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]:

```
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]))
```

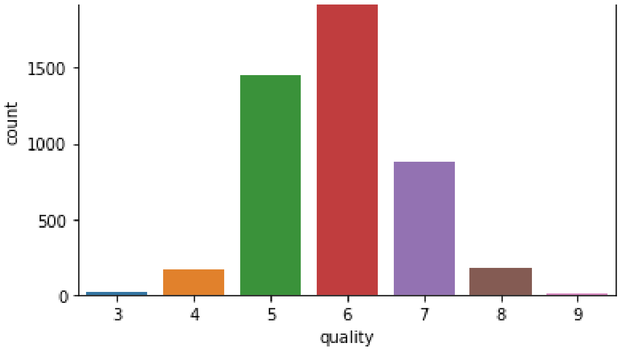
```
Total quantity of wine quality 3 :  20
Total quantity of wine quality 4 :  163
Total quantity of wine quality 5 :  1457
Total quantity of wine quality 6 :  2198
Total quantity of wine quality 7 :  880
Total quantity of wine quality 8 :  175
Total quantity of wine quality 9 :  5
```

In [5]:

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

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d5c2d807f0>
```



In [6]:

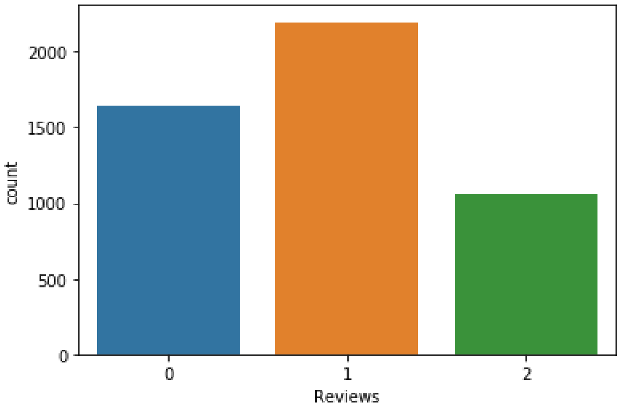
```
reviews = []
for i in data['quality']:
    if i <= 5:
        reviews.append(0)
    elif (i==6):
        reviews.append(1)
    elif (i>6):
        reviews.append(2)
data['Reviews'] = reviews
```

In [30]:

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

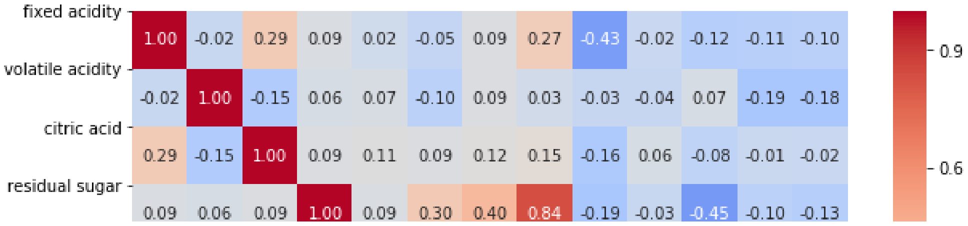
Out[30]:

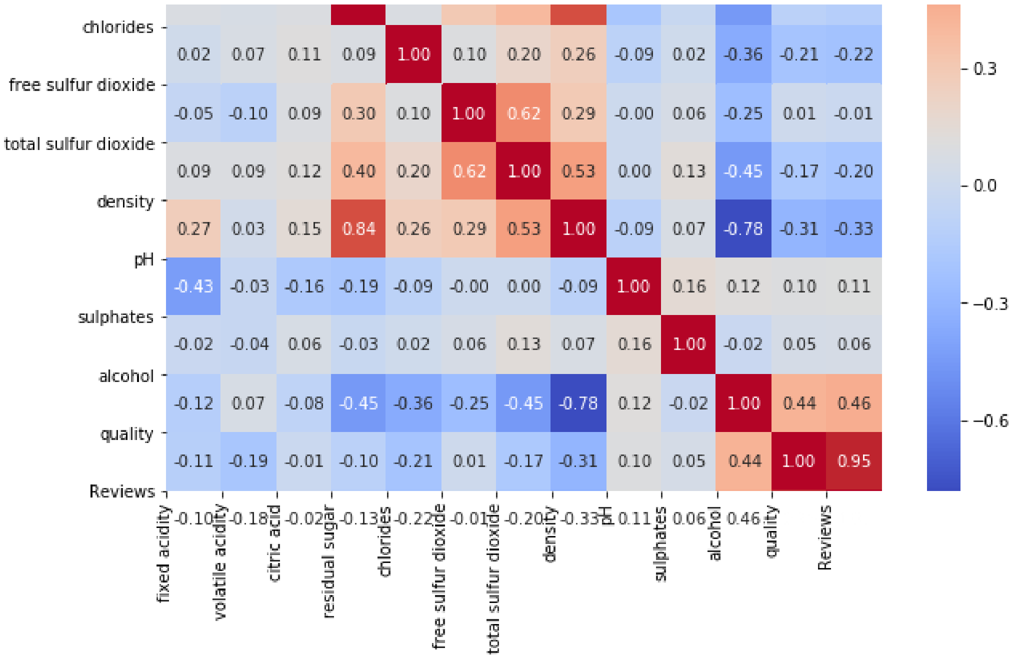
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d5c387fba8>
```



In [7]:

```
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 [8]:

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

In [9]:

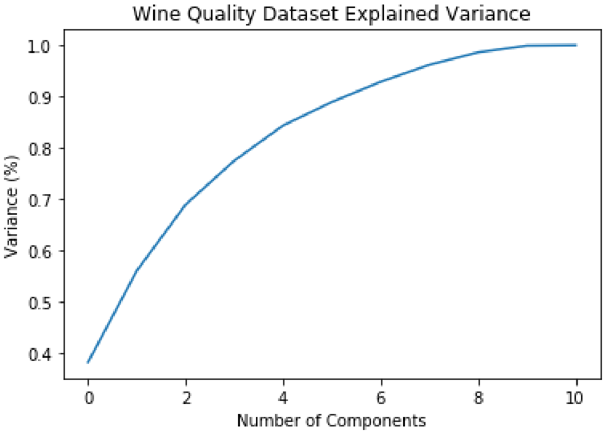
```
from sklearn.preprocessing import MinMaxScaler
minmax = MinMaxScaler()
X_s = minmax.fit_transform(X)
```

In [10]:

```
from sklearn.decomposition import PCA
pca = PCA()
X_s_pca = pca.fit(X_s)
```

In [11]:

```
plt.figure()
plt.plot(np.cumsum(X_s_pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Variance (%)') #for each component
plt.title('Wine Quality Dataset Explained Variance')
plt.show()
```



In [12]:

```
pca = PCA(n_components=9)
X_s_pca = pca.fit_transform(X_s)
```

In [13]:

```
from sklearn.model_selection import train_test_split

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

In [14]:

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

```
(3428, 9)
(1470, 9)
```

In [15]:

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

In [17]:

```
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)
```

```
[[335  83  55]
 [238 197 233]
 [ 45  69 215]]
              precision    recall  f1-score   support

           0       0.54      0.71      0.61       473
           1       0.56      0.29      0.39       668
           2       0.43      0.65      0.52       329

    accuracy                           0.51      1470
   macro avg       0.51      0.55      0.51      1470
weighted avg       0.53      0.51      0.49      1470
```

In [20]:

```
svc_rbf  = SVC(kernel = 'rbf')
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)
```

```
[[298 169   6]
 [143 476  49]
 [ 15 205 109]]
              precision    recall  f1-score   support

           0       0.65      0.63      0.64       473
           1       0.56      0.71      0.63       668
           2       0.66      0.33      0.44       329

    accuracy                           0.60      1470
   macro avg       0.63      0.56      0.57      1470
weighted avg       0.61      0.60      0.59      1470
```

In [23]:

```
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)
```

```
[[279 163  31]
 [155 389 124]
 [ 24 144 161]]
              precision    recall  f1-score   support

           0       0.61      0.59      0.60       473
           1       0.56      0.58      0.57       668
           2       0.51      0.49      0.50       329

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

In [24]:

```
svc_sigmoid  = SVC(kernel = 'sigmoid',class_weight= 'balanced')
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)
```

```
[[300  87  86]
 [295 135 238]
 [ 98  64 167]]
              precision    recall  f1-score   support

           0       0.43      0.63      0.51       473
           1       0.47      0.20      0.28       668
           2       0.34      0.51      0.41       329

    accuracy                           0.41      1470
   macro avg       0.42      0.45      0.40      1470
weighted avg       0.43      0.41      0.39      1470
```

In [26]:

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

```
[0.4877551  0.49693878 0.54897959 0.49438202 0.48314607]
Accuracy: 0.50 (+/- 0.05)
```

In [27]:

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

```
[0.54081633 0.55408163 0.60102041 0.58222676 0.58120531]
Accuracy: 0.57 (+/- 0.04)
```

In [28]:

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

```
[0.54183673 0.53163265 0.55612245 0.53115424 0.51481103]
Accuracy: 0.54 (+/- 0.03)
```

In [29]:

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

```
[0.41734694 0.47040816 0.45510204 0.47395301 0.39223698]
Accuracy: 0.44 (+/- 0.06)
```

In [ ]: