Wine Quality Classification

Dataset: Wine-quality-red

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix, classification_report,␣
↪accuracy_score
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
# Location of dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/
↪winequality-red.csv"
# Reading dataset to pandas dataframe
data = pd.read_csv(url, sep=";")
```

[3]:
```
data.head()
```

[3]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality
0      9.4        5
```

```
1      9.8        5
2      9.8        5
3      9.8        6
4      9.4        5
```

[4]:
```
data.describe()
```

[4]:
```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    1599.000000       1599.000000  1599.000000     1599.000000
mean        8.319637          0.527821     0.270976        2.538806
std         1.741096          0.179060     0.194801        1.409928
min         4.600000          0.120000     0.000000        0.900000
25%         7.100000          0.390000     0.090000        1.900000
50%         7.900000          0.520000     0.260000        2.200000
75%         9.200000          0.640000     0.420000        2.600000
max        15.900000          1.580000     1.000000       15.500000

         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  1599.000000          1599.000000           1599.000000  1599.000000
mean      0.087467            15.874922             46.467792     0.996747
std       0.047065            10.460157             32.895324     0.001887
min       0.012000             1.000000              6.000000     0.990070
25%       0.070000             7.000000             22.000000     0.995600
50%       0.079000            14.000000             38.000000     0.996750
75%       0.090000            21.000000             62.000000     0.997835
max       0.611000            72.000000            289.000000     1.003690

                pH    sulphates      alcohol      quality
count  1599.000000  1599.000000  1599.000000  1599.000000
mean      3.311113     0.658149    10.422983     5.636023
std       0.154386     0.169507     1.065668     0.807569
min       2.740000     0.330000     8.400000     3.000000
25%       3.210000     0.550000     9.500000     5.000000
50%       3.310000     0.620000    10.200000     6.000000
75%       3.400000     0.730000    11.100000     6.000000
max       4.010000     2.000000    14.900000     8.000000
```

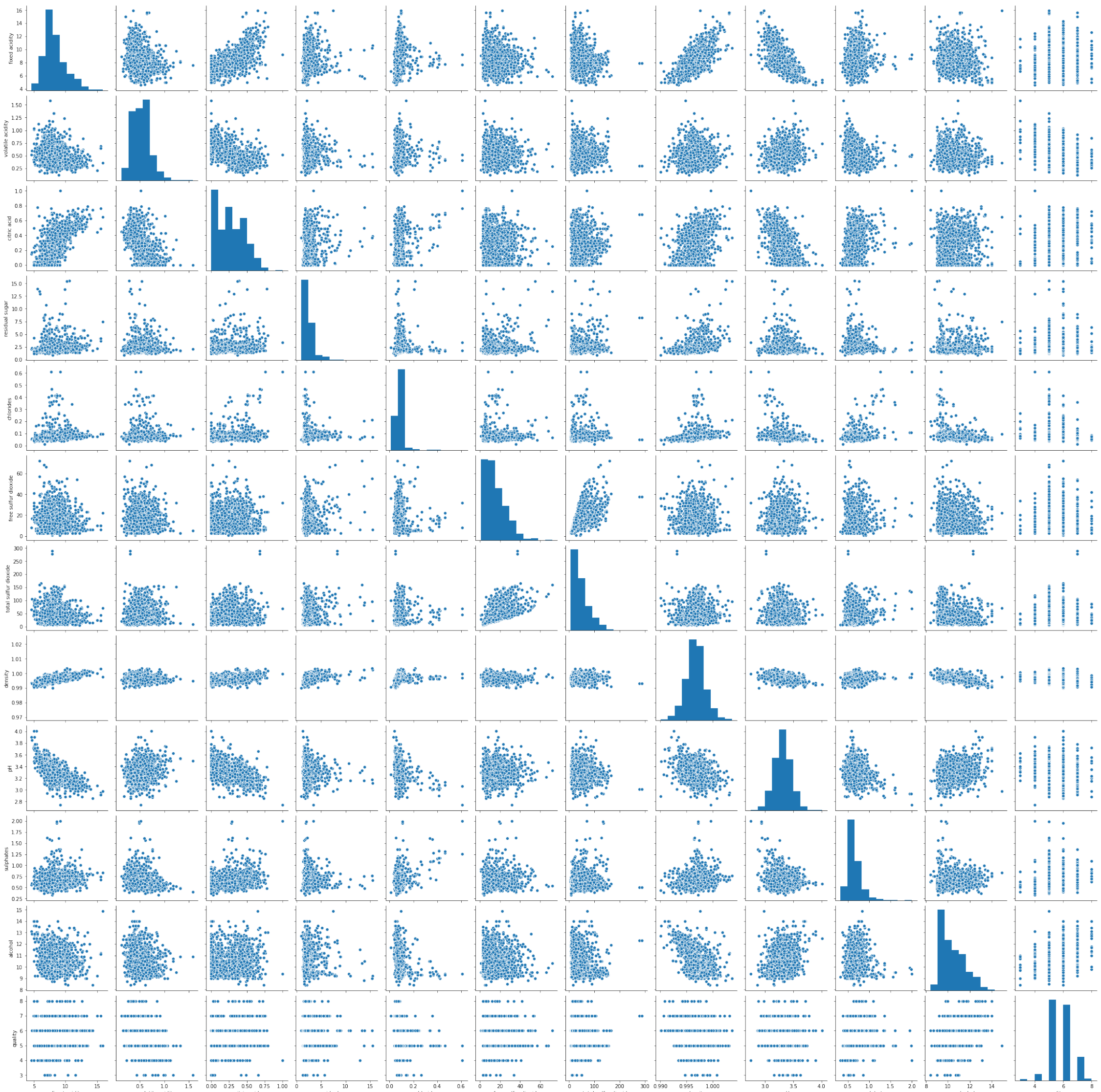
[5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
```

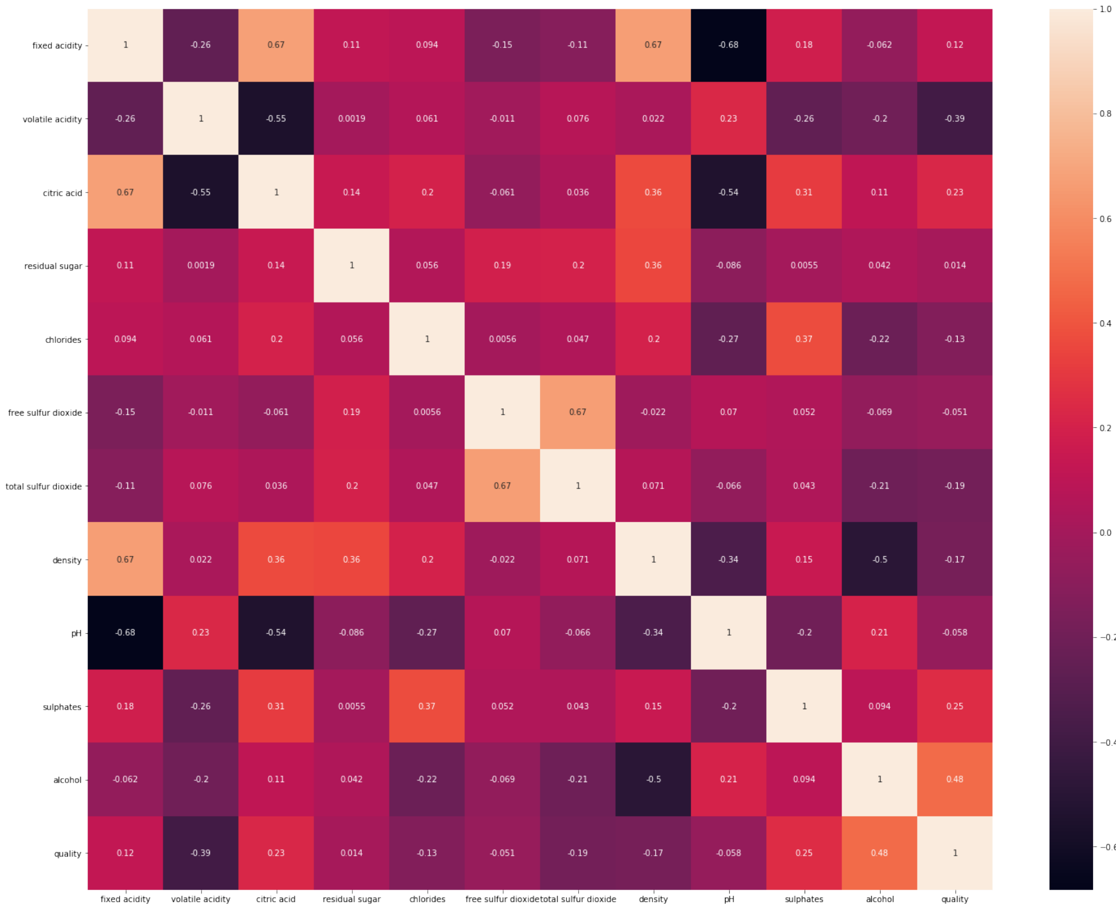
```
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

[6]:
```
# Visualizing the variables
plt.figure(figsize=(25,25))
sns.pairplot(data)
plt.show()
```

```
<Figure size 1800x1800 with 0 Axes>
```



[7]:
```python
# Correlation among variables
plt.figure(figsize=(25,20))
sns.heatmap(data.corr(),annot=True)
plt.show()
```



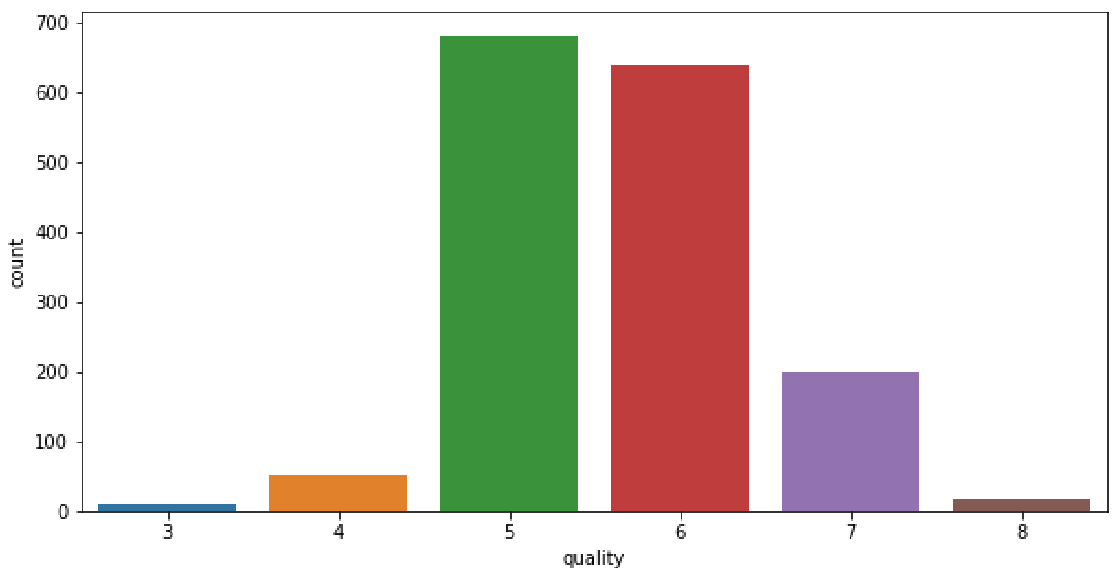
[8]:
```
# Checking if any null values are there in dataset
data.isnull().sum()
```

[8]:
```
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
dtype: int64
```

[9]:
```
# Checking the quality of wine
data['quality'].value_counts()
```

[9]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

[10]:
```python
# Plotting the quality of wine
plt.figure(figsize=(10,5))
sns.countplot(data['quality'])
plt.show()
```



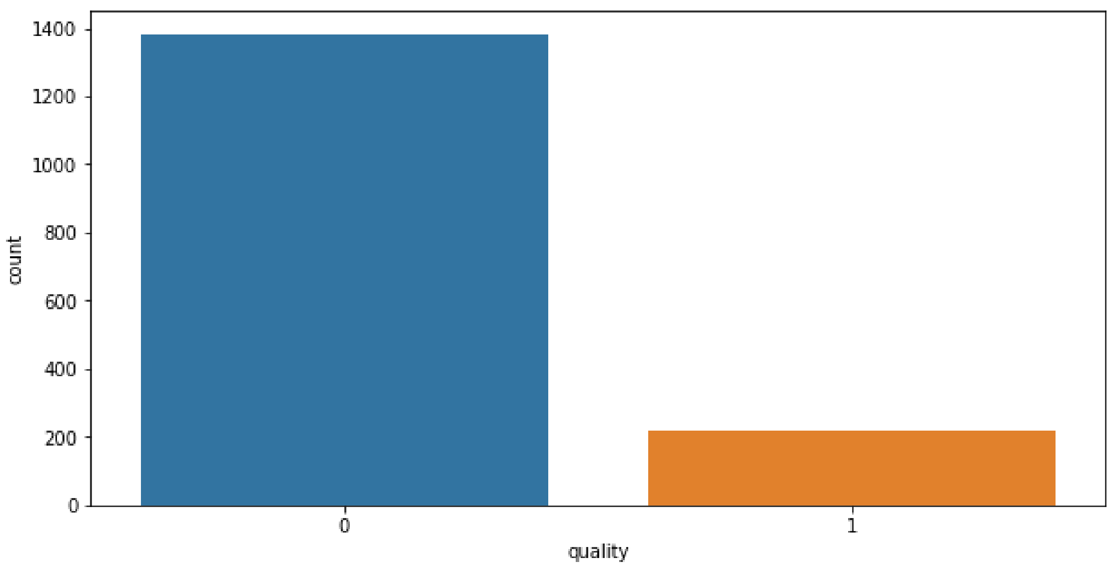
[11]:
```python
# Classifying wine quality as good and bad (anything below 7 as bad, and from 7␣
↪is good)
bins = (2, 6.99, 8)
group_names = ['bad', 'good']
data['quality'] = pd.cut(data['quality'], bins=bins, labels=group_names)
data['quality'].unique()
```

[11]:
```
[bad, good]
Categories (2, object): [bad < good]
```

[12]:
```python
# Labeling bad wine quality as 0 and good wine quality as 1
label_quality = LabelEncoder()
data['quality'] = label_quality.fit_transform(data['quality'])
data['quality'].value_counts()
```

[12]:
```
0    1382
1     217
Name: quality, dtype: int64
```

[13]:
```
# Plotting the target variables
plt.figure(figsize=(10,5))
sns.countplot(data['quality'])
plt.show()
```



[14]:
```
# Assign data from first 11 columns to X variable
X = data.drop('quality', axis=1)
# Assign data from last 12th column to y variable
y = data['quality']
```

[15]:
```
# Spliting the dataset into (3:1) train-test ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.75,␣
↪test_size = 0.25)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1199, 11)
(400, 11)
(1199,)
(400,)
```

[16]:
```
# Feature scaling (by standardization)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
print(X_train)
print(X_test)
```

```
[[-0.07563803 -0.97403017  0.3002543  … -0.25332521  0.965028
   0.55331102]
 [-0.88744108  0.43787425 -0.36651643 …  1.52214504 -0.46467959
  -1.04438534]
 [-0.30758176 -0.15946993  0.14638413 …  0.00970742 -0.58900199
  -0.01058181]
 …
 [-1.6412582  -0.21377394 -1.23844737 …  1.25911241  1.0271892
   3.37277519]
 [ 2.41775706 -1.35415828  1.37734547 … -1.30545573  4.38389397
  -0.57447464]
 [-0.30758176 -0.7568141   0.7618648  …  0.53577268  0.03261
   1.11720386]]
[[-1.06139888 -0.7568141  -0.21264626 …  0.86456347 -0.65116319
   1.58711455]
 [-0.42355362 -0.64820607  0.3002543  … -0.05605074 -0.46467959
   0.7412753 ]
 [-0.53952549  0.57363429 -0.87941698 …  1.78517767  0.2190936
   0.36534675]
 …
 [ 0.27227756 -0.64820607  1.89024602 …  1.52214504  0.03261
   0.08340033]
 [ 0.3302635  -0.26807796  1.32605541 … -0.64787416 -0.40251839
   1.58711455]
 [ 0.3302635  -0.15946993 -0.31522637 …  0.14122373  1.0893504
   0.64729316]]
```

[32]:
```
# Multilayer Perceptron Classifier
mlp = MLPClassifier(hidden_layer_sizes=(100,1000,100), max_iter=10000)
mlp.fit(X_train, y_train.values.ravel())
```

[32]:
```
MLPClassifier(hidden_layer_sizes=(100, 1000, 100), max_iter=10000)
```

[33]:
```
predictions = mlp.predict(X_test)
predictions
```

[33]:
```
array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 0])
```

[34]:
```
print(confusion_matrix(y_test,predictions))
```

```
[[325  23]
 [ 17  35]]
```

[35]:
```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.95      0.93      0.94       348
           1       0.60      0.67      0.64        52

    accuracy                           0.90       400
   macro avg       0.78      0.80      0.79       400
weighted avg       0.91      0.90      0.90       400
```

[36]:
```
print("Accuracy: " + str(accuracy_score(y_test, predictions)*100) + "%")
```

```
Accuracy: 90.0%
```