In [1]:
```
print ("Testing")
```

```
Testing
```

Import dataset

In [2]:
```
import numpy as np
```

In [3]:
```
import pandas as pd
```

In [4]:
```
data = pd.read_csv ("c:/Users/ADMIN/Project/WineQT.csv")
```

Checking for table structure

In [5]:
```
data.head()
```

Out[5]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quali
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	

checking for uniqueness

In [6]:
```
data.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 [7]:
```
data.describe()
```

Out[7]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	
count	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143
mean	8.311111	0.531339	0.268364	2.532152	0.086933	15.615486	45.914698	(
std	1.747595	0.179633	0.196686	1.355917	0.047267	10.250486	32.782130	(
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	(
25%	7.100000	0.392500	0.090000	1.900000	0.070000	7.000000	21.000000	(
50%	7.900000	0.520000	0.250000	2.200000	0.079000	13.000000	37.000000	(
75%	9.100000	0.640000	0.420000	2.600000	0.090000	21.000000	61.000000	(
max	15.900000	1.580000	1.000000	15.500000	0.611000	68.000000	289.000000	

In [8]:
```
data.nunique() # to check for unique values
```

Out[8]:
```
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 [9]:
```
data.nunique().sort_values(ascending=True)
```

Out[9]:
```
quality                  6
free sulfur dioxide      53
alcohol                  61
citric acid              77
residual sugar           80
pH                       87
sulphates                89
fixed acidity            91
chlorides               131
volatile acidity        135
total sulfur dioxide    138
density                 388
```

```
Id                    1143
dtype: int64
```

Checking for Null values

In [10]:
```
data.isnull().sum()
```

Out[10]:
```
fixed acidity         0
volatile acidity      0
citric acid           0
residual sugar        0
chlorides             0
free sulfur dioxide   0
total sulfur dioxide  0
density               0
pH                    0
sulphates             0
alcohol               0
quality               0
Id                    0
dtype: int64
```

Removing duplicates

In [11]:
```
duplicate_rows = data[data.duplicated()]
print("# of duplicate rows:", duplicate_rows.shape[0])
```

```
# of duplicate rows: 0
```

In [12]:
```
data.columns
```

Out[12]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality', 'Id'],
      dtype='object')
```

Removing column ID that seems redundant

In [13]:
```
data.drop('Id',axis=1, inplace = True)
```

In [14]:
```
data.columns
```

Out[14]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [15]:
```
data.head()
```

Out[15]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quali
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quali
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	

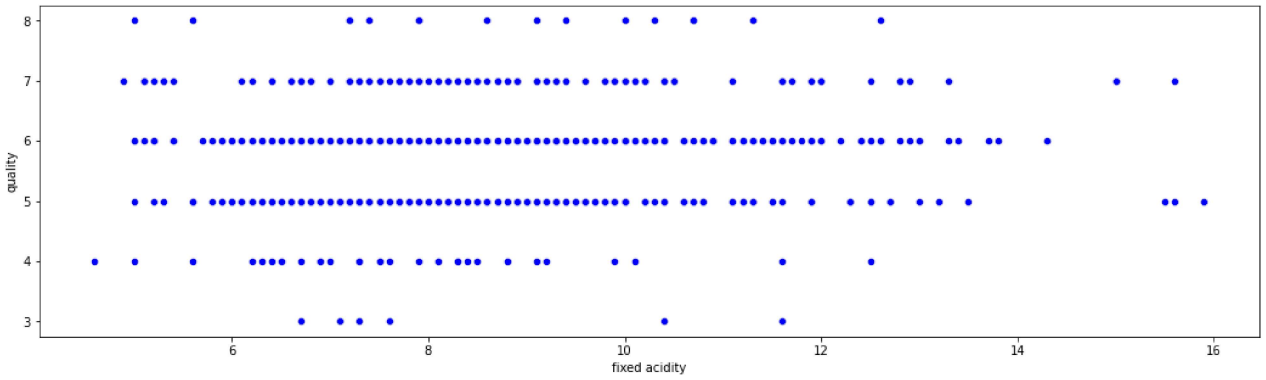
In [16]:
```
wine_features=['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol']
```

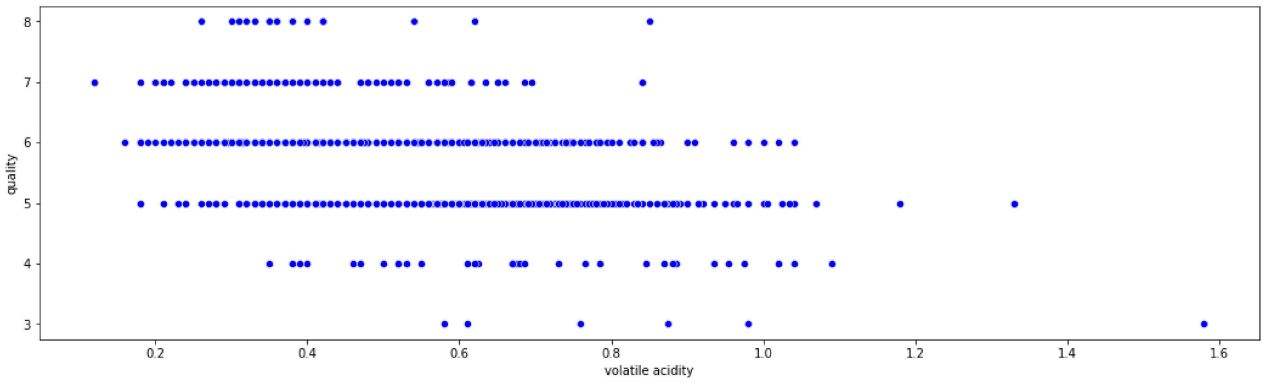
Including libraries for plotting

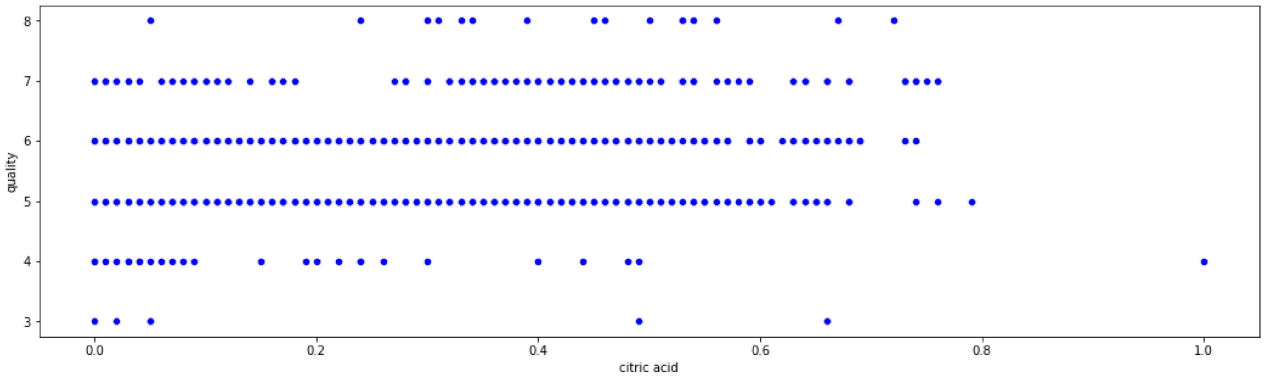
In [17]:
```
import matplotlib.pyplot as plt
```

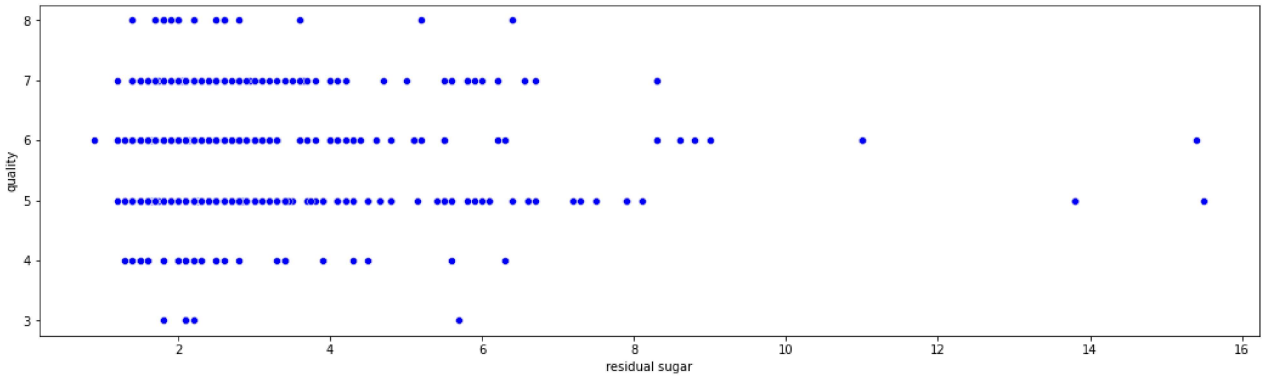
In [18]:
```
import seaborn as sns
```

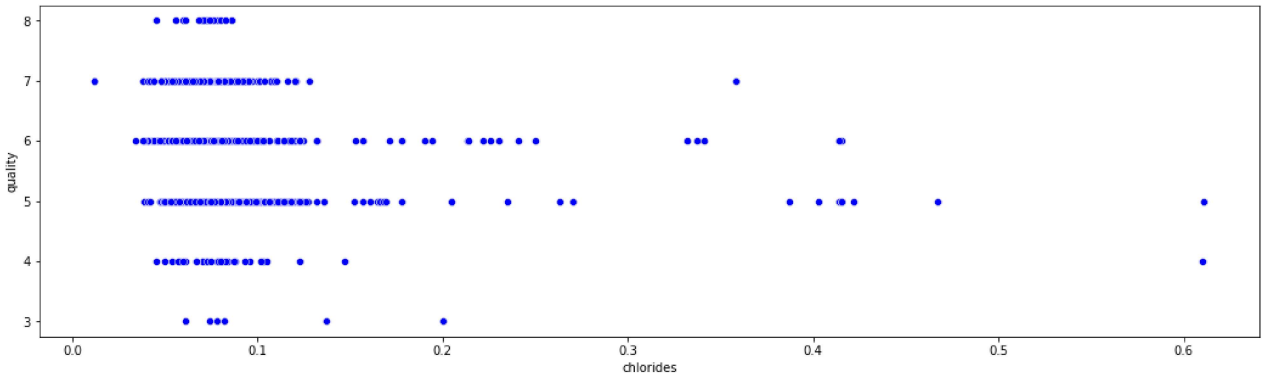
In [19]:
```
for feature in wine_features: # what is the dependency of various factors on qualtiy
    plt.figure(figsize=(18, 5))
    sns.scatterplot(x=feature,y='quality',data=data,color='blue')
    plt.show()
```

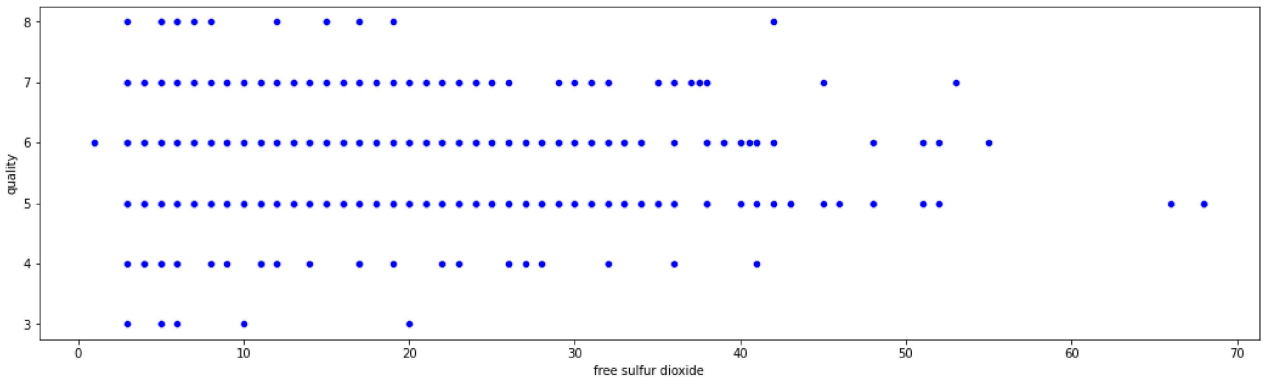


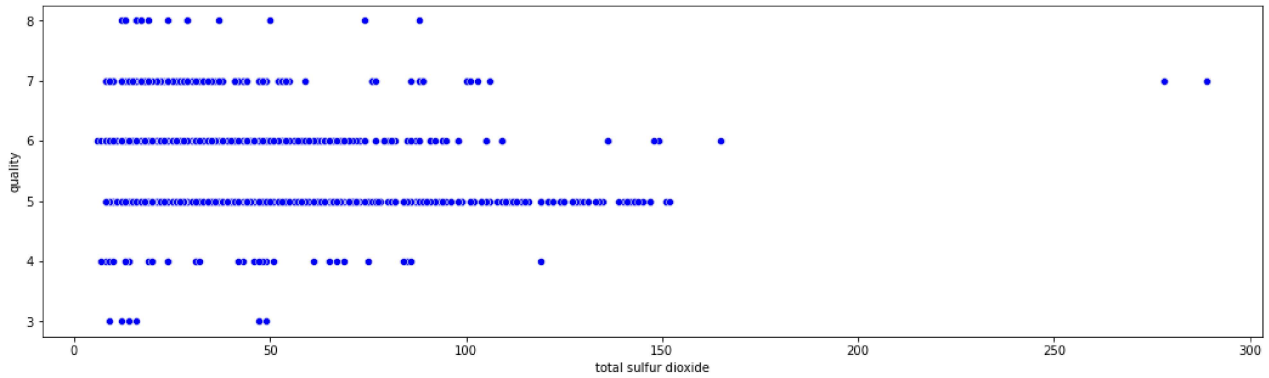


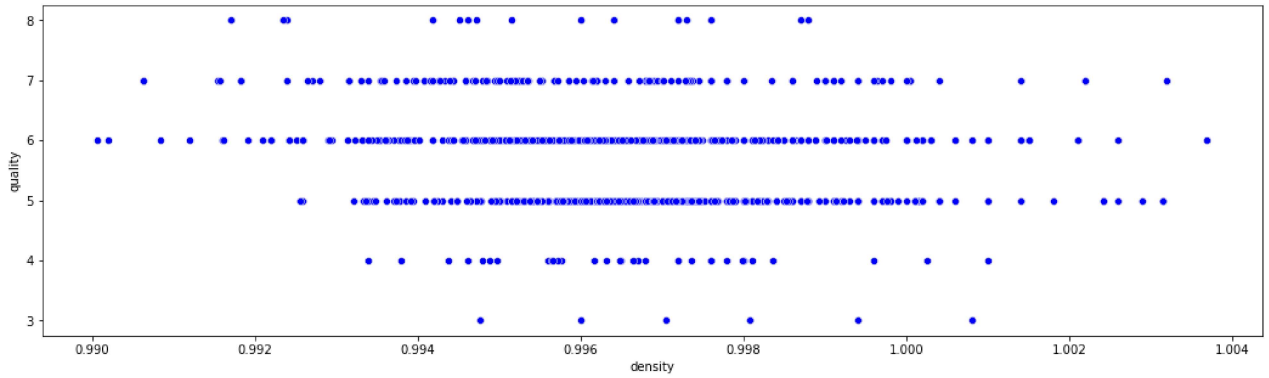

quality
citric acid

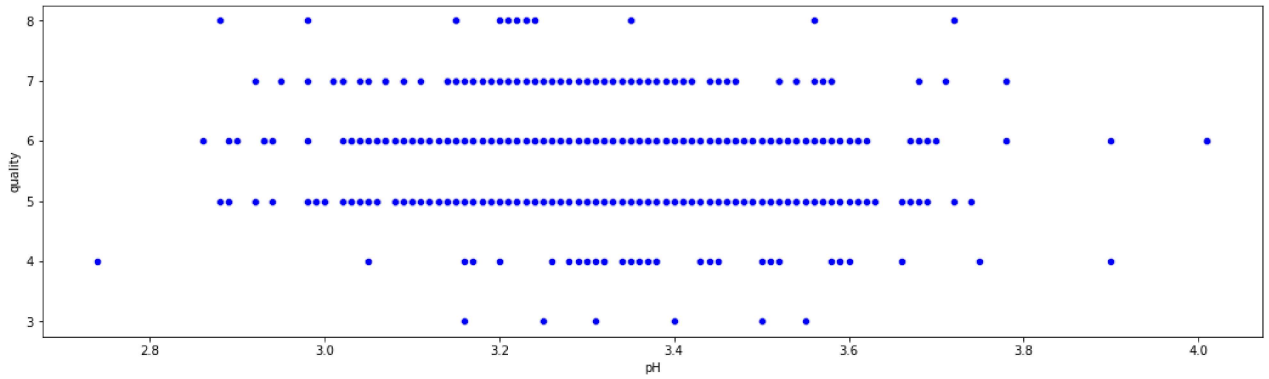

quality
residual sugar

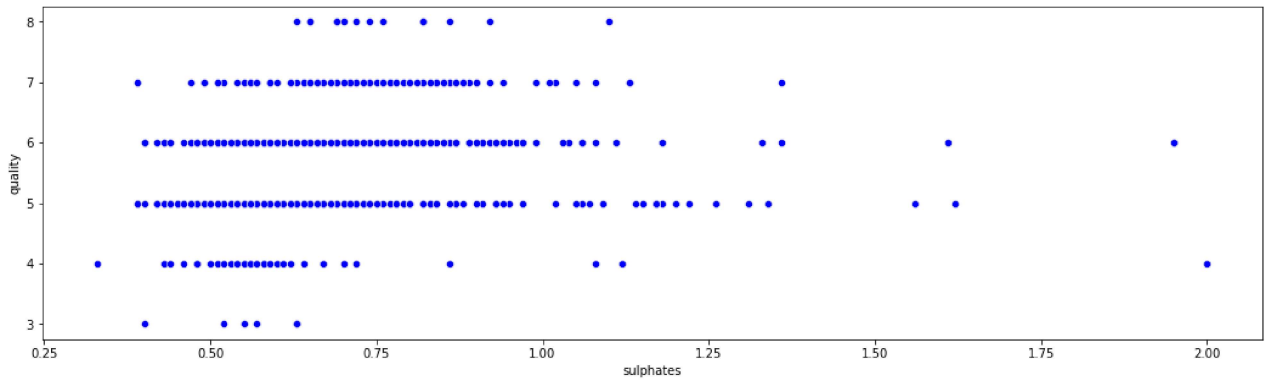

quality
chlorides

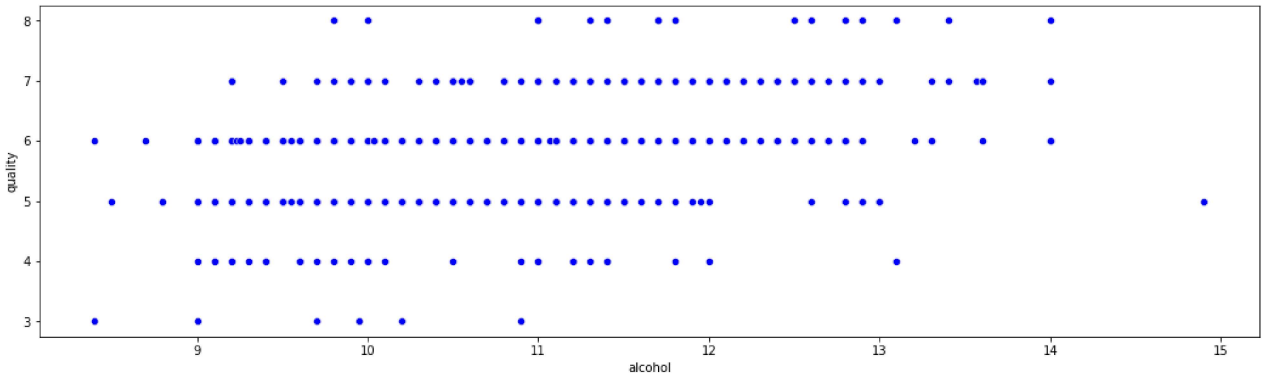

quality
free sulfur dioxide


quality
total sulfur dioxide


quality
density


quality
pH

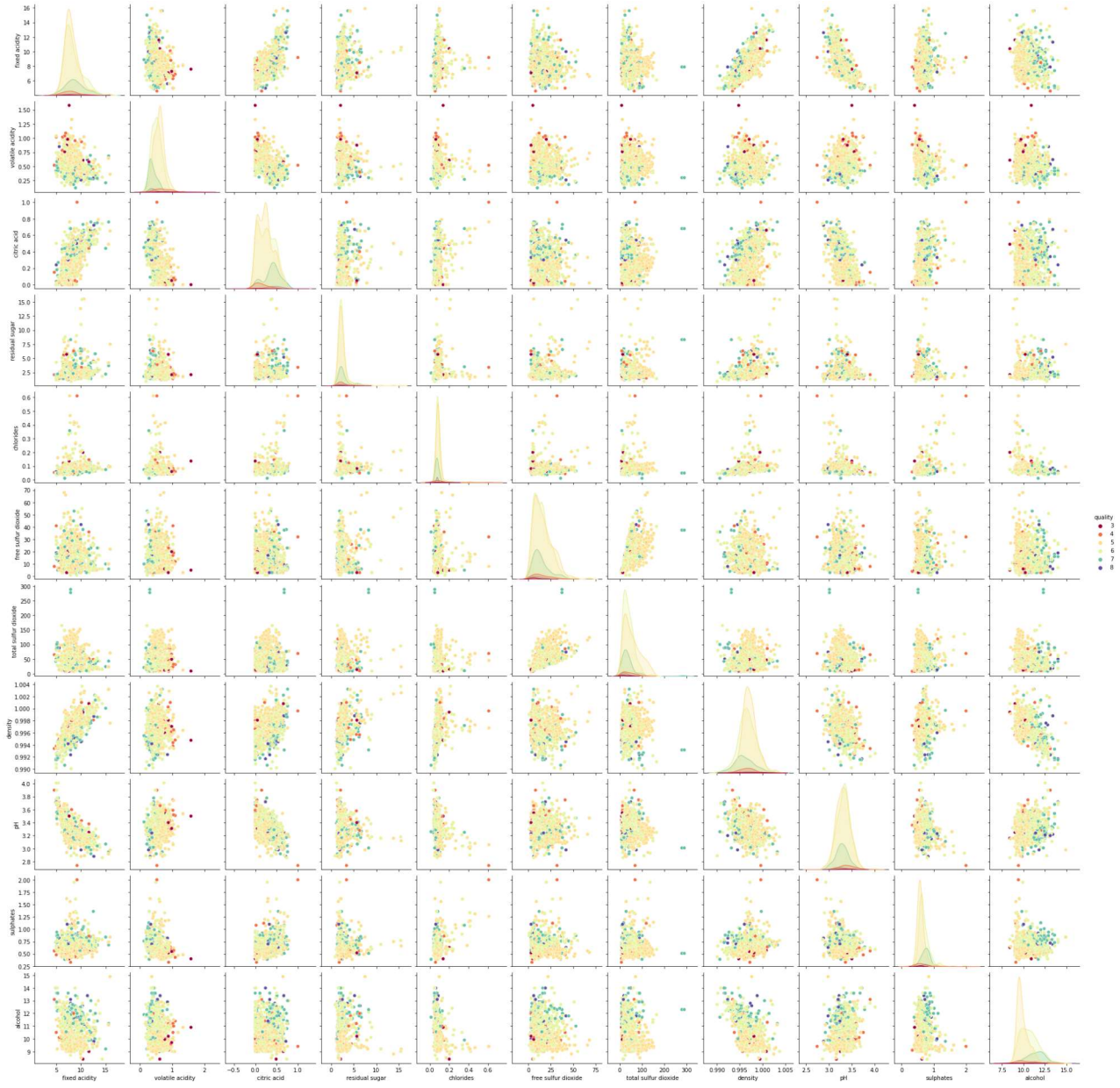

quality
sulphates



To check correlation of features

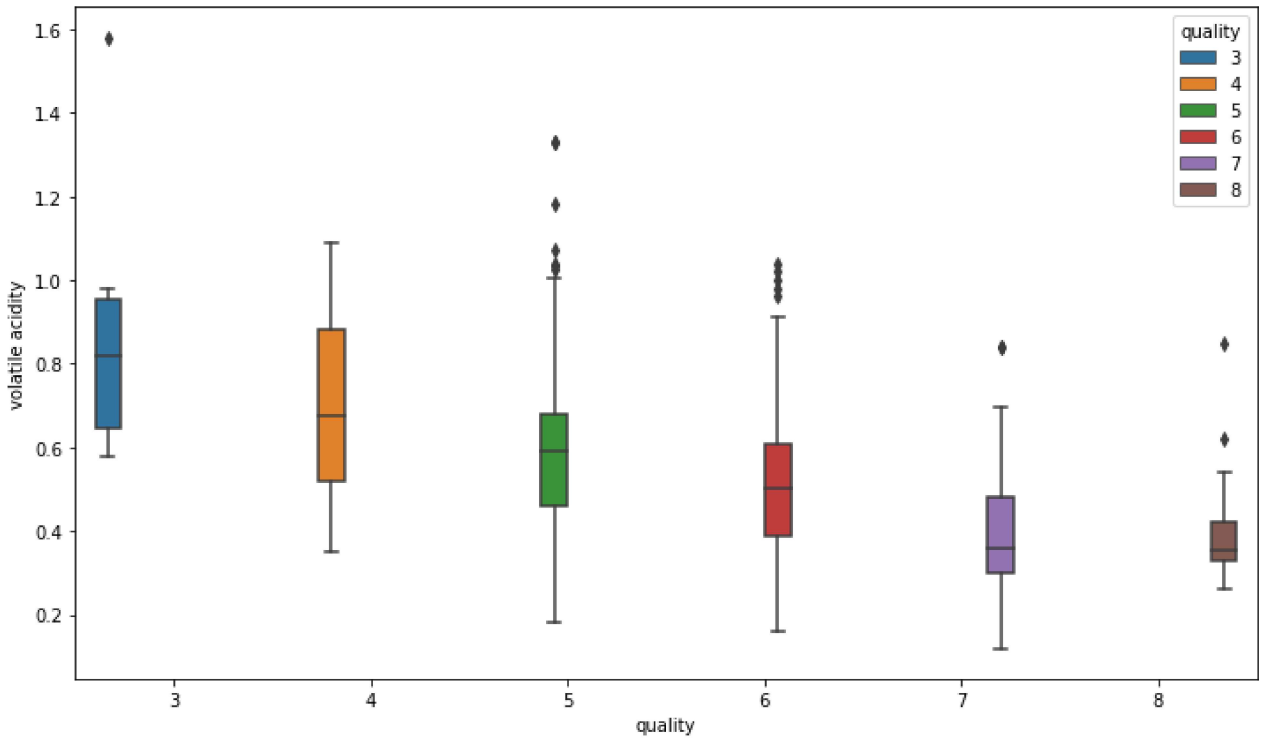
In [20]:
```
sns.pairplot(data,hue='quality',palette='Spectral') # to check for any correlation
```

Out[20]:
```
<seaborn.axisgrid.PairGrid at 0x1e90c587790>
```



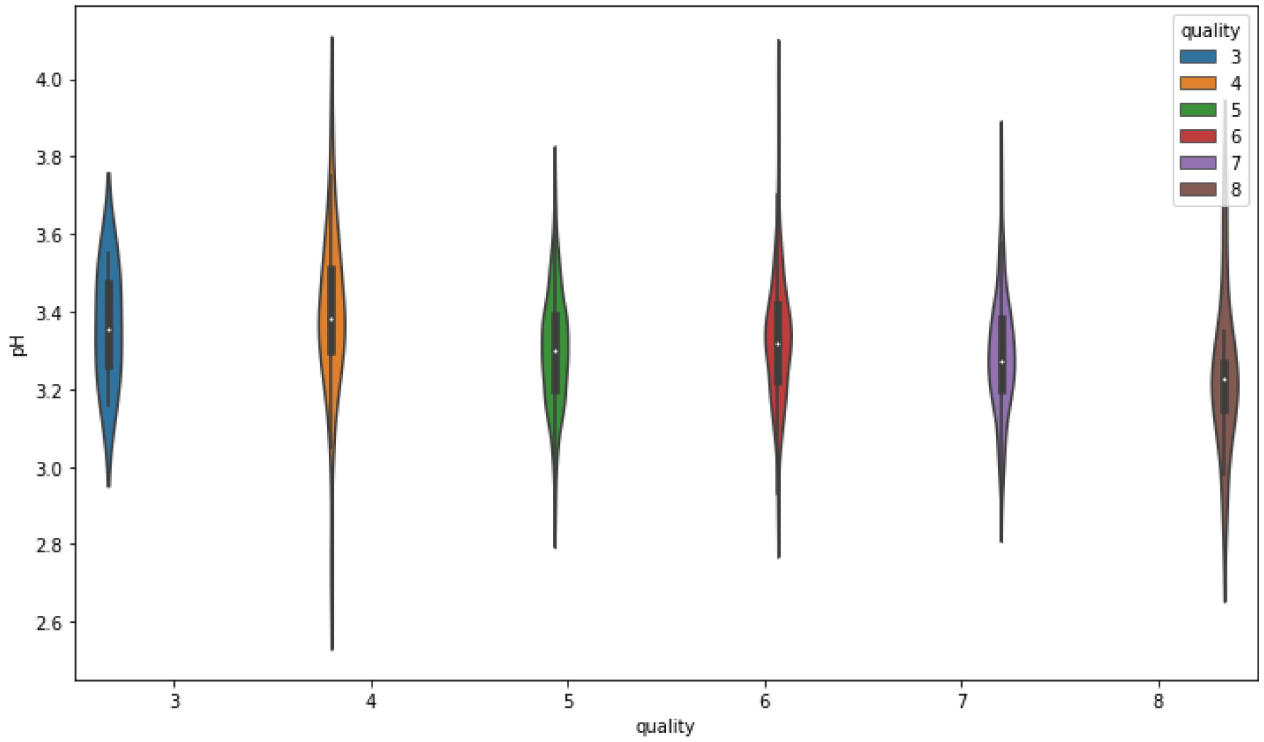
In [21]:
```
plt.figure(figsize=(12,7)) # to check for outliers
```

```
sns.boxplot(x='quality',y='volatile acidity',data=data,hue='quality')
plt.show()
```



In [22]:

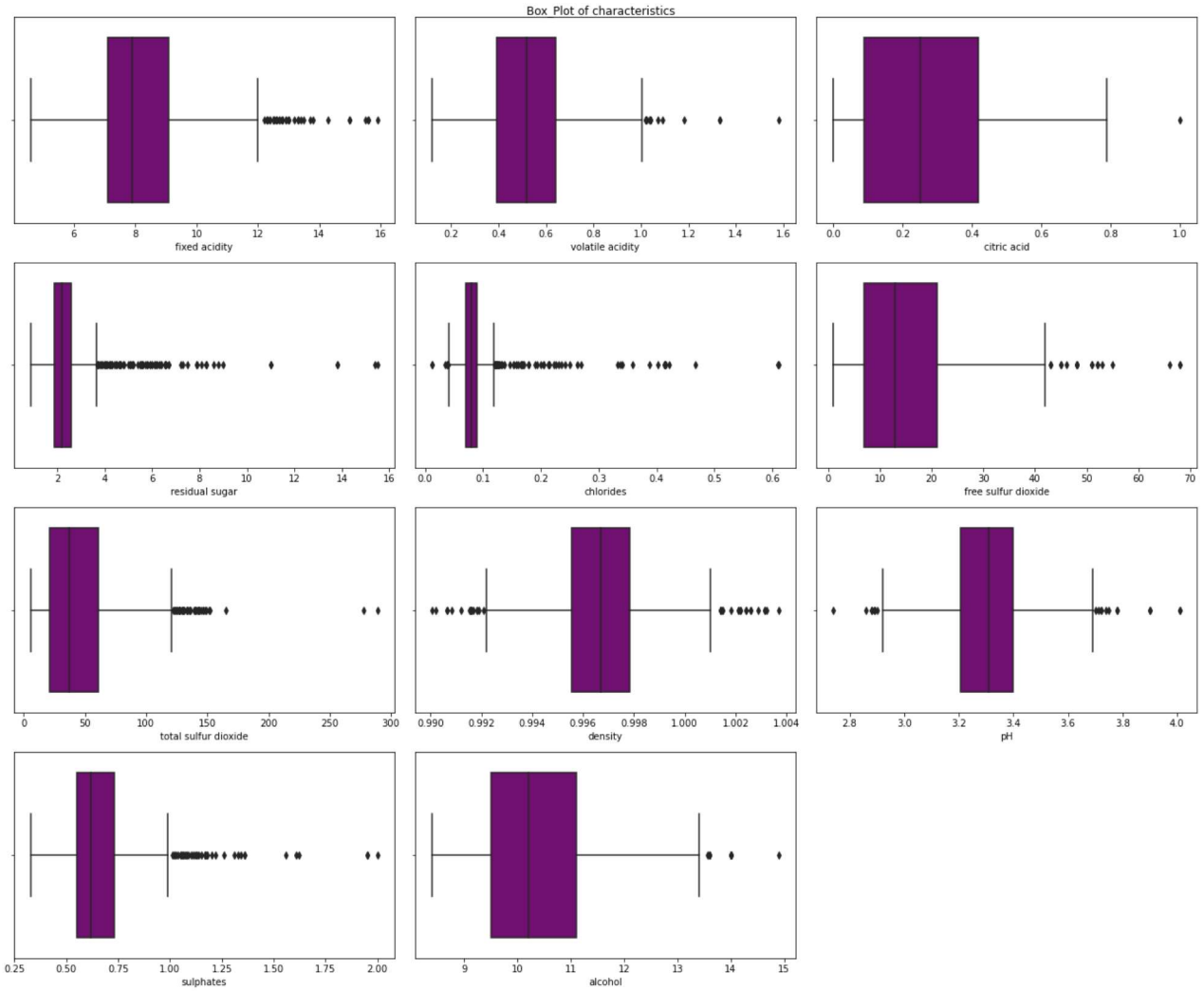
```
plt.figure(figsize=(12,7))
sns.violinplot(x='quality',y='pH',data=data,hue='quality')
plt.show()
```



In [23]:

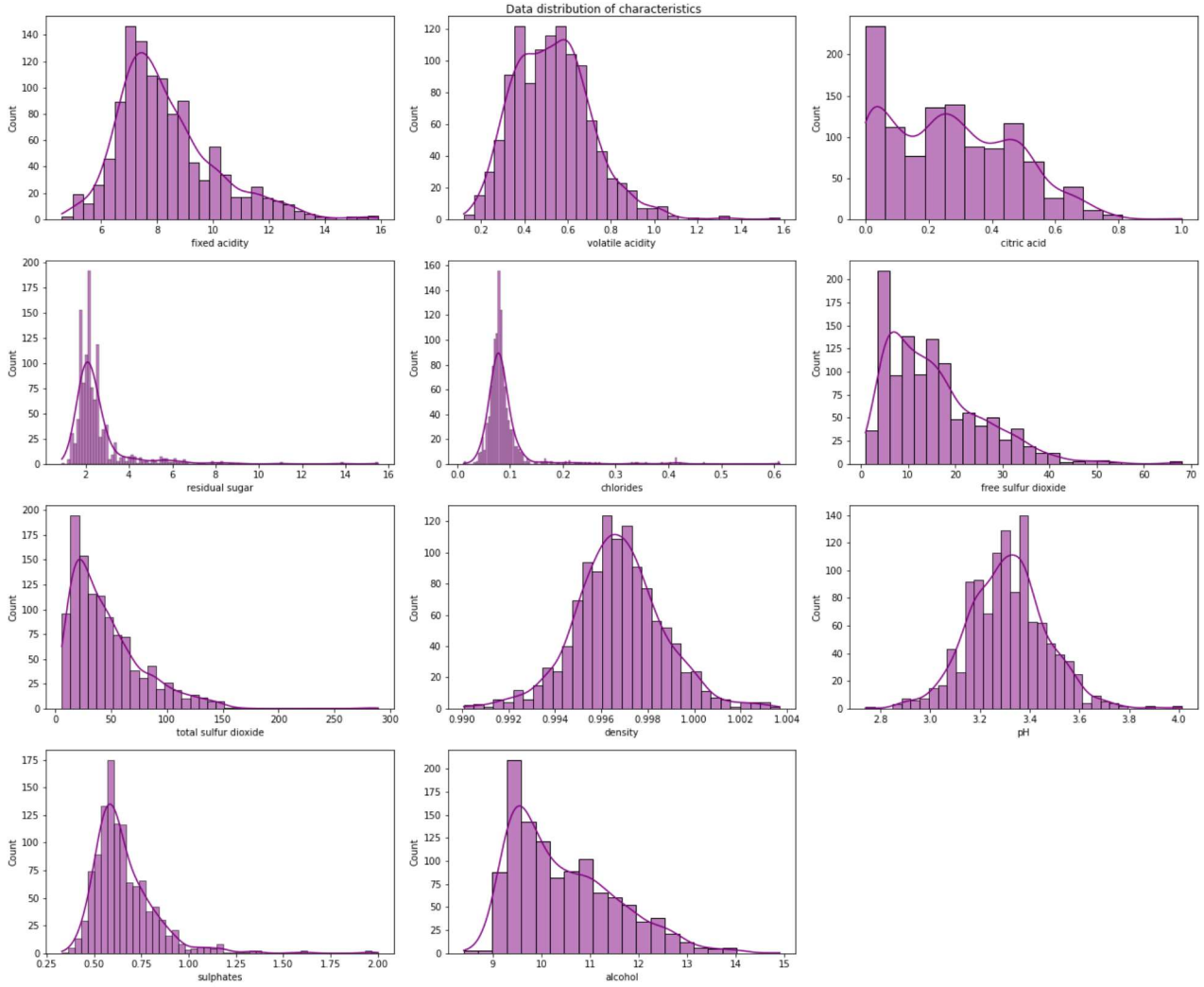
```
plt.figure(figsize=(18, 15))
for i, column in enumerate(data.select_dtypes(include=['float64']).columns):
    plt.rcParams['axes.facecolor'] = 'white'
```

```
    axis = plt.subplot(4,3, i+1)
    sns.boxplot(data=data, x=column, ax=axis,color='purple')
plt.suptitle('Box_Plot of characteristics')
plt.tight_layout()
```



In [24]:

```
plt.figure(figsize=(18, 15)) # validate the distribution of variables
for i, column in enumerate(data.select_dtypes(include=['float64']).columns):
    plt.rcParams['axes.facecolor'] = 'white'
    axis = plt.subplot(4,3, i+1)
    sns.histplot(data=data, x=column, ax=axis,kde=True, color = "purple")
plt.suptitle('Data distribution of characteristics')
plt.tight_layout()
```

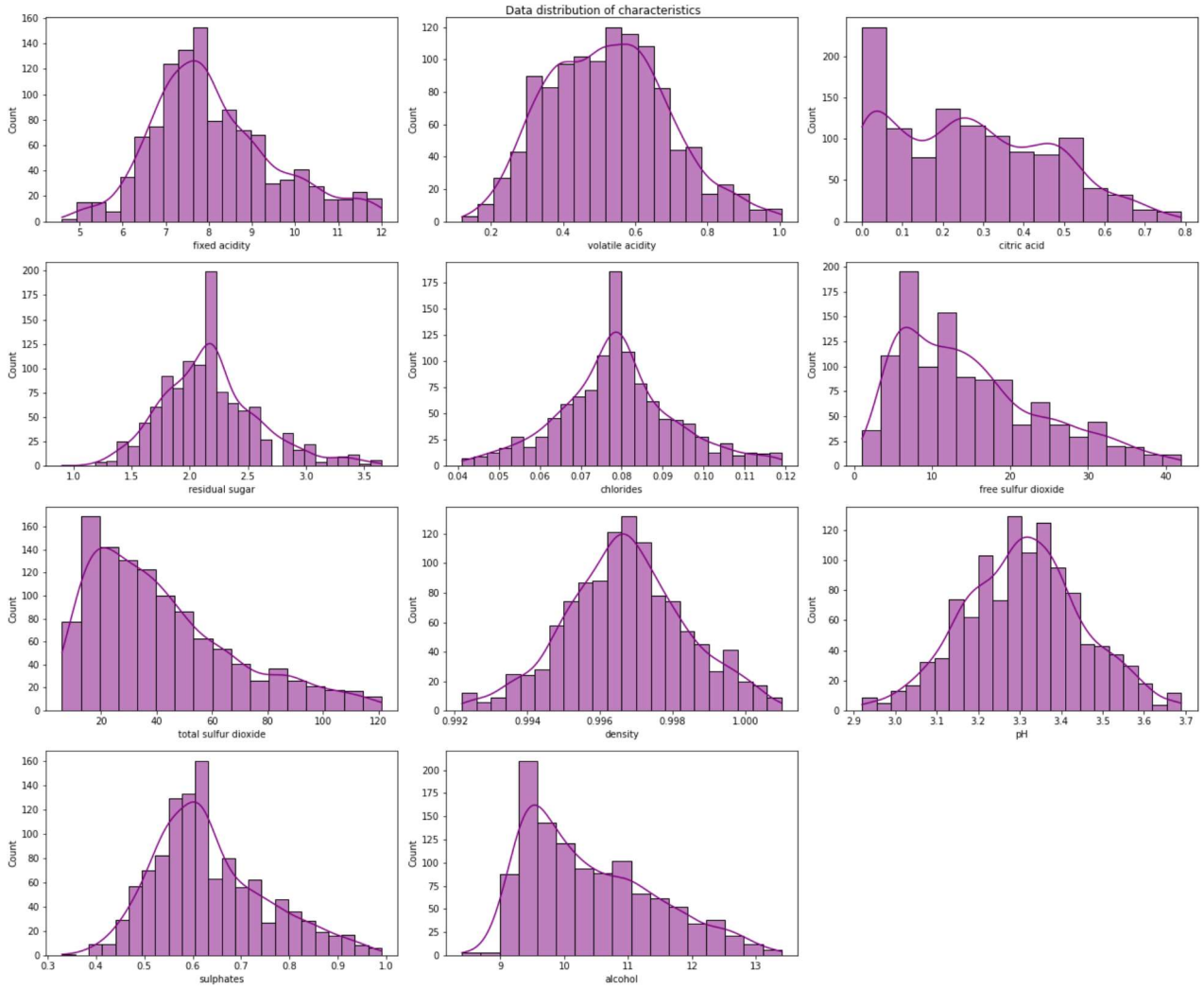


In [25]:
```
def outlier_treating(data_copy,variable): # removing outliers and replacing with median
    data=data_copy.copy()
    def outlier_detector(data_copy):
        outliers=[]
        q1=np.percentile(data_copy,25)
        q3=np.percentile(data_copy,75)
        Range=q3-q1
        lb=q1-(Range*1.5)
        ub=q3+(Range*1.5)
        for i,j in enumerate(data_copy):
            if(j<lb or j>ub):
                outliers.append(i)
        return outliers
    for i in variable:
        outlier_variable=outlier_detector(data[i])
        data.loc[outlier_variable,i]=np.median(data[i])
    return data
```

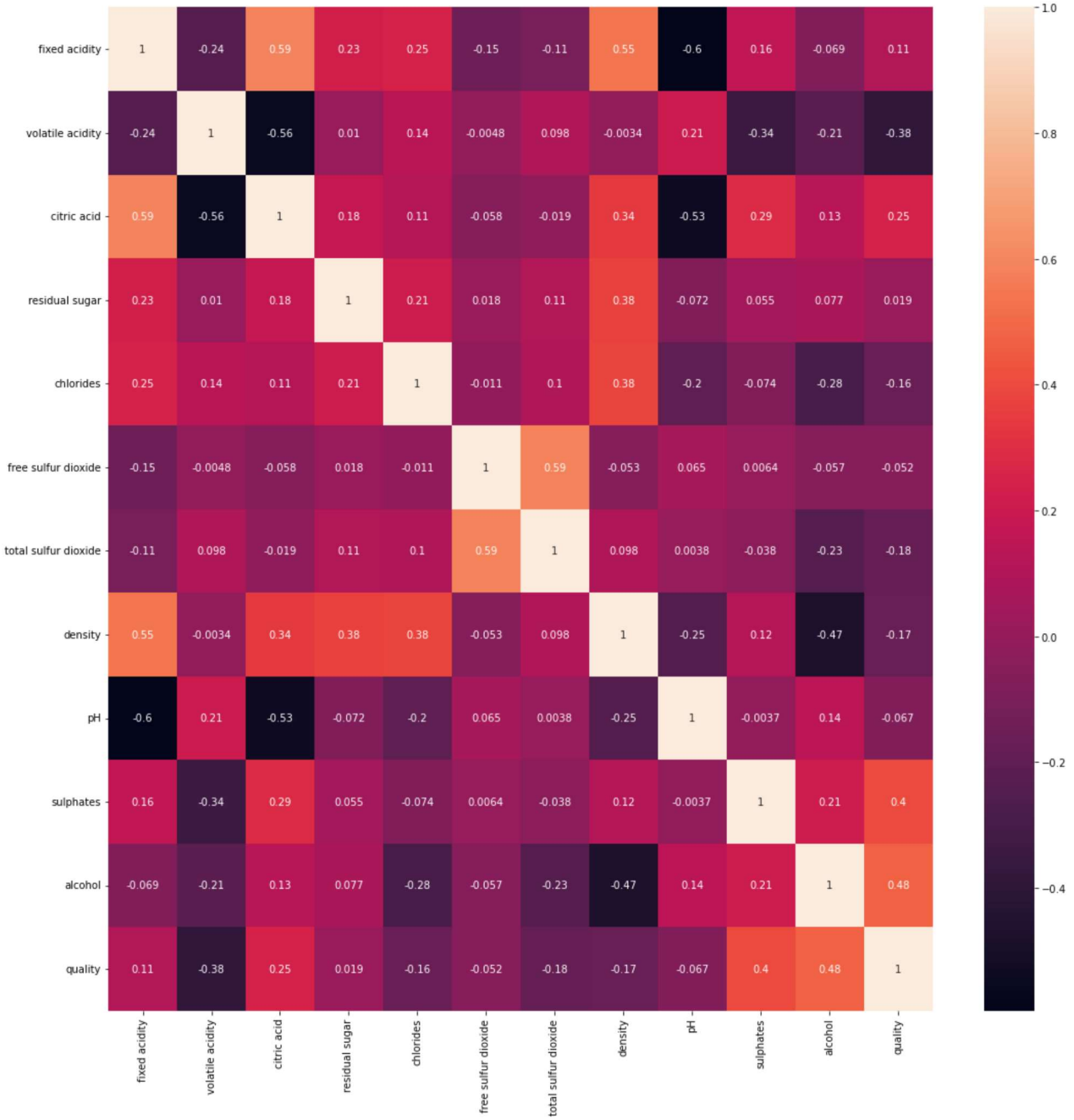
In [26]:
```
variable=list(data.select_dtypes(include=['float64']).columns)
```

In [27]:
```
data=outlier_treating(data,variable)
```

In [28]:
```
plt.figure(figsize=(18, 15)) # more uniform distribution
for i, col in enumerate(data.select_dtypes(include=['float64']).columns):
    plt.rcParams['axes.facecolor'] = 'white'
    axis = plt.subplot(4,3, i+1)
    sns.histplot(data=data, x=col, ax=axis,kde=True, color = "purple")
plt.suptitle('Data distribution of characteristics')
plt.tight_layout()
```



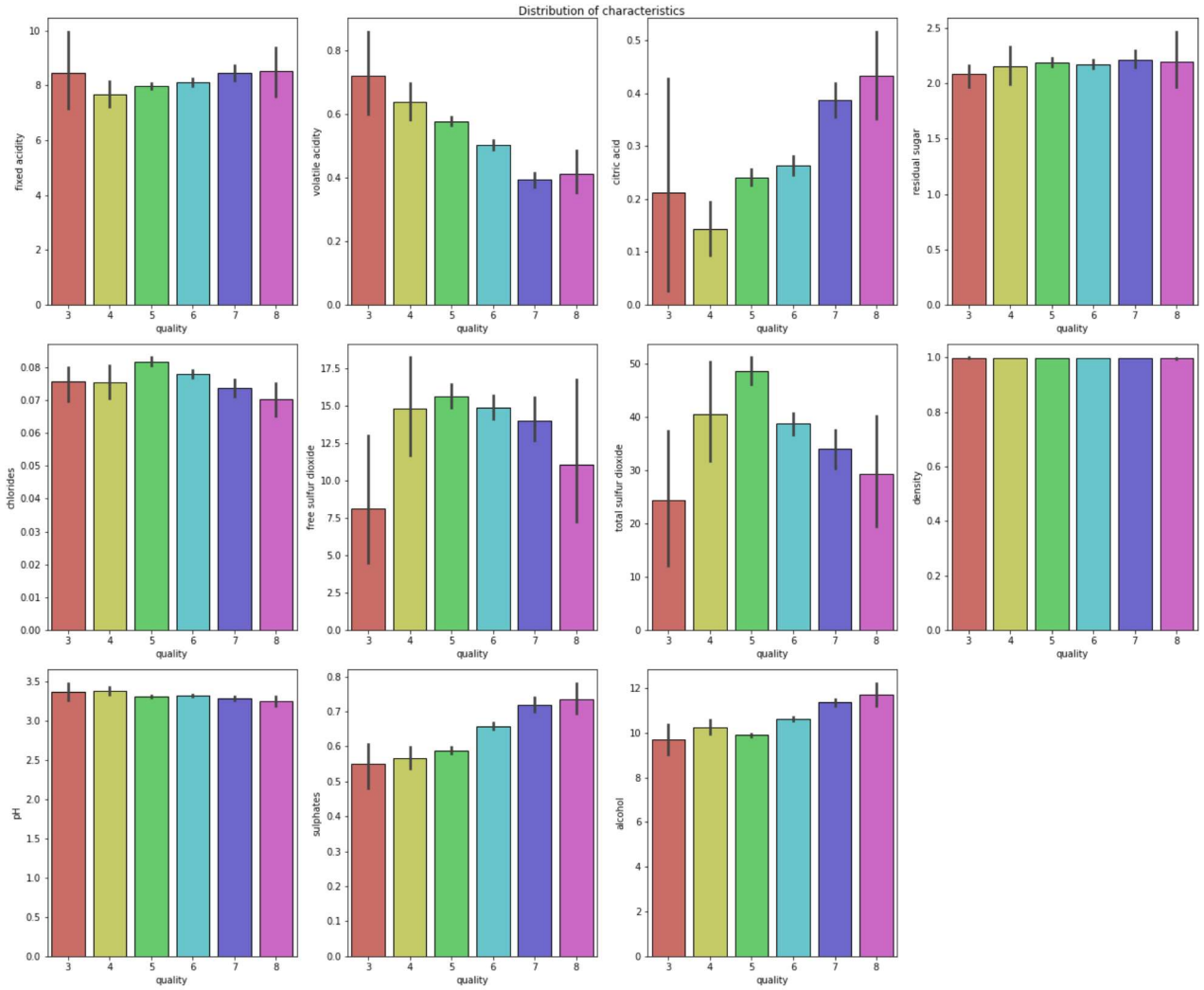
In [29]:
```
plt.figure(figsize=(18,18)) # to check for interdependency of variables
sns.heatmap(data.corr(),annot=True)
plt.show()
```



Running a loop and changing color pallette

In [30]:

```
plt.figure(figsize=(18, 15))
for i, column in enumerate(data.select_dtypes(include=['float64']).columns):
    axis = plt.subplot(3,4, i+1)
    sns.barplot(data=data,x='quality', y=column, ax=axis,edgecolor="black",palette='hls
plt.suptitle('Distribution of characteristics')
plt.tight_layout()
```



unbalanced target class

In [31]:
```
data["quality"].value_counts().reset_index().sort_values(by = "quality") #various quali
```

Out[31]:

	index	quality
5	3	6
4	8	16
3	4	33
2	7	143
1	6	462
0	5	483

In [32]:
```
from sklearn.feature_selection import SelectKBest
```

In [33]:
```
from sklearn.feature_selection import f_classif
```

In [34]:

```
X=data.iloc[:,0:-1]
y=data.iloc[:,-1]
```

In [35]:
```
! pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\admin\anaconda3\lib\site-packages
(0.0)
Requirement already satisfied: imbalanced-learn in c:\users\admin\anaconda3\lib\site-pac
kages (from imblearn) (0.9.1)
Requirement already satisfied: numpy>=1.17.3 in c:\users\admin\anaconda3\lib\site-packag
es (from imbalanced-learn->imblearn) (1.20.3)
Requirement already satisfied: scikit-learn>=1.1.0 in c:\users\admin\anaconda3\lib\site-
packages (from imbalanced-learn->imblearn) (1.1.2)
Requirement already satisfied: scipy>=1.3.2 in c:\users\admin\anaconda3\lib\site-package
s (from imbalanced-learn->imblearn) (1.7.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\admin\anaconda3\lib\site
-packages (from imbalanced-learn->imblearn) (2.2.0)
Requirement already satisfied: joblib>=1.0.0 in c:\users\admin\anaconda3\lib\site-packag
es (from imbalanced-learn->imblearn) (1.1.0)
```

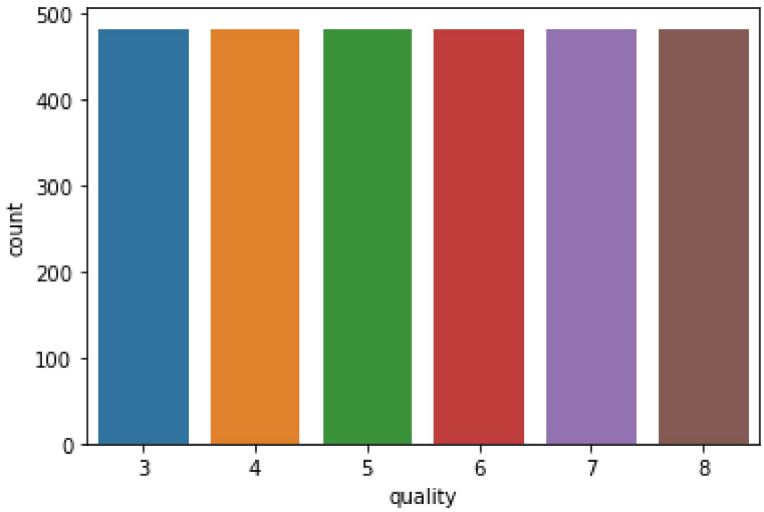
In [36]:
```
from imblearn.over_sampling import SMOTE # oversampling to remove bias
```

In [37]:
```
sm = SMOTE(sampling_strategy='auto', random_state=42)
```

In [38]:
```
X,y = sm.fit_resample(X,y)
```

In [39]:
```
sns.countplot(y)
plt.show() #synthetic expansion to equal samples
```

```
C:\Users\ADMIN\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
s the following variable as a keyword arg: x. From version 0.12, the only valid position
al argument will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```



In [40]:
```
fs = SelectKBest(score_func=f_classif, k='all')
```

```
fs.fit(X, y)
```

Out[40]:

```
SelectKBest
SelectKBest(k='all')
```

In [41]:

```
feature_contribution=(fs.scores_/sum(fs.scores_))*100
```

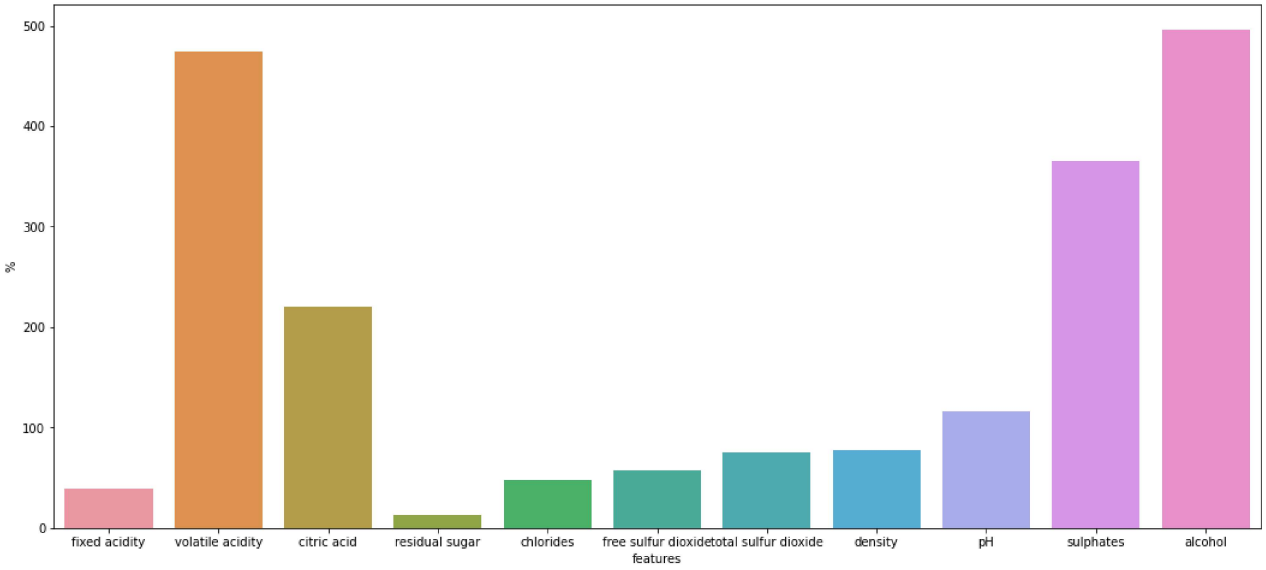
In [42]:

```
for i,j in enumerate(X.columns):
    print(f'{j} : {feature_contribution[i]:.2f}%')
plt.figure(figsize=(18,8))

sns.barplot(x=X.columns,y=fs.scores_)

#plt.title("test")
plt.xlabel("features")
plt.ylabel("%")
plt.show()
```

```
fixed acidity : 2.01%
volatile acidity : 23.89%
citric acid : 11.12%
residual sugar : 0.68%
chlorides : 2.42%
free sulfur dioxide : 2.91%
total sulfur dioxide : 3.82%
density : 3.92%
pH : 5.86%
sulphates : 18.38%
alcohol : 25.00%
```



In [43]:

```
X_S=X[['volatile acidity','citric acid','chlorides','total sulfur dioxide','density','p
```

In [44]:

```
from sklearn.model_selection import train_test_split
```

In [45]:

```
X_training,X_testing,y_training,y_testing=train_test_split(X_S,y,test_size=0.20,stratif
```

In [46]:
```
#! pip install xgboost
```

In [47]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score,f1_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
#from xgboost import XGBClassifier
```

In [48]:
```
from sklearn.preprocessing import StandardScaler
```

In [49]:
```
scr=StandardScaler()
X_training=scr.fit_transform(X_training)
X_testing=scr.transform(X_testing)
```

In [50]:
```
model_comparison={}
```

In [51]:
```
classifier=DecisionTreeClassifier(criterion = 'entropy', random_state = 1)
classifier.fit(X_training,y_training)
y_predict=classifier.predict(X_testing)
print(f"Model Accuracy : {accuracy_score(y_predict,y_testing)*100:.2f}%")
print(f"Model F1-Score : {f1_score(y_predict,y_testing,average='weighted')*100:.2f}%")
accuracies = cross_val_score(estimator = classifier, X = X_training, y = y_training, cv
print("Cross Val Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Cross Val Standard Deviation: {:.2f} %".format(accuracies.std()*100))
print(classification_report(y_predict,y_testing,zero_division=1))
model_comparison['Decision Tree']=[accuracy_score(y_predict,y_testing),f1_score(y_predi
```

```
Model Accuracy : 74.31%
Model F1-Score : 74.39%
Cross Val Accuracy: 74.55 %
Cross Val Standard Deviation: 1.15 %
              precision    recall  f1-score   support

           3       1.00      0.93      0.97       104
           4       0.78      0.84      0.81        90
           5       0.57      0.54      0.56       101
           6       0.47      0.49      0.48        94
           7       0.74      0.76      0.75        95
           8       0.89      0.89      0.89        96

    accuracy                           0.74       580
   macro avg       0.74      0.74      0.74       580
weighted avg       0.75      0.74      0.74       580
```

In [52]:

```
classifier=KNeighborsClassifier()
classifier.fit(X_training,y_training)
y_predict=classifier.predict(X_testing)
print(f"Model Accuracy : {accuracy_score(y_predict,y_testing)*100:.2f}%")
print(f"Model F1-Score : {f1_score(y_predict,y_testing,average='weighted')*100:.2f}%")
accuracies = cross_val_score(estimator = classifier, X = X_training, y = y_training, cv
print("Cross Val Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Cross Val Standard Deviation: {:.2f} %".format(accuracies.std()*100))
print(classification_report(y_predict,y_testing,zero_division=1))
model_comparison['KNN']=[accuracy_score(y_predict,y_testing),f1_score(y_predict,y_testi
```

```
Model Accuracy : 74.14%
Model F1-Score : 75.85%
Cross Val Accuracy: 74.85 %
Cross Val Standard Deviation: 0.79 %
              precision    recall  f1-score   support

           3       1.00      0.88      0.94       110
           4       0.92      0.68      0.78       131
           5       0.42      0.58      0.48        69
           6       0.40      0.56      0.47        70
           7       0.74      0.76      0.75        95
           8       0.97      0.89      0.93       105

    accuracy                           0.74       580
   macro avg       0.74      0.72      0.72       580
weighted avg       0.79      0.74      0.76       580
```

In [53]:

```
classifier=SVC(kernel = 'rbf', random_state = 1)
classifier.fit(X_training,y_training)
y_predict=classifier.predict(X_testing)
print(f"Model Accuracy : {accuracy_score(y_predict,y_testing)*100:.2f}%")
print(f"Model F1-Score : {f1_score(y_predict,y_testing,average='weighted')*100:.2f}%")
accuracies = cross_val_score(estimator = classifier, X = X_training, y = y_training, cv
print("Cross Val Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Cross Val Standard Deviation: {:.2f} %".format(accuracies.std()*100))
print(classification_report(y_predict,y_testing,zero_division=1))
model_comparison['Support Vector Classifier']=[accuracy_score(y_predict,y_testing),f1_s
```

```
Model Accuracy : 72.59%
Model F1-Score : 73.41%
Cross Val Accuracy: 73.86 %
Cross Val Standard Deviation: 1.56 %
              precision    recall  f1-score   support

           3       1.00      0.92      0.96       106
           4       0.82      0.70      0.76       114
           5       0.53      0.56      0.55        91
           6       0.41      0.47      0.44        86
           7       0.61      0.76      0.67        78
           8       0.98      0.90      0.94       105

    accuracy                           0.73       580
   macro avg       0.73      0.72      0.72       580
weighted avg       0.75      0.73      0.73       580
```

In [54]:

```
classifier=RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_stat
```

```
classifier.fit(X_training,y_training)
y_predict=classifier.predict(X_testing)
print(f"Model Accuracy : {accuracy_score(y_predict,y_testing)*100:.2f}%")
print(f"Model F1-Score : {f1_score(y_predict,y_testing,average='weighted')*100:.2f}%")
accuracies = cross_val_score(estimator = classifier, X = X_training, y = y_training, cv
print("Cross Val Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Cross Val Standard Deviation: {:.2f} %".format(accuracies.std()*100))
print(classification_report(y_predict,y_testing,zero_division=1))
model_comparison['Random Forest']=[accuracy_score(y_predict,y_testing),f1_score(y_predi
```

```
Model Accuracy : 80.34%
Model F1-Score : 80.86%
Cross Val Accuracy: 79.72 %
Cross Val Standard Deviation: 0.92 %
              precision    recall  f1-score   support

           3       1.00      0.93      0.97       104
           4       0.91      0.84      0.87       105
           5       0.65      0.65      0.65        96
           6       0.53      0.62      0.57        82
           7       0.77      0.82      0.79        92
           8       0.97      0.92      0.94       101

    accuracy                           0.80       580
   macro avg       0.80      0.80      0.80       580
weighted avg       0.82      0.80      0.81       580
```

In [55]:

```
Model_com_df=pd.DataFrame(model_comparison).T # to compare the models used above
Model_com_df.columns=['Model Accuracy','Model F1-Score','CV Accuracy','CV std']
Model_com_df=Model_com_df.sort_values(by='Model F1-Score',ascending=False)
Model_com_df.style.format("{:.2%}").background_gradient(cmap='Blues')
```

Out[55]:

	Model Accuracy	Model F1-Score	CV Accuracy	CV std
Random Forest	80.34%	80.86%	79.72%	0.92%
KNN	74.14%	75.85%	74.85%	0.79%
Decision Tree	74.31%	74.39%	74.55%	1.15%
Support Vector Classifier	72.59%	73.41%	73.86%	1.56%

In [ ]: