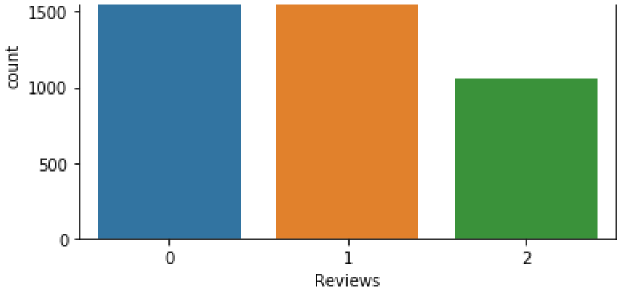
```
reviews = []
for i in data['quality']:
    if i <= 5:
        reviews.append(0)
    elif (i==6):
        reviews.append(1)
    elif (i>6):
        reviews.append(2)
data['Reviews'] = reviews
```

In [12]:

```
sns.countplot(x='Reviews', data=data)
```

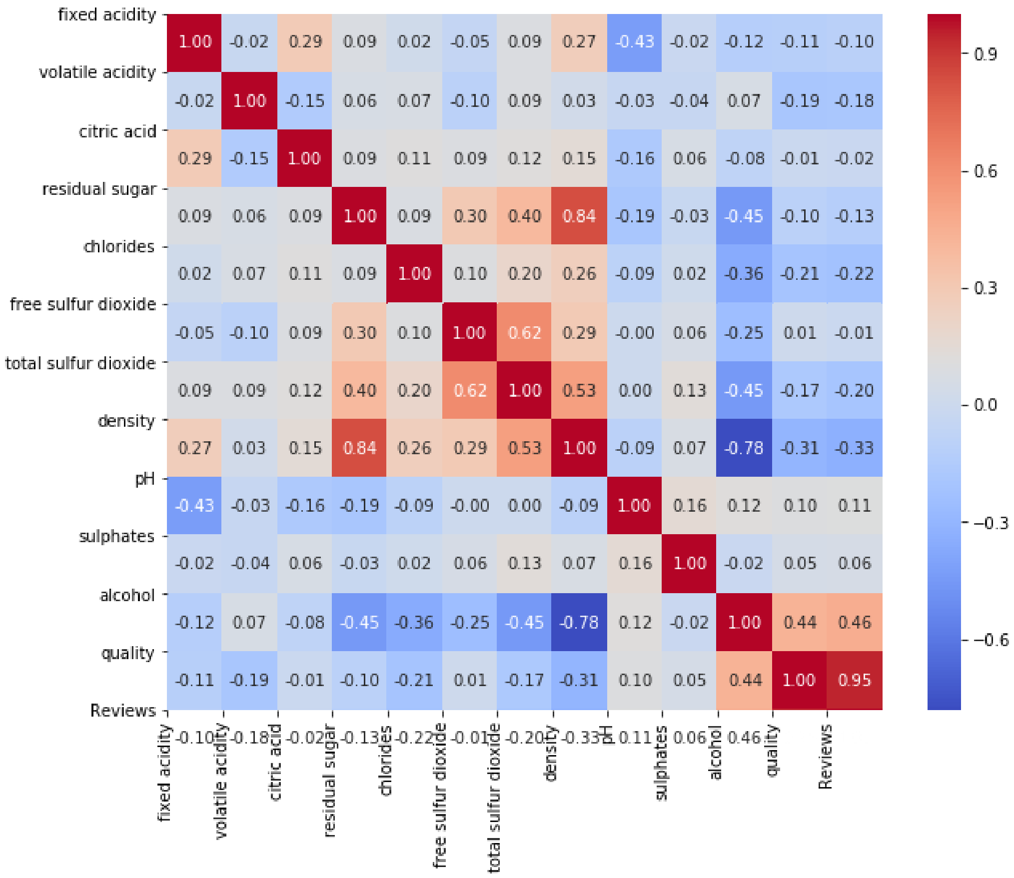
Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a5d73fd2e8>
```



In [13]:

```python
corr = data.corr()
#Plot figsize
fig, ax = plt.subplots(figsize=(10, 8))
#Generate Heat Map, allow annotations and place floats in map
sns.heatmap(corr, cmap='coolwarm', annot=True, fmt=".2f")
#Apply xticks
plt.xticks(range(len(corr.columns)), corr.columns);
#Apply yticks
plt.yticks(range(len(corr.columns)), corr.columns)
#show plot
plt.show()
```



In [14]:

```python
data.head()
```

Out[14]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	Reviews
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6	1
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6	1
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6	1
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6	1

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	Reviews
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6	1

In [15]:

```
data.tail()
```

Out[15]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	Reviews
4893	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2	6	1
4894	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6	5	0
4895	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4	6	1
4896	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8	7	2
4897	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8	6	1

In [16]:

```
X = data.iloc[:,0:11]
y = data.iloc[:,12]
y_mul = data.iloc[:,11]
```

In [20]:

```
from sklearn.preprocessing import MinMaxScaler
minmax = MinMaxScaler()
X_s = minmax.fit_transform(X)
```

In [21]:

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X_s,y,test_size = 0.3,random_state = 42)
```

In [22]:

```
print(X_train.shape)
print(X_test.shape)
```

```
(3428, 11)
(1470, 11)
```

In [23]:

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
```

In [38]:

```
svc_lin  = SVC(kernel = 'linear',class_weight = 'balanced')
svc_lin.fit(X_train, y_train)
y_pred_svc_lin=svc_lin.predict(X_test)
lin_svc_conf_matrix = confusion_matrix(y_test, y_pred_svc_lin)
lin_svc_clf = classification_report(y_test, y_pred_svc_lin)
print(lin_svc_conf_matrix)
print(lin_svc_clf)
```

```
[[334  86  53]
 [236 202 230]
 [ 47  56 226]]
              precision    recall  f1-score   support

           0       0.54      0.71      0.61       473
           1       0.59      0.30      0.40       668
```

```
           2       0.44      0.69      0.54       329

    accuracy                           0.52      1470
   macro avg       0.52      0.57      0.52      1470
weighted avg       0.54      0.52      0.50      1470
```

In [39]:

```python
svc_rbf  = SVC(kernel = 'rbf',class_weight = 'balanced')
svc_rbf.fit(X_train, y_train)
y_pred_svc_rbf=svc_rbf.predict(X_test)
rbf_svc_conf_matrix = confusion_matrix(y_test, y_pred_svc_rbf)
rbf_svc_clf = classification_report(y_test, y_pred_svc_rbf)
print(rbf_svc_conf_matrix)
print(rbf_svc_clf)
```

```
[[334 104  35]
 [191 262 215]
 [ 34  68 227]]
              precision    recall  f1-score   support

           0       0.60      0.71      0.65       473
           1       0.60      0.39      0.48       668
           2       0.48      0.69      0.56       329

    accuracy                           0.56      1470
   macro avg       0.56      0.60      0.56      1470
weighted avg       0.57      0.56      0.55      1470
```

In [42]:

```python
svc_poly  = SVC(kernel = 'poly',class_weight = 'balanced')
svc_poly.fit(X_train, y_train)
y_pred_svc_poly =  svc_poly.predict(X_test)
poly_svc_conf_matrix = confusion_matrix(y_test, y_pred_svc_poly)
poly_svc_clf = classification_report(y_test, y_pred_svc_poly)
print(poly_svc_conf_matrix)
print(poly_svc_clf)
```

```
[[350  93  30]
 [212 265 191]
 [ 24  85 220]]
              precision    recall  f1-score   support

           0       0.60      0.74      0.66       473
           1       0.60      0.40      0.48       668
           2       0.50      0.67      0.57       329

    accuracy                           0.57      1470
   macro avg       0.56      0.60      0.57      1470
weighted avg       0.58      0.57      0.56      1470
```

In [46]:

```python
svc_sigmoid  = SVC(kernel = 'sigmoid')
svc_sigmoid.fit(X_train, y_train)
y_pred_svc_sigmoid=svc_sigmoid.predict(X_test)
sigmoid_svc_conf_matrix = confusion_matrix(y_test, y_pred_svc_sigmoid)
sigmoid_svc_clf = classification_report(y_test, y_pred_svc_sigmoid)
print(sigmoid_svc_conf_matrix)
print(sigmoid_svc_clf)
```

```
[[206 110 157]
 [287 178 203]
 [189  74  66]]
              precision    recall  f1-score   support

           0       0.30      0.44      0.36       473
           1       0.49      0.27      0.35       668
```

```
           2       0.15      0.20      0.17       329

    accuracy                           0.31      1470
   macro avg       0.32      0.30      0.29      1470
weighted avg       0.36      0.31      0.31      1470
```

In [49]:

```
from sklearn.model_selection import cross_val_score
clf = SVC(kernel='linear', C=1)
scores = cross_val_score(clf, X_s, y, cv=5)
print(scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.49183673 0.51938776 0.56326531 0.53626149 0.56588355]
Accuracy: 0.54 (+/- 0.06)
```

In [37]:

```
from sklearn.model_selection import cross_val_score
clf = SVC(kernel='rbf', C=1)
scores = cross_val_score(clf, X_s, y, cv=5)
print(scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.54489796 0.56734694 0.61632653 0.58937692 0.57814096]
Accuracy: 0.58 (+/- 0.05)
```

In [50]:

```
from sklearn.model_selection import cross_val_score
clf = SVC(kernel='poly',C=1)
scores = cross_val_score(clf, X_s, y, cv=5)
print(scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.53877551 0.55       0.61938776 0.55771195 0.56894791]
Accuracy: 0.57 (+/- 0.06)
```

In [48]:

```
from sklearn.model_selection import cross_val_score
clf = SVC(kernel='sigmoid',C=1)
scores = cross_val_score(clf, X_s, y, cv=5)
print(scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.3755102  0.41632653 0.39387755 0.35955056 0.29213483]
Accuracy: 0.37 (+/- 0.08)
```

In [ ]: