In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
plt.style.use('fivethirtyeight')
```

In [4]:
```
df = pd.read_csv('C:/Users/nisho/Documents/SEM 5/ML and core applications/winequality-red.csv')
df.head(10)
```

Out[4]:

| | 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 |
| **5** | 7.4 | 0.66 | 0.00 | 1.8 | 0.075 | 13.0 | 40.0 | 0.9978 | 3.51 | 0.56 | 9.4 | 5 |
| **6** | 7.9 | 0.60 | 0.06 | 1.6 | 0.069 | 15.0 | 59.0 | 0.9964 | 3.30 | 0.46 | 9.4 | 5 |
| **7** | 7.3 | 0.65 | 0.00 | 1.2 | 0.065 | 15.0 | 21.0 | 0.9946 | 3.39 | 0.47 | 10.0 | 7 |
| **8** | 7.8 | 0.58 | 0.02 | 2.0 | 0.073 | 9.0 | 18.0 | 0.9968 | 3.36 | 0.57 | 9.5 | 7 |
| **9** | 7.5 | 0.50 | 0.36 | 6.1 | 0.071 | 17.0 | 102.0 | 0.9978 | 3.35 | 0.80 | 10.5 | 5 |

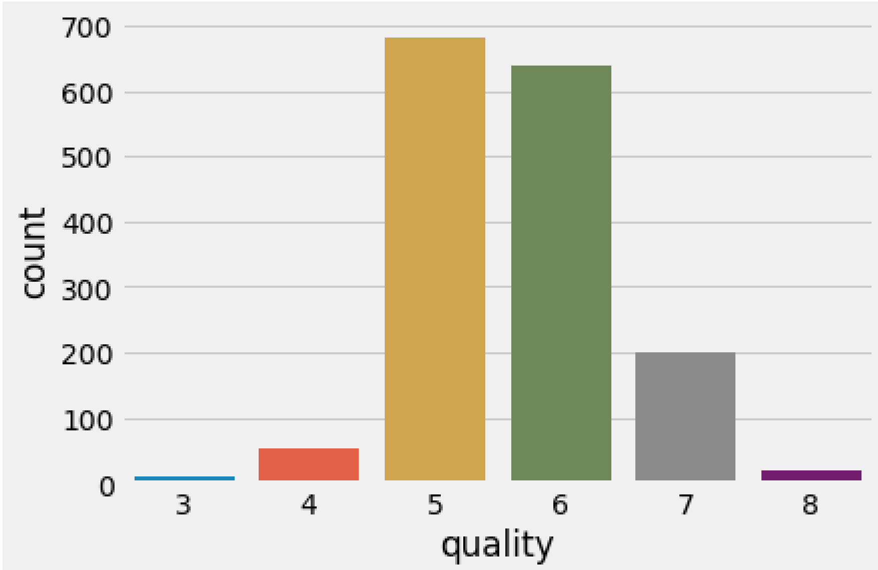
# EDA

In [5]:
```
df.describe().T
```

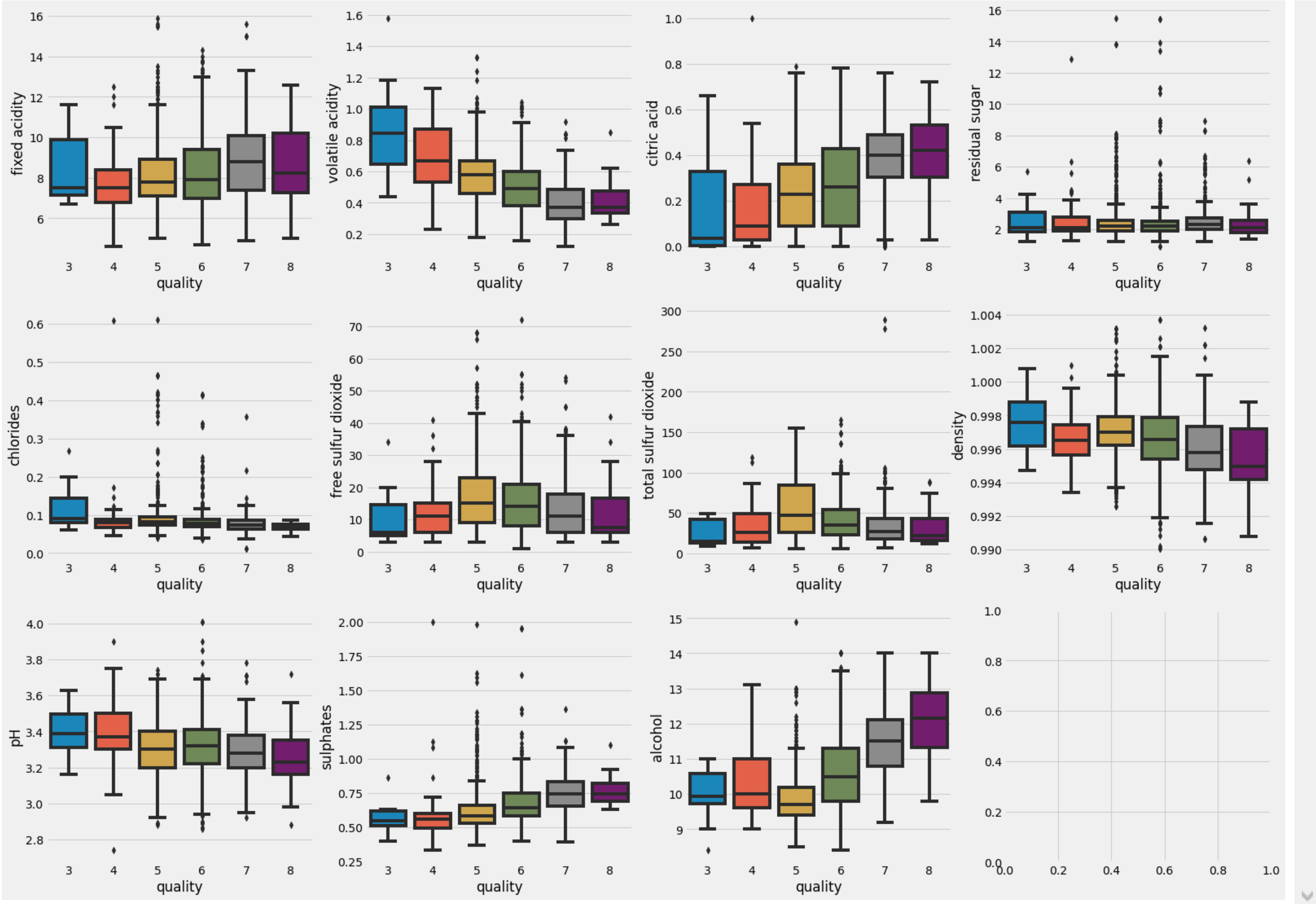
Out[5]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **fixed acidity** | 1599.0 | 8.319637 | 1.741096 | 4.60000 | 7.1000 | 7.90000 | 9.200000 | 15.90000 |
| **volatile acidity** | 1599.0 | 0.527821 | 0.179060 | 0.12000 | 0.3900 | 0.52000 | 0.640000 | 1.58000 |
| **citric acid** | 1599.0 | 0.270976 | 0.194801 | 0.00000 | 0.0900 | 0.26000 | 0.420000 | 1.00000 |
| **residual sugar** | 1599.0 | 2.538806 | 1.409928 | 0.90000 | 1.9000 | 2.20000 | 2.600000 | 15.50000 |
| **chlorides** | 1599.0 | 0.087467 | 0.047065 | 0.01200 | 0.0700 | 0.07900 | 0.090000 | 0.61100 |
| **free sulfur dioxide** | 1599.0 | 15.874922 | 10.460157 | 1.00000 | 7.0000 | 14.00000 | 21.000000 | 72.00000 |
| **total sulfur dioxide** | 1599.0 | 46.467792 | 32.895324 | 6.00000 | 22.0000 | 38.00000 | 62.000000 | 289.00000 |
| **density** | 1599.0 | 0.996747 | 0.001887 | 0.99007 | 0.9956 | 0.99675 | 0.997835 | 1.00369 |
| **pH** | 1599.0 | 3.311113 | 0.154386 | 2.74000 | 3.2100 | 3.31000 | 3.400000 | 4.01000 |
| **sulphates** | 1599.0 | 0.658149 | 0.169507 | 0.33000 | 0.5500 | 0.62000 | 0.730000 | 2.00000 |
| **alcohol** | 1599.0 | 10.422983 | 1.065668 | 8.40000 | 9.5000 | 10.20000 | 11.100000 | 14.90000 |
| **quality** | 1599.0 | 5.636023 | 0.807569 | 3.00000 | 5.0000 | 6.00000 | 6.000000 | 8.00000 |

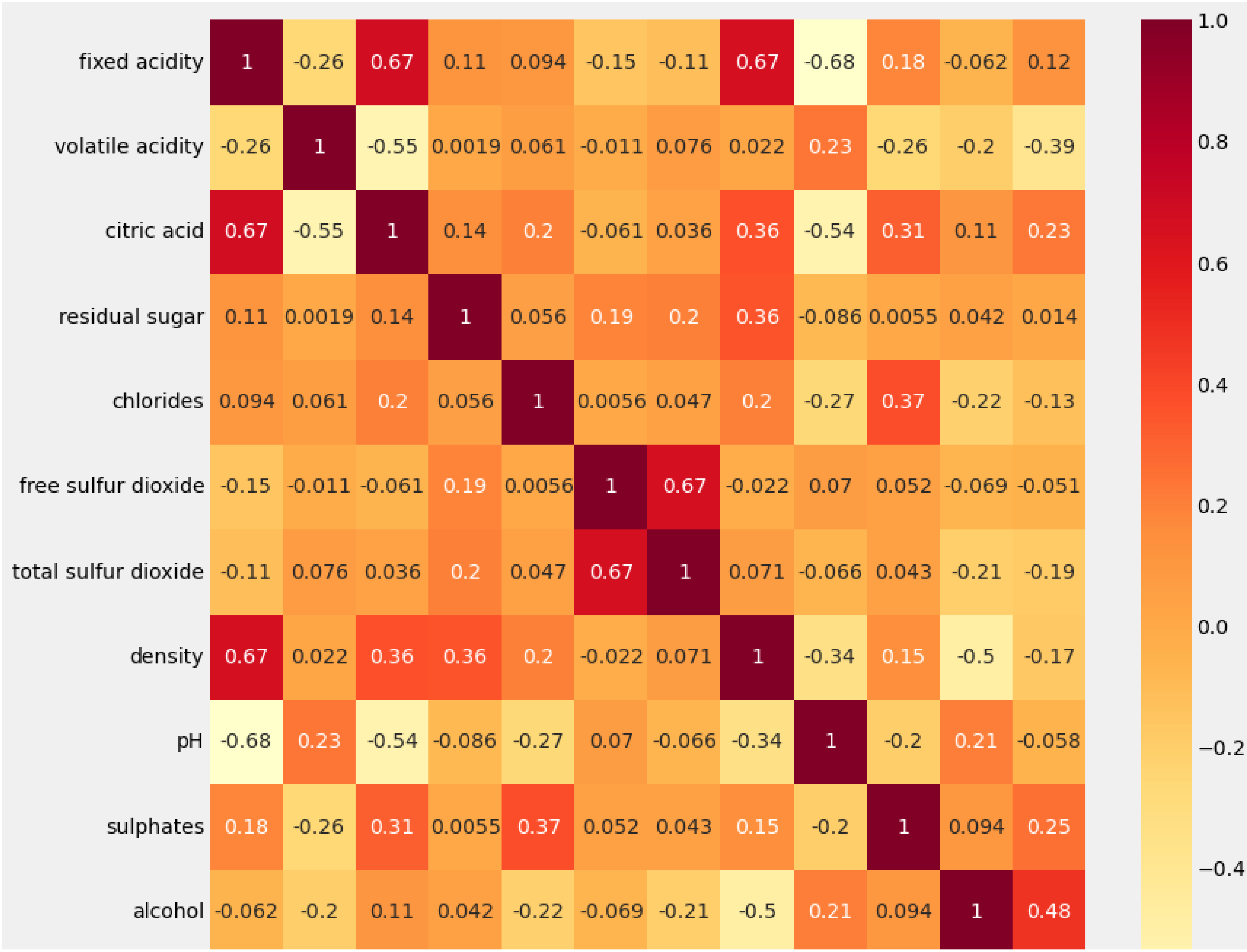
In [6]:
```
sns.countplot(x='quality', data=df)
plt.show()
```

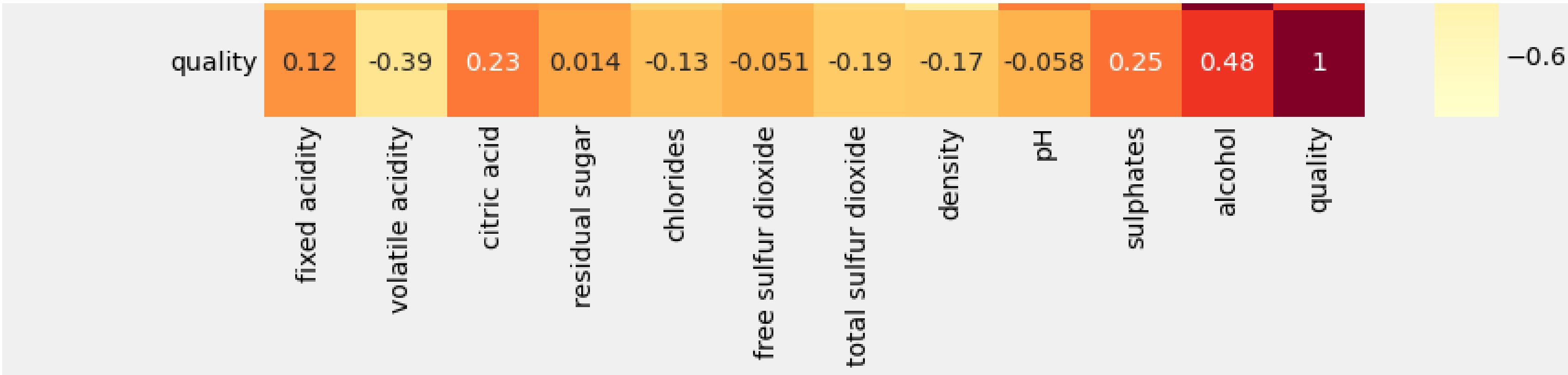


In [7]:
```
cols = list(df.columns)
fig, ax = plt.subplots(3,4, figsize=(24,18))

for i in range(11):
    j = i // 4
    k = i % 4
    sns.boxplot(y=cols[i], x = 'quality', data=df, ax = ax[j][k])

plt.show()
```

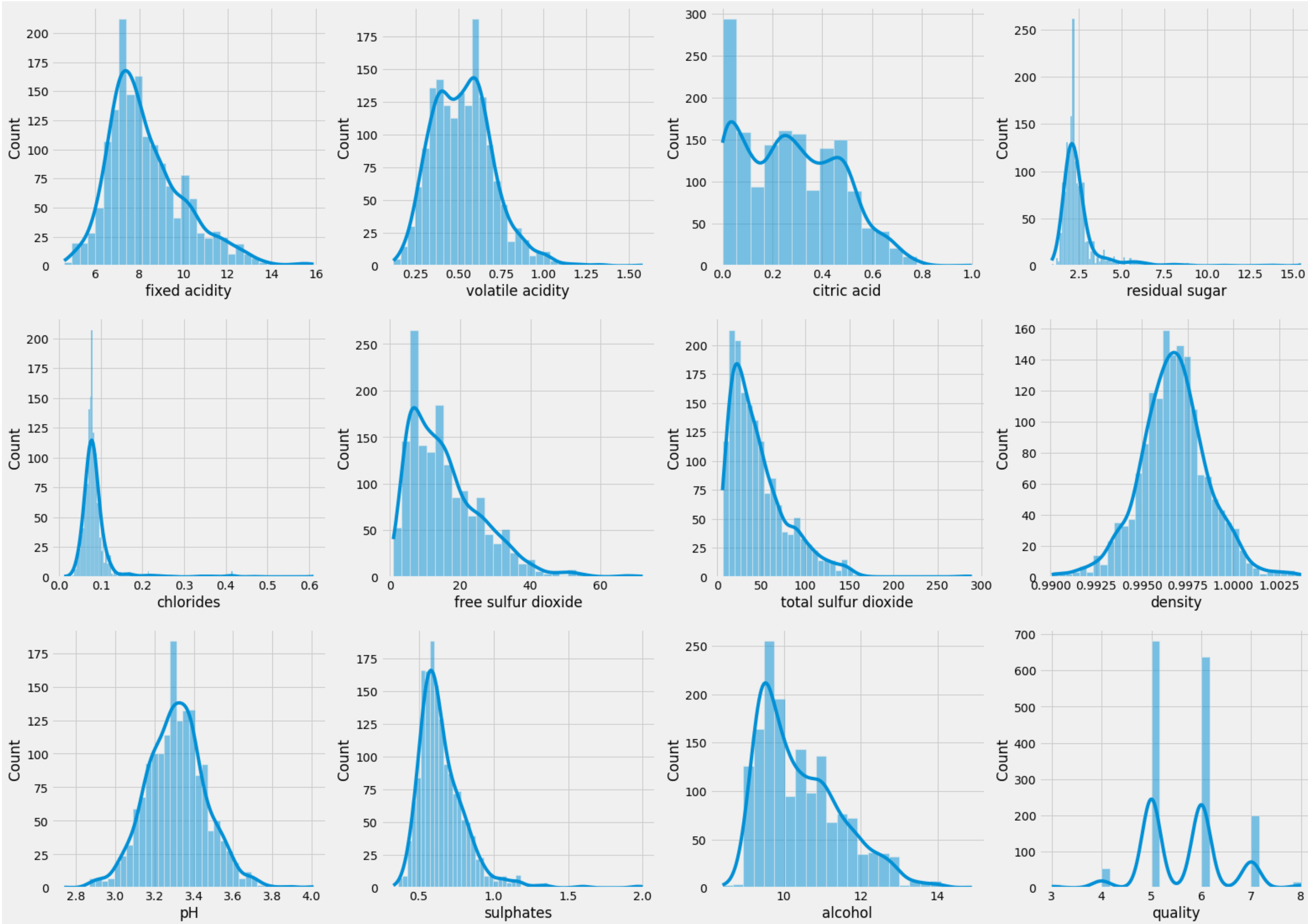

fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol
quality

In [8]:
```
corr = df.corr()
plt.figure(figsize=(12,12))
sns.heatmap(corr, cmap='YlOrRd', annot=True)
plt.show()
```



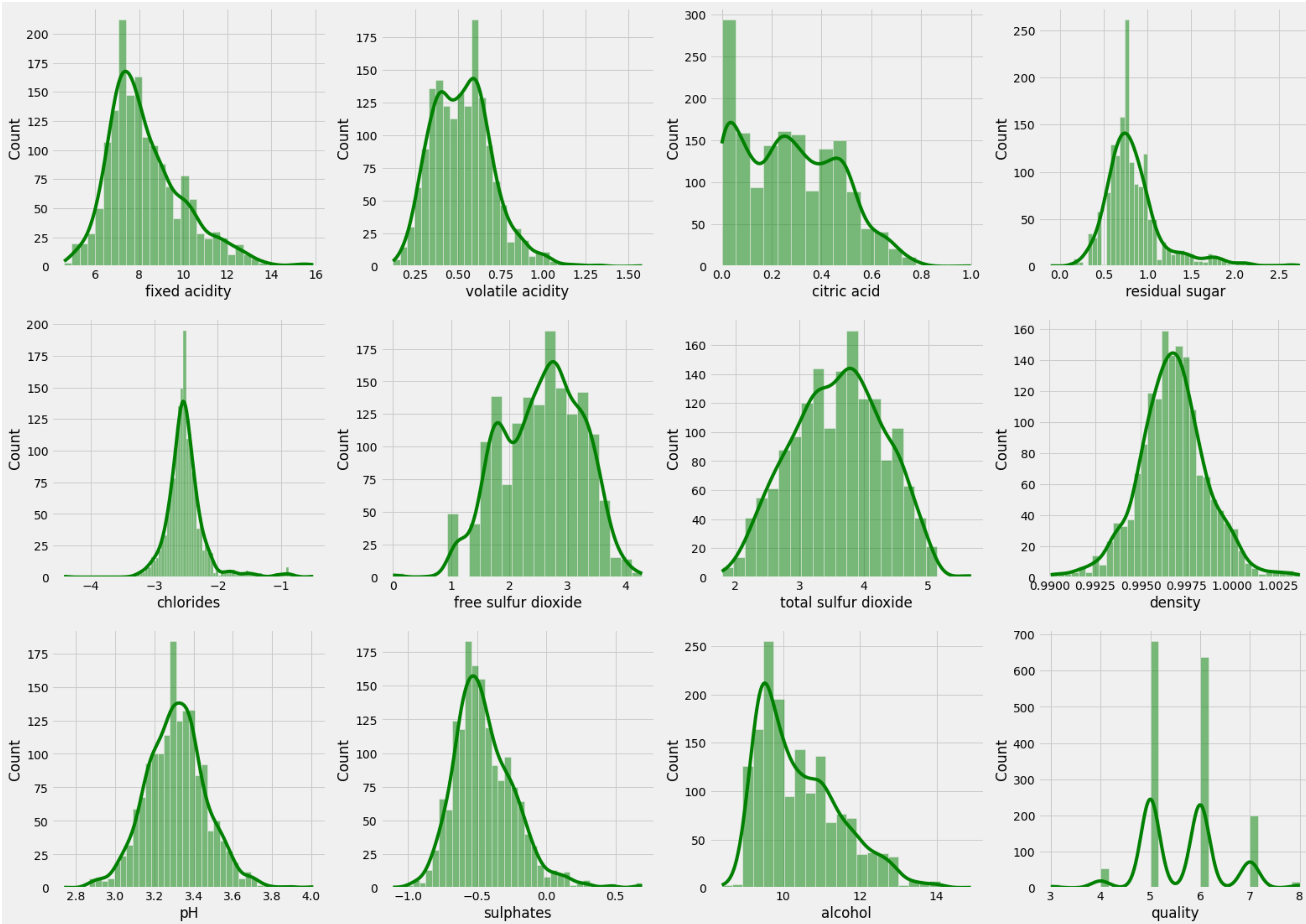

quality
0.12
-0.39
0.23
0.014
-0.13
-0.051
-0.19
-0.17
-0.058
0.25
0.48
1
−0.6
fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol
quality

In [9]:
```
fig, ax = plt.subplots(3,4, figsize=(24,18))

for i in range(12):
    j = i//4
    k = i%4
    sns.histplot(x=cols[i], data=df, ax=ax[j][k], kde=True)
```



In [16]:
```
skew_cols = ['residual sugar', 'chlorides','free sulfur dioxide','total sulfur dioxide','sulphates']

for col in skew_cols:
    df[col] = df[col].apply(lambda x: np.log(x))
```

In [19]:
```
fig, ax = plt.subplots(3,4, figsize=(24,18))

for i in range(12):
    j = i//4
    k = i%4
    sns.histplot(x=cols[i], data=df, ax=ax[j][k], kde=True, color='green')
```



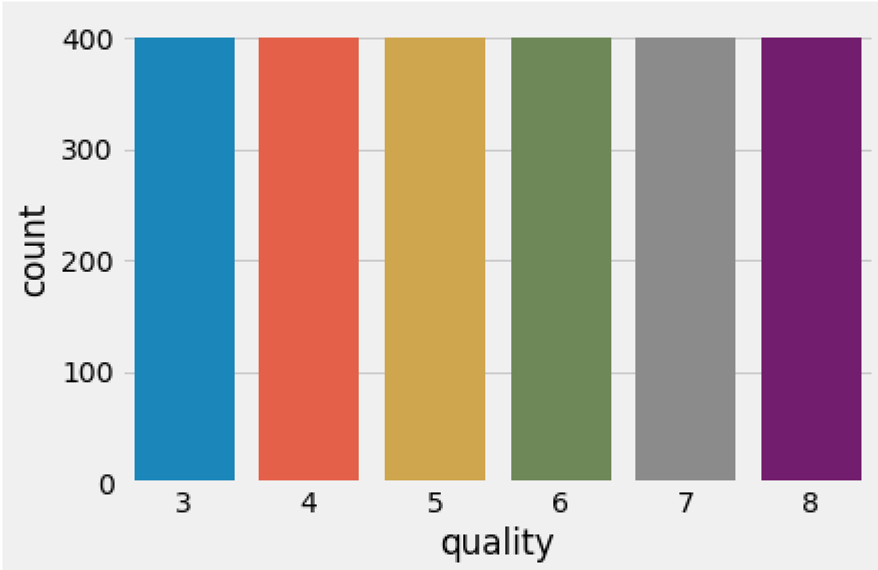
In [20]:
```
df_3 = df[df.quality==3]
df_4 = df[df.quality==4]
df_5 = df[df.quality==5]
df_6 = df[df.quality==6]
df_7 = df[df.quality==7]
df_8 = df[df.quality==8]
```

In [21]:
```
from sklearn.utils import resample

df_3_upsampled = resample(df_3, replace=True, n_samples=400, random_state=42)
df_4_upsampled = resample(df_4, replace=True, n_samples=400, random_state=42)
df_7_upsampled = resample(df_7, replace=True, n_samples=400, random_state=42)
df_8_upsampled = resample(df_8, replace=True, n_samples=400, random_state=42)

df_5_downsampled = df_5.sample(n=400).reset_index(drop=True)
df_6_downsampled = df_6.sample(n=400).reset_index(drop=True)
```

In [22]:
```
df_resampled = pd.concat([df_3_upsampled, df_4_upsampled, df_7_upsampled, df_8_upsampled,
                          df_5_downsampled, df_6_downsampled]).reset_index(drop=True)
df_resampled.quality.value_counts().sort_index()
```

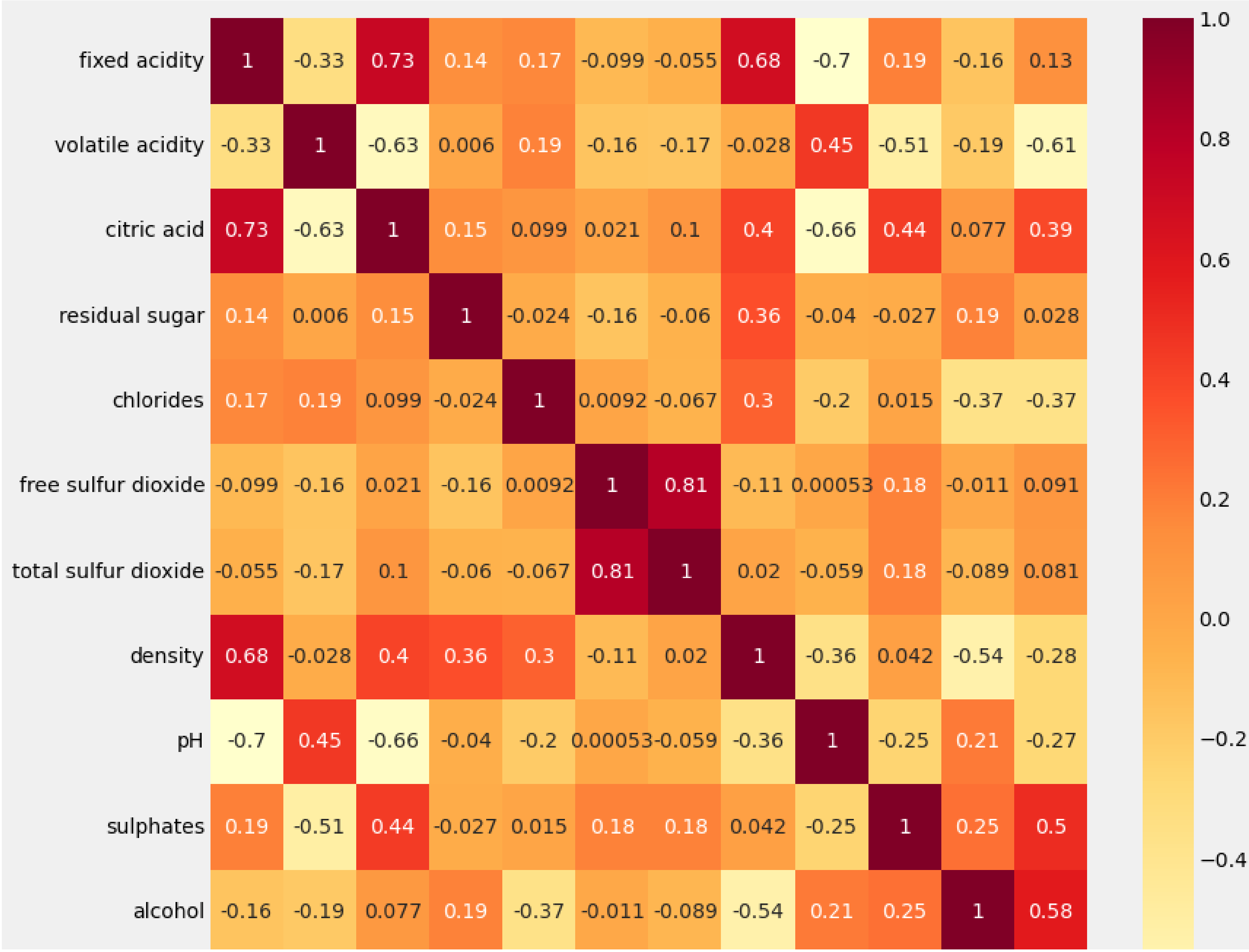
Out[22]:
```
3    400
4    400
5    400
6    400
7    400
8    400
Name: quality, dtype: int64
```

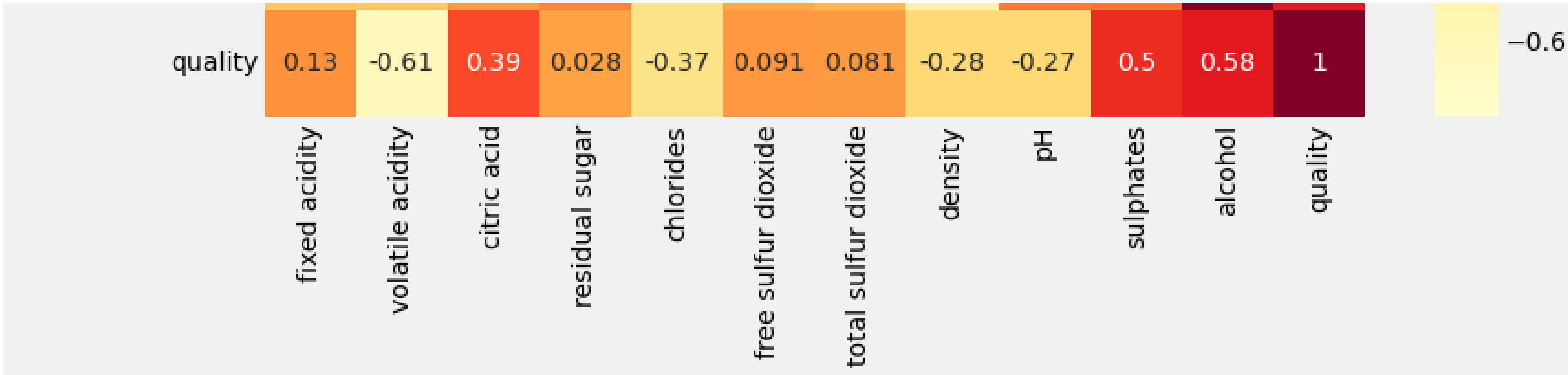
In [23]:
```
sns.countplot(x='quality', data=df_resampled)
plt.show()
```




Now we have equal sample sizes for all classes

In [24]:
```
corr_2 = df_resampled.corr()
plt.figure(figsize=(12,12))
sns.heatmap(corr_2, cmap='YlOrRd', annot=True)
plt.show()
```






In [25]:
```
corr_2.loc[(corr_2.quality >= 0.05) | (corr_2.quality <= -0.05), 'quality']
```

Out[25]:
```
fixed acidity           0.126471
volatile acidity       -0.611091
citric acid             0.387725
chlorides              -0.365112
free sulfur dioxide     0.091006
total sulfur dioxide    0.080836
density                -0.275677
pH                     -0.271249
sulphates               0.497583
alcohol                 0.580071
quality                 1.000000
Name: quality, dtype: float64
```

In [26]:
```
X = df_resampled.drop(['residual sugar', 'quality'], axis=1)
y = df_resampled['quality']
```

# Modelling

# decision trees

In [27]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
```

In [28]:
```
model = DecisionTreeClassifier(random_state = 42)
score = cross_val_score(model, X, y, cv=5)
print('Initial Score for DT classifier: ', score, '\nMean score: ', score.mean())
```

```
Initial Score for DT classifier:  [0.84791667 0.81666667 0.85208333 0.81666667 0.82083333]
Mean score:  0.8308333333333333
```

In [29]:
```
from sklearn.model_selection import GridSearchCV
```

In [30]:
```
model = DecisionTreeClassifier(random_state=42)
parameters = {'max_depth':[5,10,15,20], 'max_features' : ['auto','sqrt','log2']}
cv = GridSearchCV(model, parameters, cv=5)
cv.fit(X, y)
```

Out[30]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=42),
             param_grid={'max_depth': [5, 10, 15, 20],
                         'max_features': ['auto', 'sqrt', 'log2']})
```

In [31]:
```
cv.best_score_
```

Out[31]:
```
0.8324999999999999
```

In [32]:
```
cv.best_estimator_
```

Out[32]:
```
DecisionTreeClassifier(max_depth=15, max_features='auto', random_state=42)
```

# Random forest

In [33]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [34]:
```
model = RandomForestClassifier(random_state=42)

score = cross_val_score(model, X, y, cv=5, scoring='accuracy')
print('Initial score for RF: ', score, '\nMean Score: ', score.mean())
```

```
Initial score for RF:  [0.88333333 0.8625     0.86875    0.85416667 0.84791667]
Mean Score:  0.8633333333333333
```

In [35]:
```
model = RandomForestClassifier(random_state=42, max_depth=15)

score = cross_val_score(model, X, y, cv=5, scoring='accuracy')
print('Improved score for RF: ', score, '\nMean Score: ', score.mean())
```

```
Improved score for RF:  [0.89375    0.85625    0.85833333 0.86458333 0.85625   ]
Mean Score:  0.8658333333333333
```

# Conclusion

Since random forest has a comparatively better mean score it outperforms decision trees