In [1]:

```
import numpy as np
import pandas as pd

import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
```

In [2]:

```
df=pd.read_csv("C:/Users/Saved Janu/Downloads/archive (2)/WineQT.csv")
```

In [3]:

```
df.head()
```

Out[3]:

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

In [4]:

```
print(df.columns)
print(df.shape)
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual suga
r',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'densit
y',
       'pH', 'sulphates', 'alcohol', 'quality', 'Id'],
      dtype='object')
(1143, 13)
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1143 entries, 0 to 1142
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1143 non-null   float64
 1   volatile acidity      1143 non-null   float64
 2   citric acid           1143 non-null   float64
 3   residual sugar        1143 non-null   float64
 4   chlorides             1143 non-null   float64
 5   free sulfur dioxide   1143 non-null   float64
 6   total sulfur dioxide  1143 non-null   float64
 7   density               1143 non-null   float64
 8   pH                    1143 non-null   float64
 9   sulphates             1143 non-null   float64
 10  alcohol               1143 non-null   float64
 11  quality               1143 non-null   int64  
 12  Id                    1143 non-null   int64  
dtypes: float64(11), int64(2)
memory usage: 116.2 KB
```

In [6]:
```
df.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	75%	
fixed acidity	1143.0	8.311111	1.747595	4.60000	7.10000	7.90000	9.100000	15.
volatile acidity	1143.0	0.531339	0.179633	0.12000	0.39250	0.52000	0.640000	1.
citric acid	1143.0	0.268364	0.196686	0.00000	0.09000	0.25000	0.420000	1.
residual sugar	1143.0	2.532152	1.355917	0.90000	1.90000	2.20000	2.600000	15.
chlorides	1143.0	0.086933	0.047267	0.01200	0.07000	0.07900	0.090000	0.
free sulfur dioxide	1143.0	15.615486	10.250486	1.00000	7.00000	13.00000	21.000000	68.
total sulfur dioxide	1143.0	45.914698	32.782130	6.00000	21.00000	37.00000	61.000000	289.
density	1143.0	0.996730	0.001925	0.99007	0.99557	0.99668	0.997845	1.
pH	1143.0	3.311015	0.156664	2.74000	3.20500	3.31000	3.400000	4.
sulphates	1143.0	0.657708	0.170399	0.33000	0.55000	0.62000	0.730000	2.
alcohol	1143.0	10.442111	1.082196	8.40000	9.50000	10.20000	11.100000	14.
quality	1143.0	5.657043	0.805824	3.00000	5.00000	6.00000	6.000000	8.
Id	1143.0	804.969379	463.997116	0.00000	411.00000	794.00000	1209.500000	1597.

In [7]:
```
df.nunique()
```

Out[7]:
```
fixed acidity             91
volatile acidity         135
citric acid               77
residual sugar            80
chlorides                131
free sulfur dioxide       53
total sulfur dioxide     138
density                  388
pH                        87
sulphates                 89
alcohol                   61
quality                    6
Id                      1143
dtype: int64
```

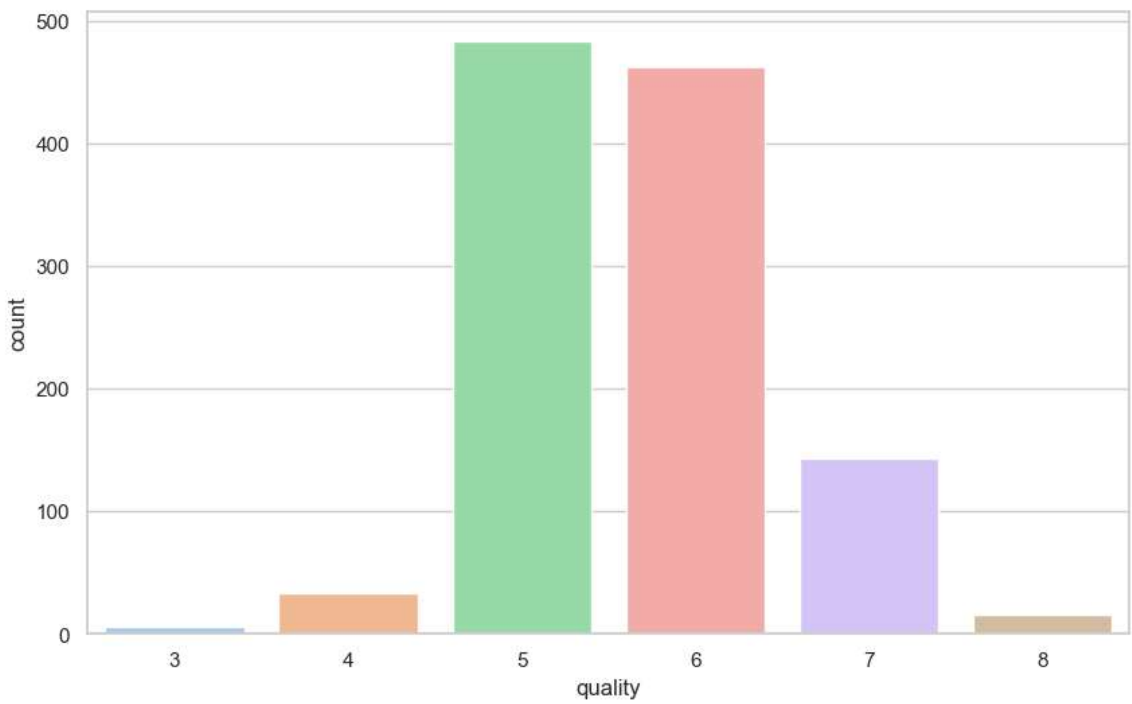
In [8]:
```
df.duplicated().sum()
```

Out[8]:
```
0
```

In [10]:
```
sns.set(style="whitegrid")
print(df['quality'].value_counts())
fig = plt.figure(figsize = (10,6))
sns.countplot( data=df,x='quality', palette='pastel')
```

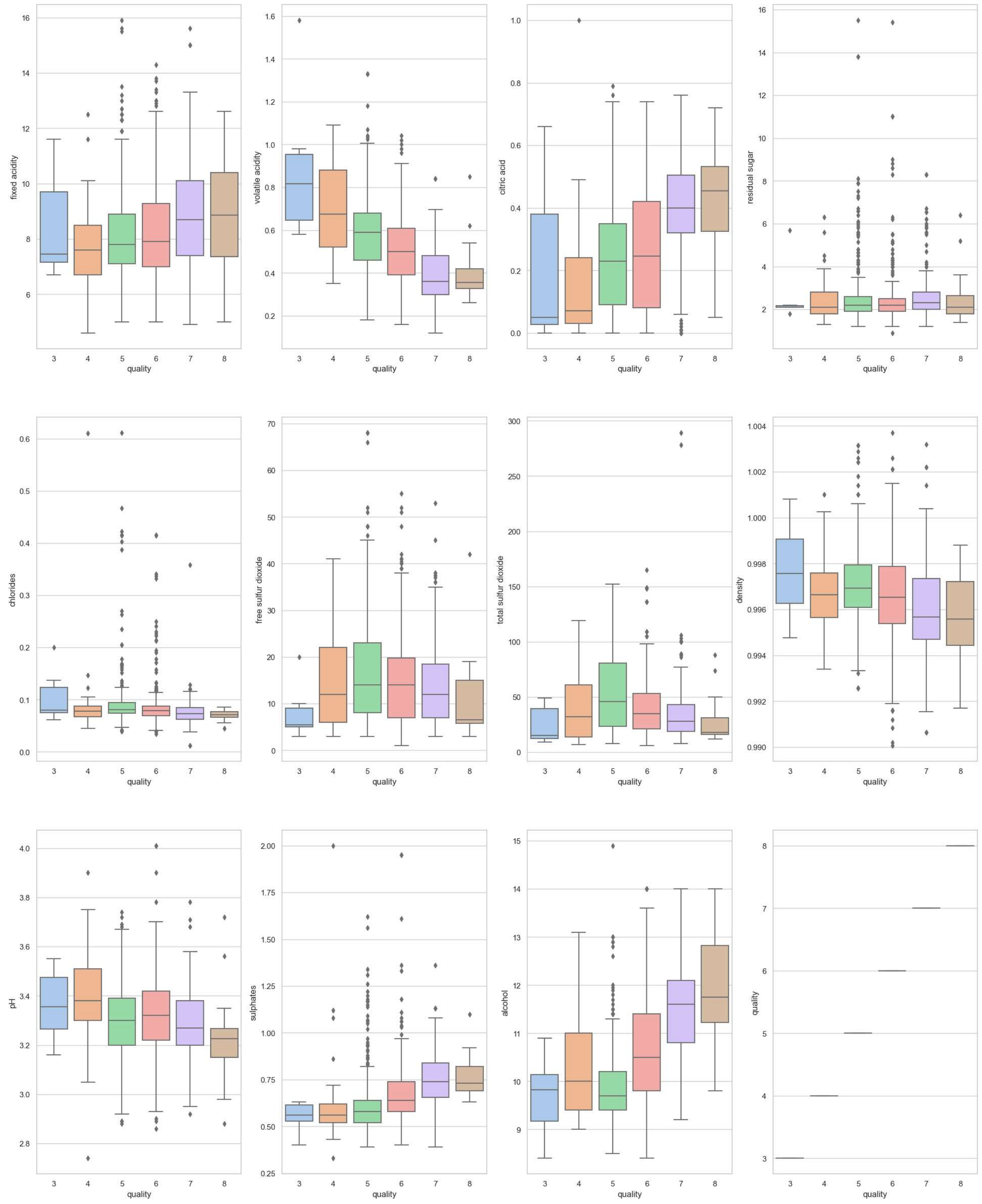
```
5    483
6    462
7    143
4     33
8     16
3      6
Name: quality, dtype: int64
```

Out[10]:
```
<Axes: xlabel='quality', ylabel='count'>
```



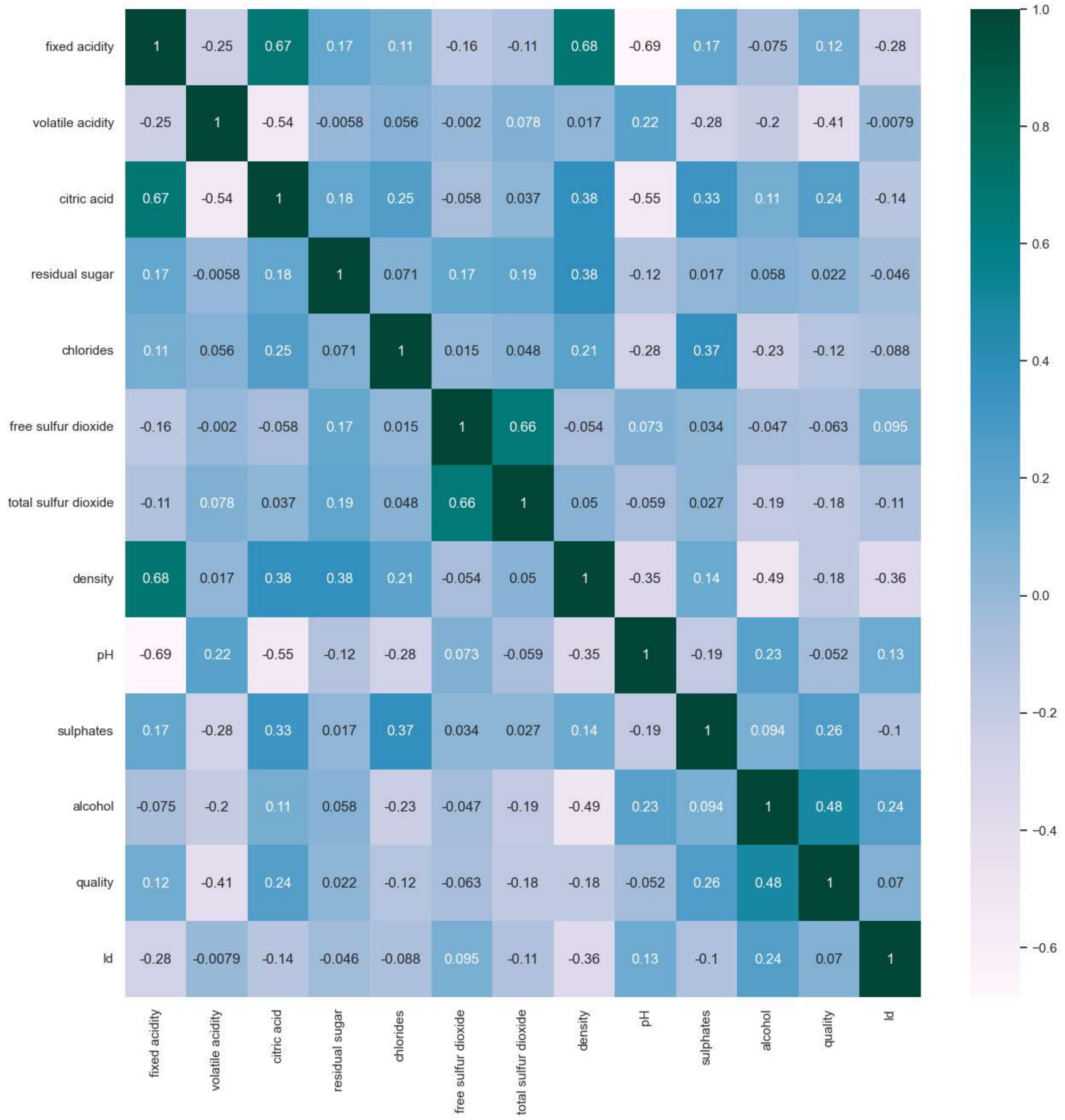
In [12]:
```
import warnings
warnings.filterwarnings("ignore")

sns.set(style="whitegrid")
fig, ax1 = plt.subplots(3,4, figsize=(24,30))
k = 0
columns = list(df.columns)
for i in range(3):
    for j in range(4):
            sns.boxplot(data=df,x='quality', y=columns[k], ax = ax1[i][j],
            k += 1
plt.show()
```



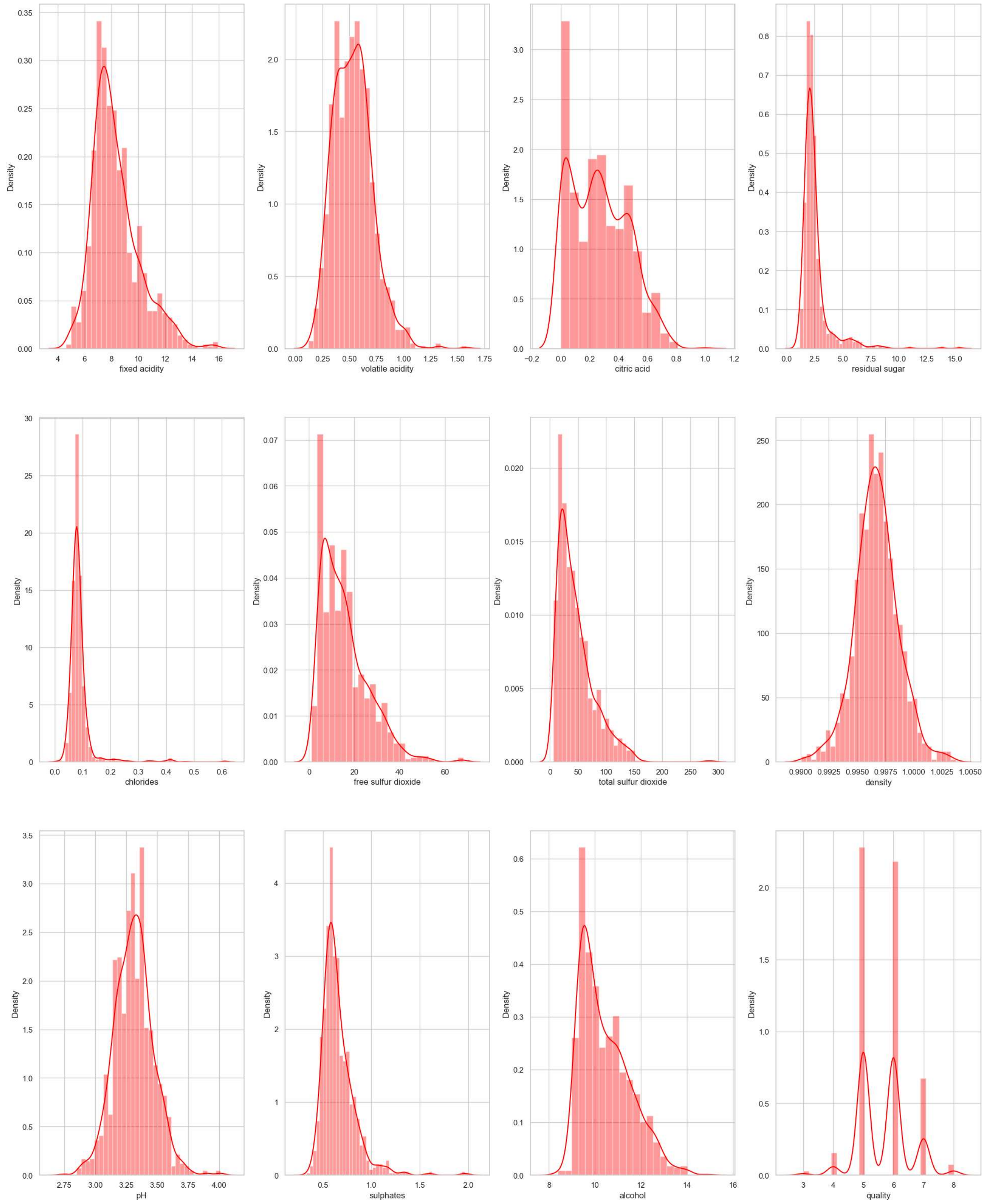
In [13]:
```
plt.figure(figsize = (15,15))
sns.heatmap(df.corr(),annot=True, cmap= 'PuBuGn')
```

Out[13]: <Axes: >



In [14]:
```
color = sns.color_palette("pastel")

fig, ax1 = plt.subplots(3,4, figsize=(24,30))
k = 0
columns = list(df.columns)
for i in range(3):
    for j in range(4):
            sns.distplot(df[columns[k]], ax = ax1[i][j], color = 'red')
            k += 1
plt.show()
```



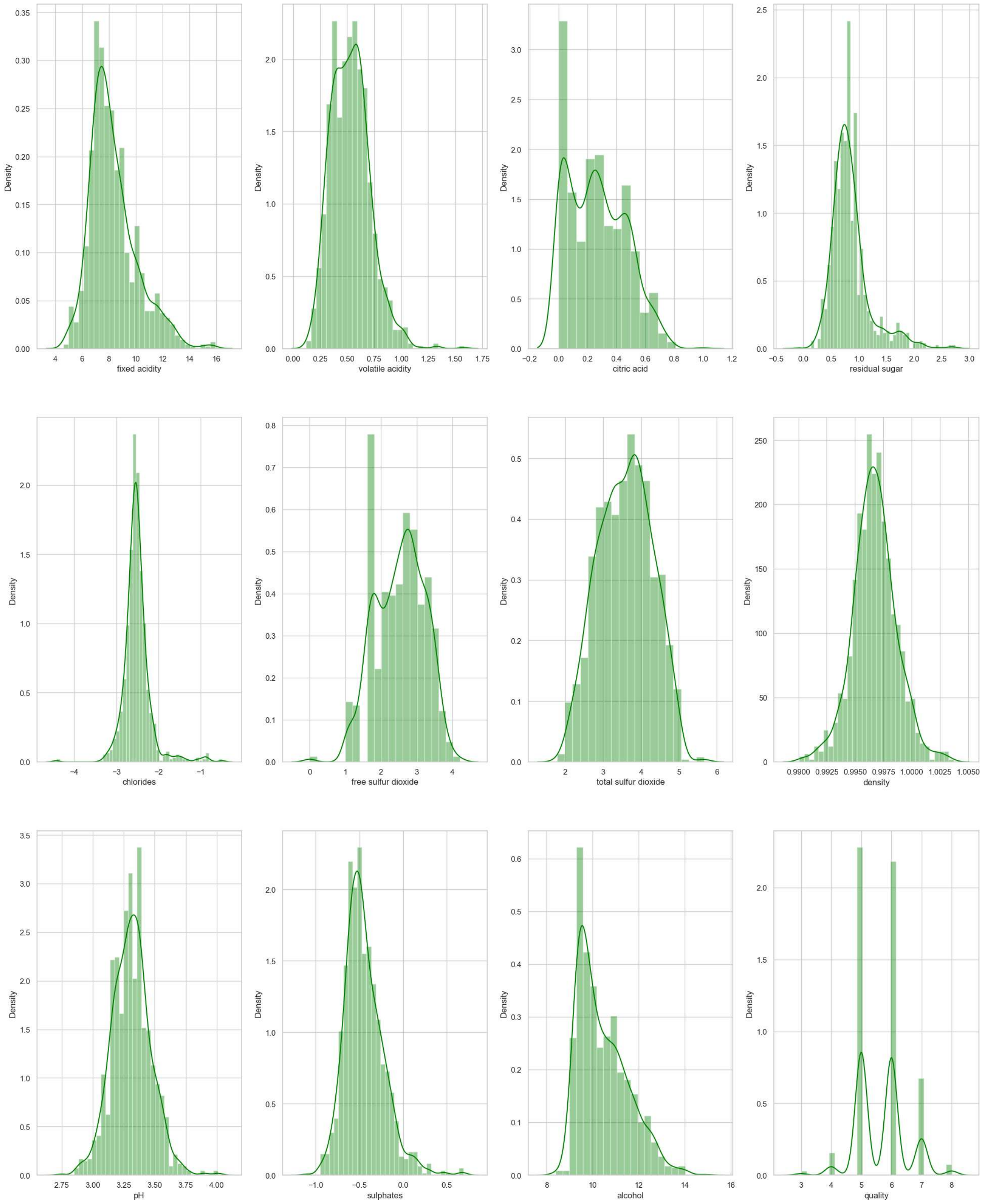
In [15]:
```
def log_transform(col):
    return np.log(col[0])

df['residual sugar'] = df[['residual sugar']].apply(log_transform, axis=1)
df['chlorides'] = df[['chlorides']].apply(log_transform, axis=1)
df['free sulfur dioxide'] = df[['free sulfur dioxide']].apply(log_transform
df['total sulfur dioxide'] = df[['total sulfur dioxide']].apply(log_transfo
df['sulphates'] = df[['sulphates']].apply(log_transform, axis=1)
```

In [16]:

```
color = sns.color_palette("pastel")

fig, ax1 = plt.subplots(3,4, figsize=(24,30))
k = 0
columns = list(df.columns)
for i in range(3):
    for j in range(4):
            sns.distplot(df[columns[k]], ax = ax1[i][j], color = 'green')
            k += 1
plt.show()
```



In [17]:
```
df.corr()['quality'].sort_values(ascending=False)
```

Out[17]:
```
quality                 1.000000
alcohol                 0.484866
sulphates               0.315097
citric acid             0.240821
fixed acidity           0.121970
Id                      0.069708
residual sugar          0.031487
pH                     -0.052453
free sulfur dioxide    -0.054185
total sulfur dioxide   -0.170128
density                -0.175208
chlorides              -0.175391
volatile acidity       -0.407394
Name: quality, dtype: float64
```

In [18]:
```
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 [20]:
```
from sklearn.utils import resample

df_3_upsampled = resample(df_3, replace=True, n_samples=600, random_state=1
df_4_upsampled = resample(df_4, replace=True, n_samples=600, random_state=1
df_7_upsampled = resample(df_7, replace=True, n_samples=600, random_state=1
df_8_upsampled = resample(df_8, replace=True, n_samples=600, random_state=1

# Decreases the rows of Majority one's to make balance data :
df_5_downsampled = df[df.quality==5].sample(n=600,replace=True).reset_index
df_6_downsampled = df[df.quality==6].sample(n=600,replace=True).reset_index
```

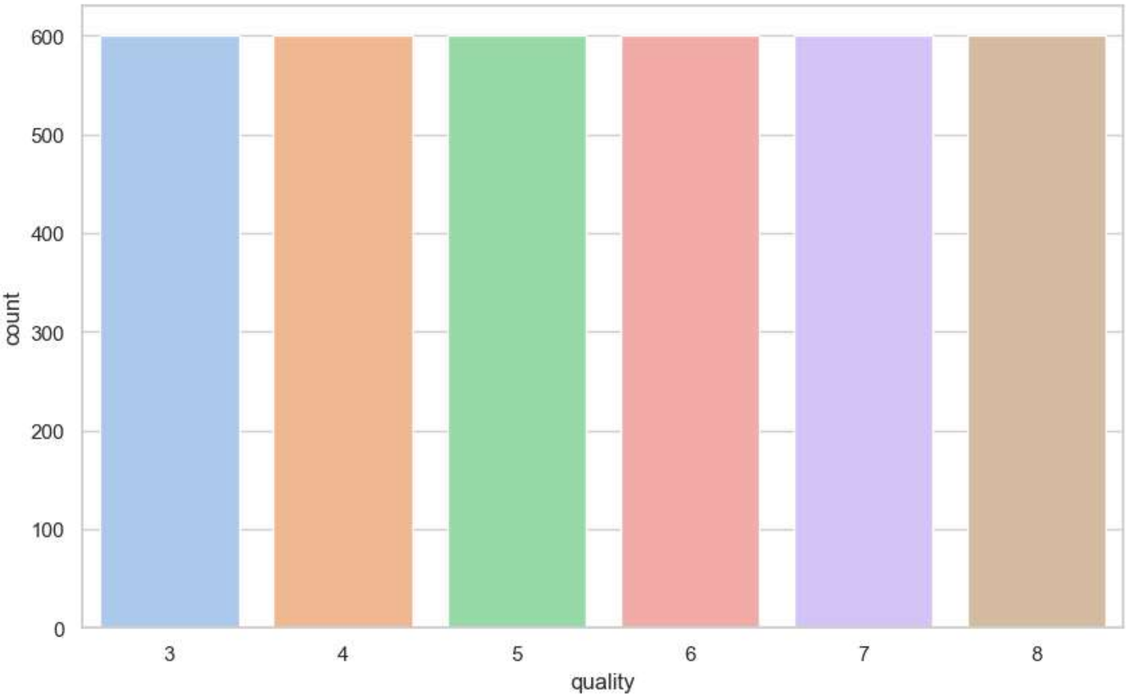
In [21]:
```
Balanced_df = pd.concat([df_3_upsampled, df_4_upsampled, df_7_upsampled,
                         df_8_upsampled, df_5_downsampled, df_6_downsampled


# Display new class counts
Balanced_df.quality.value_counts()
```

Out[21]:
```
3    600
4    600
7    600
8    600
5    600
6    600
Name: quality, dtype: int64
```

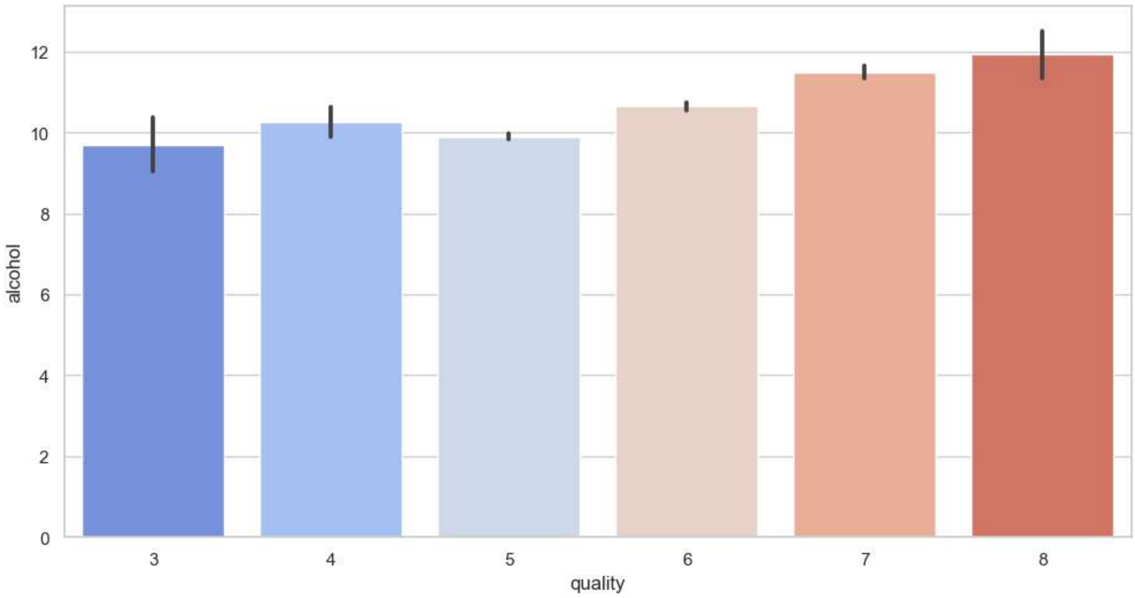
In [22]:
```
plt.figure(figsize=(10,6))
sns.countplot(x='quality', data=Balanced_df, order=[3, 4, 5, 6, 7, 8], pale
```

Out[22]: <Axes: xlabel='quality', ylabel='count'>



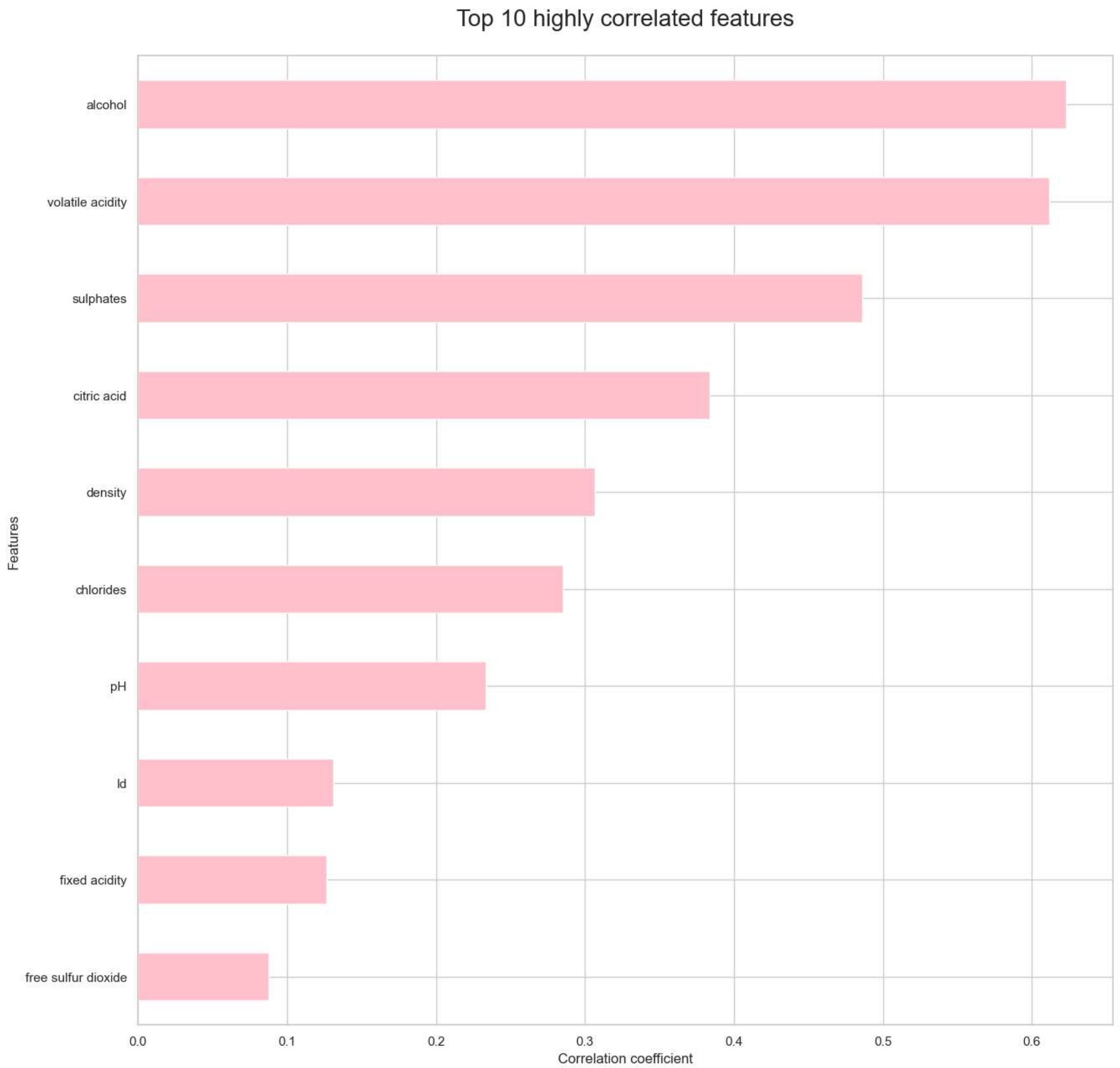
In [23]:
```
plt.figure(figsize = (12,6))
sns.barplot(x='quality', y = 'alcohol', data = df, palette = 'coolwarm')
```

Out[23]: <Axes: xlabel='quality', ylabel='alcohol'>



In [24]:
```
plt.figure(figsize=(15,15))
Balanced_df.corr().quality.apply(lambda x: abs(x)).sort_values(ascending=Fa
# calculating the top 10 highest correlated features
# with respect to the target variable i.e. "quality"
plt.title("Top 10 highly correlated features", size=20, pad=26)
plt.xlabel("Correlation coefficient")
plt.ylabel("Features")
```

Out[24]: Text(0, 0.5, 'Features')



In [25]:
```
selected_features = ['fixed acidity', 'volatile acidity', 'citric acid', 'c
                     'free sulfur dioxide', 'total sulfur dioxide', 'densit
                     'sulphates', 'alcohol']
```

In [26]:
```
X = Balanced_df[selected_features]
y = Balanced_df.quality
```

In [27]:
```
from sklearn.model_selection import train_test_split

# Splitting the data into 70% and 30% to construct Training and Testing Data
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size=0.3,random
```

In [28]:
```
from sklearn.neighbors import KNeighborsClassifier
# For weights = 'uniform'
for n_neighbors in [5,10,15,20]:
    model = KNeighborsClassifier(n_neighbors)
    model.fit(X_train, y_train)
    scr = model.score(X_test, y_test)
    print("For n_neighbors = ", n_neighbors ," score is ", scr)
```

```
For n_neighbors =  5  score is  0.8490740740740741
For n_neighbors =  10  score is  0.7898148148148149
For n_neighbors =  15  score is  0.7592592592592593
For n_neighbors =  20  score is  0.7222222222222222
```

In [29]:
```
# For weights = 'distance'
for n_neighbors in [5,10,15,20]:
    model = KNeighborsClassifier(n_neighbors, weights='distance')
    model.fit(X_train, y_train)
    scr = model.score(X_test, y_test)
    print("For n_neighbors = ", n_neighbors ," score is ",scr)
```

```
For n_neighbors =  5  score is  0.9416666666666667
For n_neighbors =  10  score is  0.9425925925925925
For n_neighbors =  15  score is  0.9324074074074075
For n_neighbors =  20  score is  0.9296296296296296
```

In [30]:
```
# Creating a k-nearest neighbors Classifier
KNN_Model = KNeighborsClassifier(n_neighbors=5, weights='distance')

# Train the model using the training set
KNN_Model.fit(X_train, y_train)
results = KNN_Model.fit(X_train, y_train)
```

In [31]:
```
KNN_train_predictions = KNN_Model.predict(X_train)
```

In [32]:
```
KNN_test_predictions = KNN_Model.predict(X_test)
```

In [33]:
```
from sklearn.metrics import classification_report, confusion_matrix

print("\n Train Data: KNN_Confusion Matrix:\n ")
print(confusion_matrix(y_train, KNN_train_predictions))

print("\n Train Data: KNN_Classification Report:\n ")
print(classification_report(y_train, KNN_train_predictions))

print("\n \n Test Data: KNN_Confusion Matrix: \n ")
print(confusion_matrix(y_test, KNN_test_predictions))

print("\n Test Data: KNN_Classification Report:\n ")
print(classification_report(y_test, KNN_test_predictions))
```

```
 Train Data: KNN_Confusion Matrix:

[[422   0   0   0   0   0]
 [  0 392   0   0   0   0]
 [  0   0 423   0   0   0]
 [  0   0   0 436   0   0]
 [  0   0   0   0 423   0]
 [  0   0   0   0   0 424]]

 Train Data: KNN_Classification Report:

              precision    recall  f1-score   support

           3       1.00      1.00      1.00       422
           4       1.00      1.00      1.00       392
           5       1.00      1.00      1.00       423
           6       1.00      1.00      1.00       436
           7       1.00      1.00      1.00       423
           8       1.00      1.00      1.00       424

    accuracy                           1.00      2520
   macro avg       1.00      1.00      1.00      2520
weighted avg       1.00      1.00      1.00      2520



 Test Data: KNN_Confusion Matrix:

[[178   0   0   0   0   0]
 [  0 208   0   0   0   0]
 [  0   8 150  16   2   1]
 [  0   1  15 134  12   2]
 [  0   0   2   4 171   0]
 [  0   0   0   0   0 176]]

 Test Data: KNN_Classification Report:

              precision    recall  f1-score   support

           3       1.00      1.00      1.00       178
           4       0.96      1.00      0.98       208
           5       0.90      0.85      0.87       177
           6       0.87      0.82      0.84       164
           7       0.92      0.97      0.94       177
           8       0.98      1.00      0.99       176

    accuracy                           0.94      1080
   macro avg       0.94      0.94      0.94      1080
weighted avg       0.94      0.94      0.94      1080
```

In [ ]: