Aarjav Jain(21BIT0466)

```
#import necessary libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

#Load the Dataset

df=pd.read_csv("/content/winequality-red.csv")
df.head()

   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

df.shape
```

```
(1599, 12)
```

```
df.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
```

```
df.isnull().sum()
```

```
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
```

```
df.corr()
```

```
                      fixed acidity  volatile acidity  citric acid  \
fixed acidity              1.000000         -0.256131     0.671703   
volatile acidity          -0.256131          1.000000    -0.552496   
citric acid                0.671703         -0.552496     1.000000   
residual sugar             0.114777          0.001918     0.143577   
chlorides                  0.093705          0.061298     0.203823   
free sulfur dioxide       -0.153794         -0.010504    -0.060978   
total sulfur dioxide      -0.113181          0.076470     0.035533   
density                    0.668047          0.022026     0.364947   
```

```
pH                              -0.682978         0.234937    -0.541904
sulphates                        0.183006        -0.260987     0.312770
alcohol                         -0.061668        -0.202288     0.109903
quality                          0.124052        -0.390558     0.226373

                      residual sugar  chlorides  free sulfur
dioxide  \
fixed acidity               0.114777   0.093705             -0.153794

volatile acidity            0.001918   0.061298             -0.010504

citric acid                 0.143577   0.203823             -0.060978

residual sugar              1.000000   0.055610              0.187049

chlorides                   0.055610   1.000000              0.005562

free sulfur dioxide         0.187049   0.005562              1.000000

total sulfur dioxide        0.203028   0.047400              0.667666

density                     0.355283   0.200632             -0.021946

pH                         -0.085652  -0.265026              0.070377

sulphates                   0.005527   0.371260              0.051658

alcohol                     0.042075  -0.221141             -0.069408

quality                     0.013732  -0.128907             -0.050656


                      total sulfur dioxide   density        pH
sulphates  \
fixed acidity                    -0.113181  0.668047 -0.682978
0.183006
volatile acidity                  0.076470  0.022026  0.234937  -
0.260987
citric acid                       0.035533  0.364947 -0.541904
0.312770
residual sugar                    0.203028  0.355283 -0.085652
0.005527
chlorides                         0.047400  0.200632 -0.265026
0.371260
free sulfur dioxide               0.667666 -0.021946  0.070377
0.051658
total sulfur dioxide              1.000000  0.071269 -0.066495
0.042947
density                           0.071269  1.000000 -0.341699
0.148506
pH                               -0.066495 -0.341699  1.000000  -
```

```
0.196648
sulphates                              0.042947  0.148506 -0.196648
1.000000
alcohol                               -0.205654 -0.496180  0.205633
0.093595
quality                               -0.185100 -0.174919 -0.057731
0.251397

                       alcohol   quality
fixed acidity        -0.061668  0.124052
volatile acidity     -0.202288 -0.390558
citric acid           0.109903  0.226373
residual sugar        0.042075  0.013732
chlorides            -0.221141 -0.128907
free sulfur dioxide  -0.069408 -0.050656
total sulfur dioxide -0.205654 -0.185100
density              -0.496180 -0.174919
pH                    0.205633 -0.057731
sulphates             0.093595  0.251397
alcohol               1.000000  0.476166
quality               0.476166  1.000000
```

```
df.corr().quality.sort_values(ascending =False)
```

```
quality                 1.000000
alcohol                 0.476166
sulphates               0.251397
citric acid             0.226373
fixed acidity           0.124052
residual sugar          0.013732
free sulfur dioxide    -0.050656
pH                     -0.057731
chlorides              -0.128907
density                -0.174919
total sulfur dioxide   -0.185100
volatile acidity       -0.390558
Name: quality, dtype: float64
```

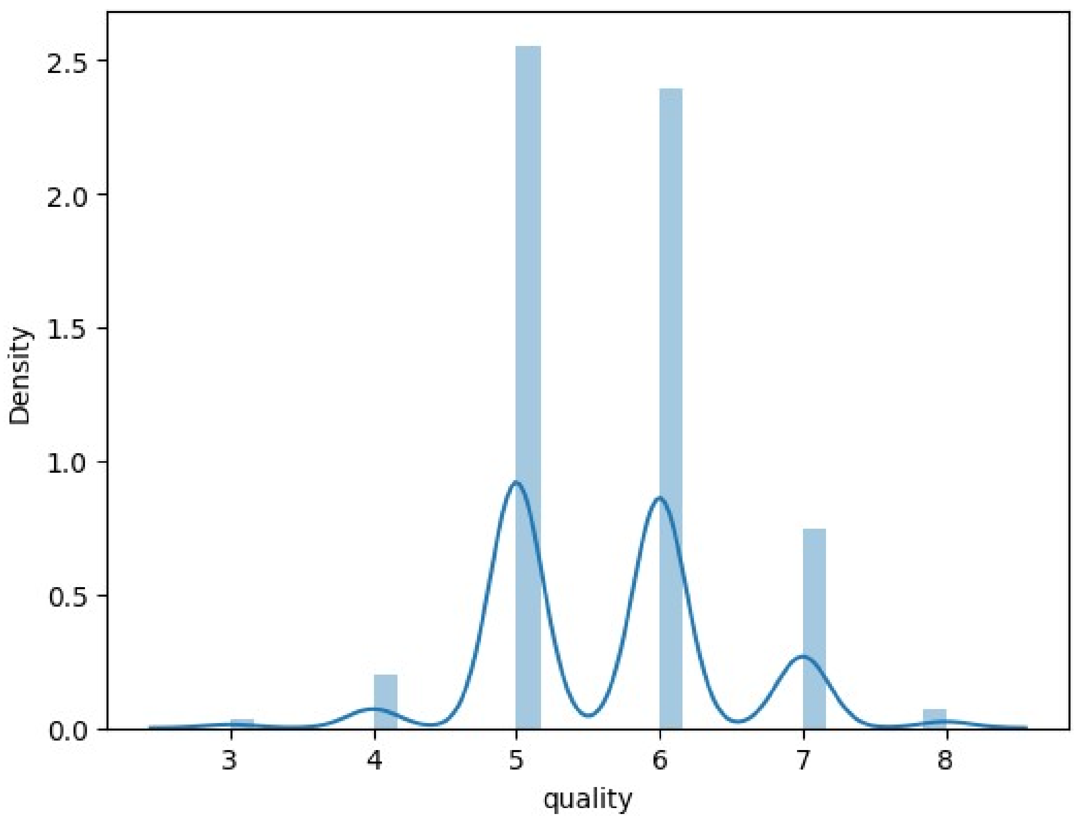
Visualisation

Univariate Analysis

```
sns.distplot(df.quality)
```

```
<ipython-input-13-e8684199aa87>:1: UserWarning:
```

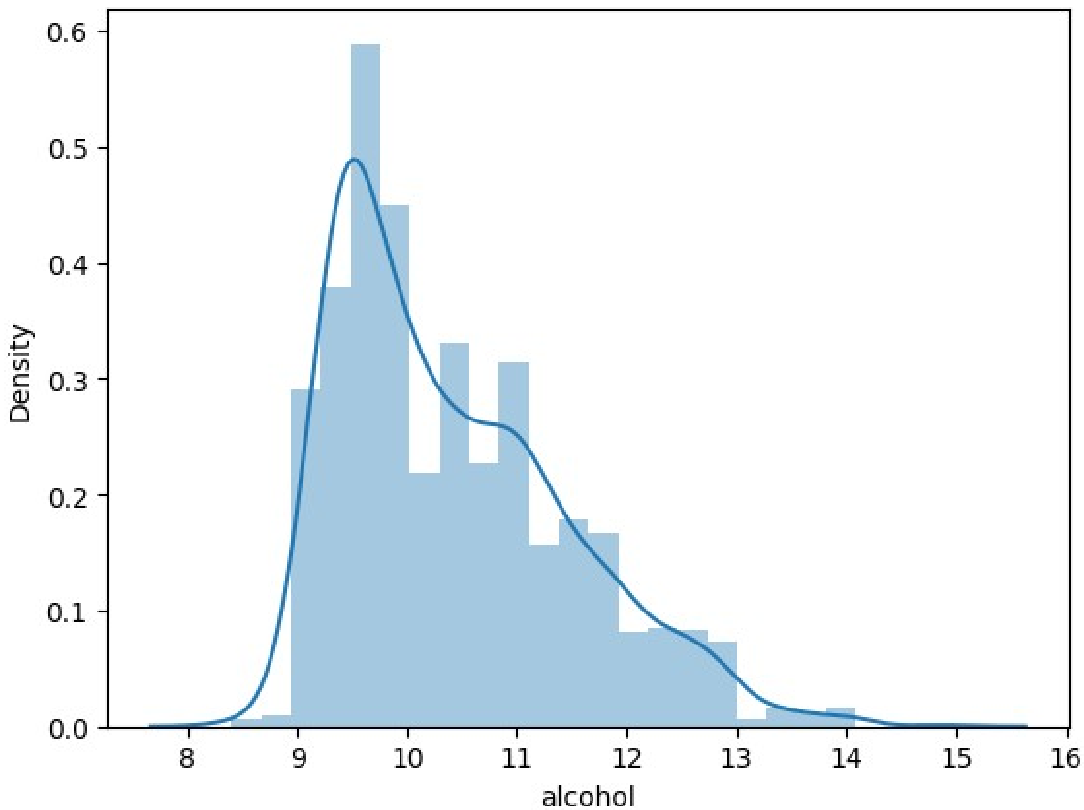
```
`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.quality)

<Axes: xlabel='quality', ylabel='Density'>
```




```
sns.distplot(df.alcohol)

<ipython-input-14-cc0e16fd78a8>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
```

```
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.alcohol)
```

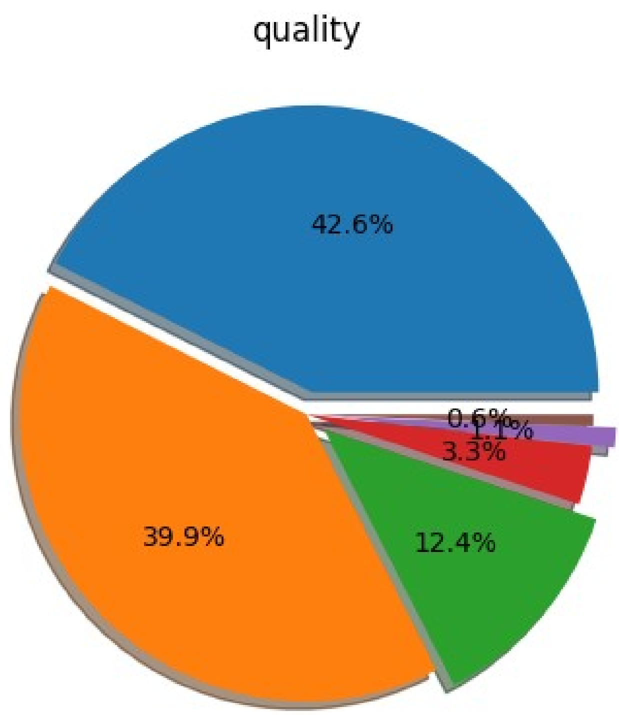
```
<Axes: xlabel='alcohol', ylabel='Density'>
```




```
df.quality.unique()
```

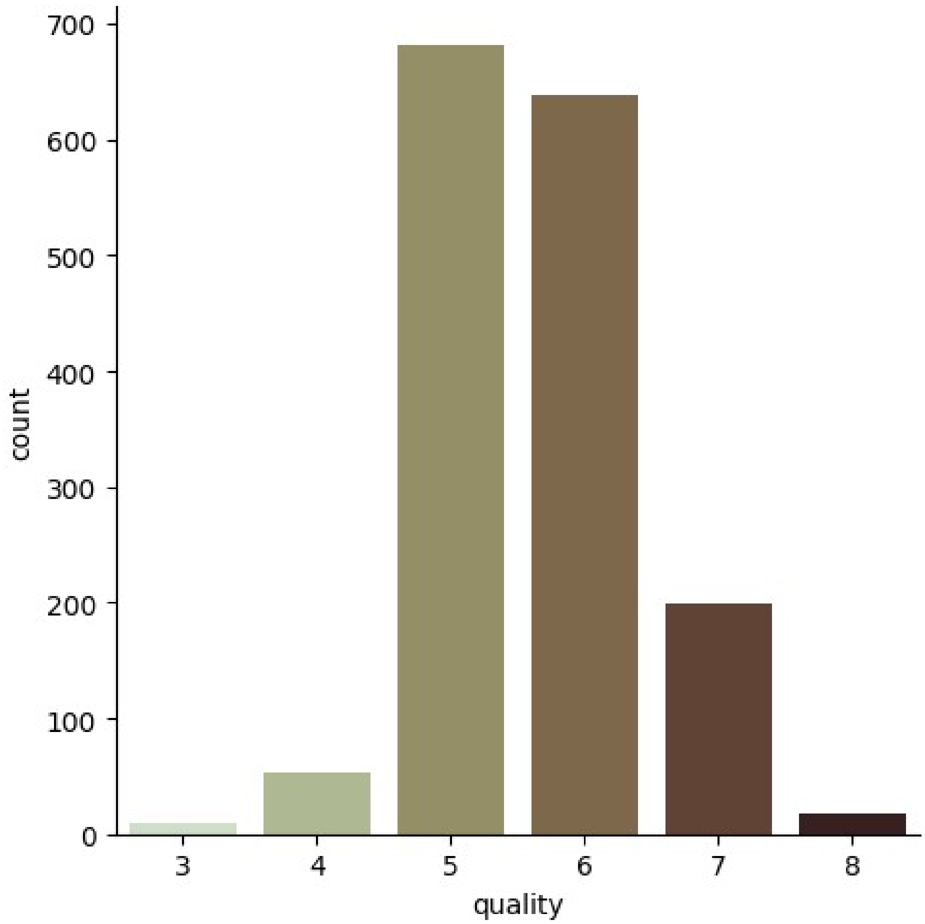
```
array([5, 6, 7, 4, 8, 3])
```

```
plt.pie(df.quality.value_counts(),
[0.08,0,0.08,0,0.08,0],autopct='%1.1f%%', shadow=True)
plt.title('quality')
plt.show()
```

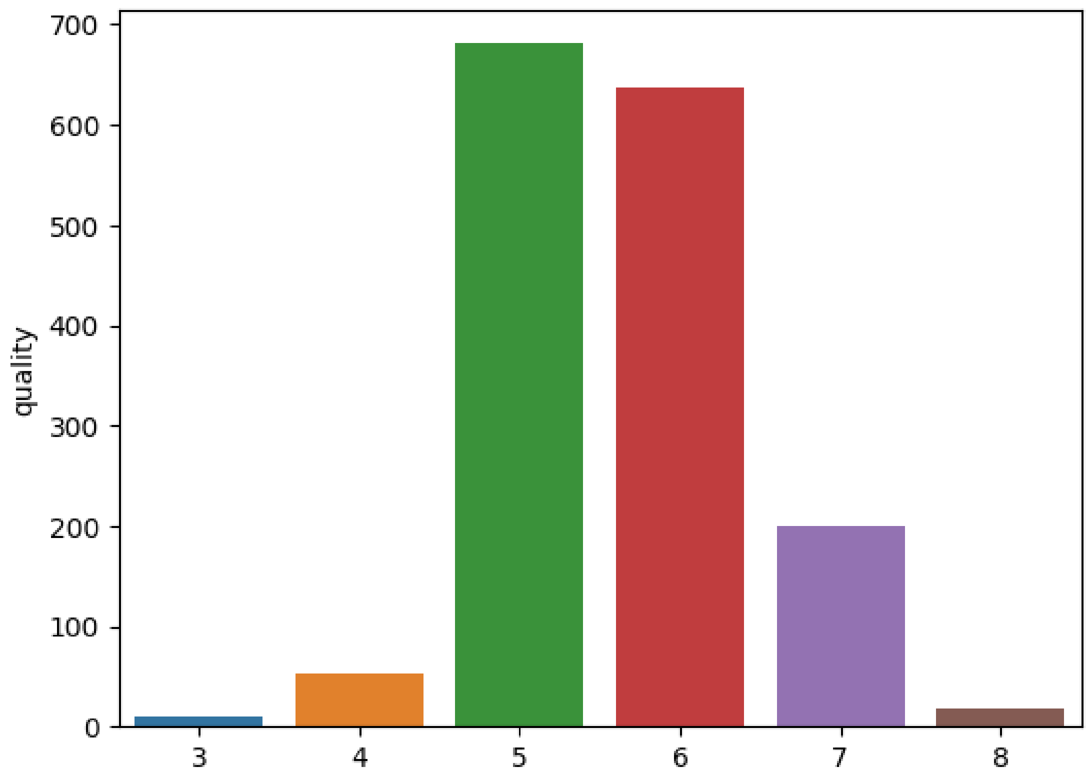



```
sns.catplot(data=df, x='quality',kind='count', palette="ch:.75")
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb6e1d5bfd0>
```



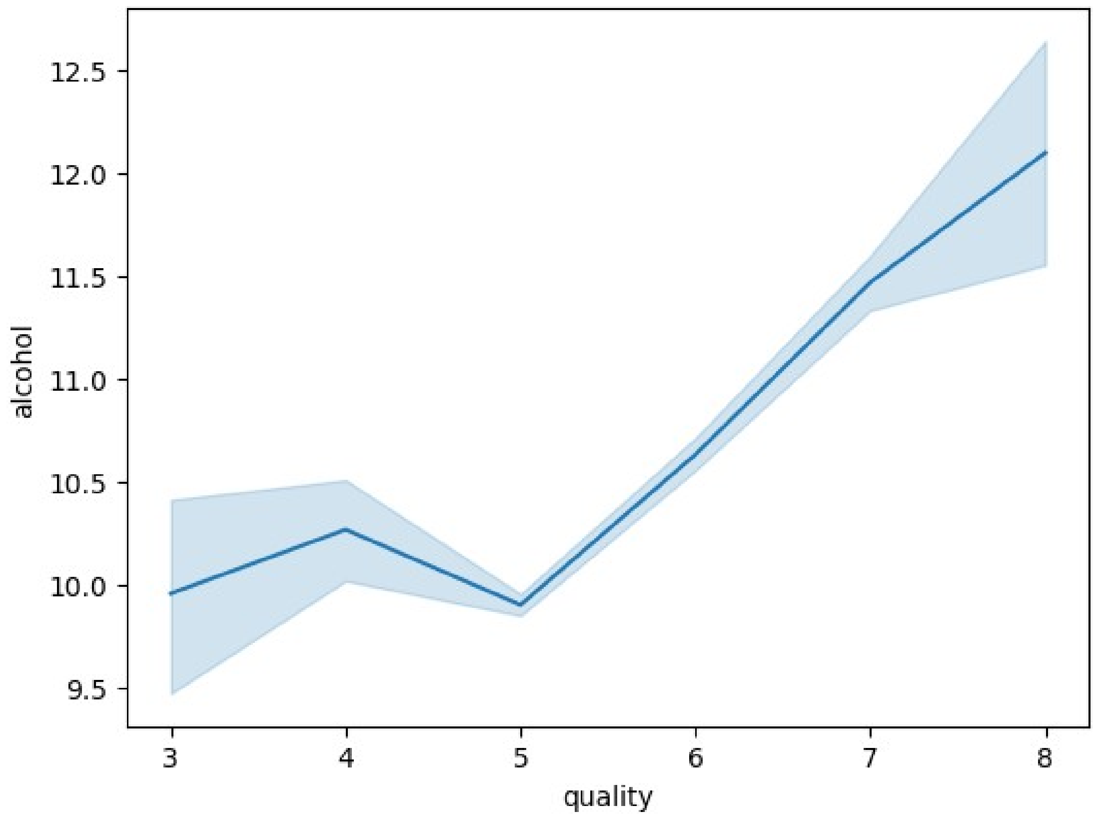

```
sns.barplot(x =df.quality.value_counts().index,y
=df.quality.value_counts() )

<Axes: ylabel='quality'>
```



Bivariate Analysis

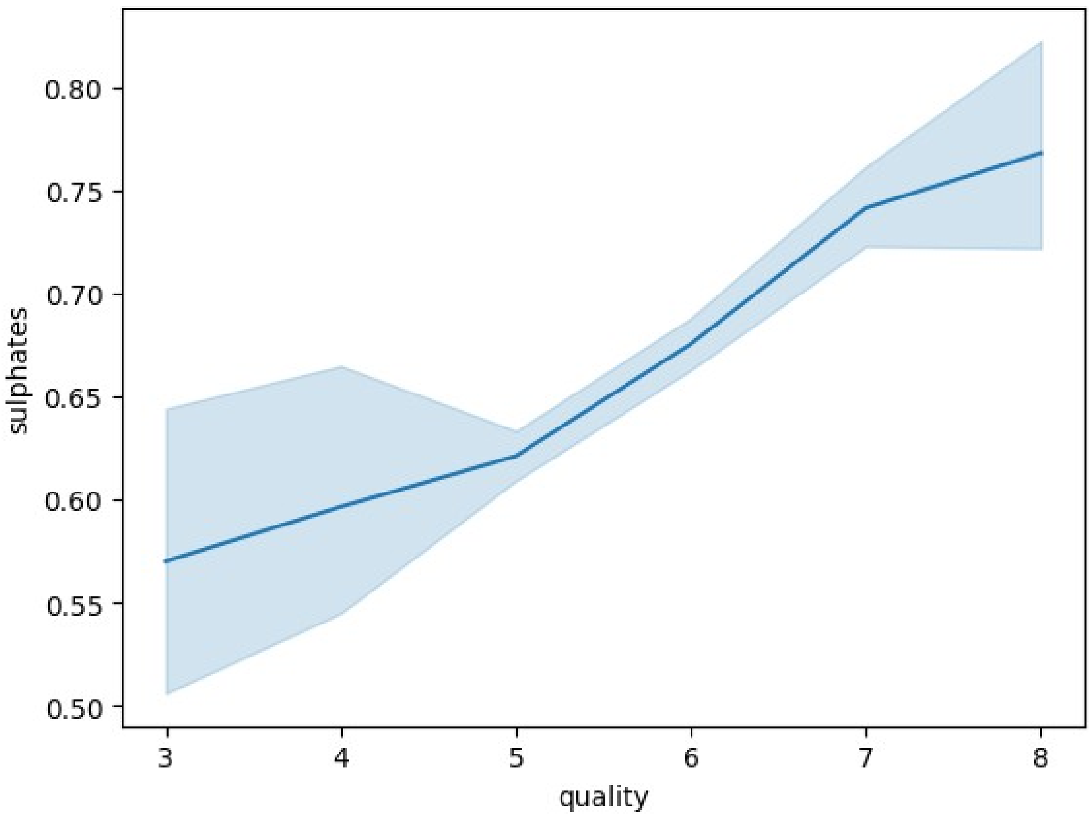
```
sns.lineplot(x=df.quality, y=df.alcohol)
```

```
<Axes: xlabel='quality', ylabel='alcohol'>
```




```
sns.lineplot(x=df.quality, y=df.sulphates)
```

```
<Axes: xlabel='quality', ylabel='sulphates'>
```



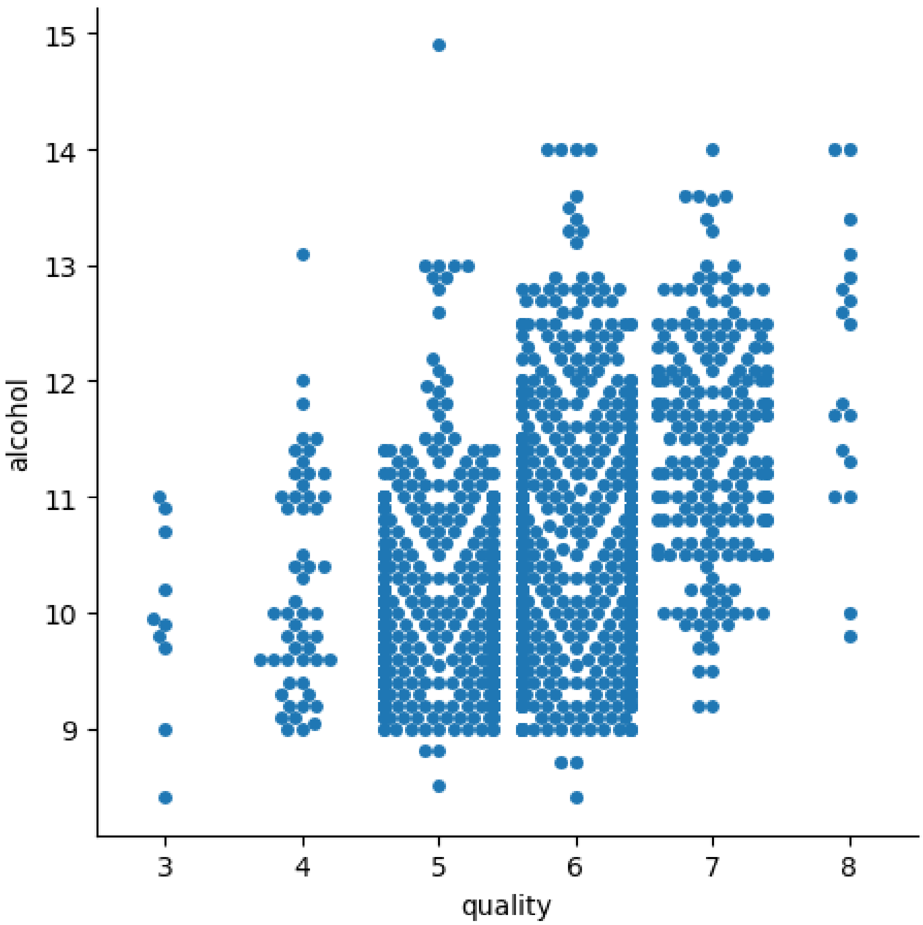

```
sns.catplot(data=df, x="quality", y="alcohol", kind="swarm")
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544: UserWarning: 66.2% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544: UserWarning: 49.1% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544: UserWarning: 7.5% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb6e1911600>
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544: UserWarning: 73.7% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544: UserWarning: 61.0% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544:
UserWarning: 16.6% of the points cannot be placed; you may want to
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```



Multivariate Analysis

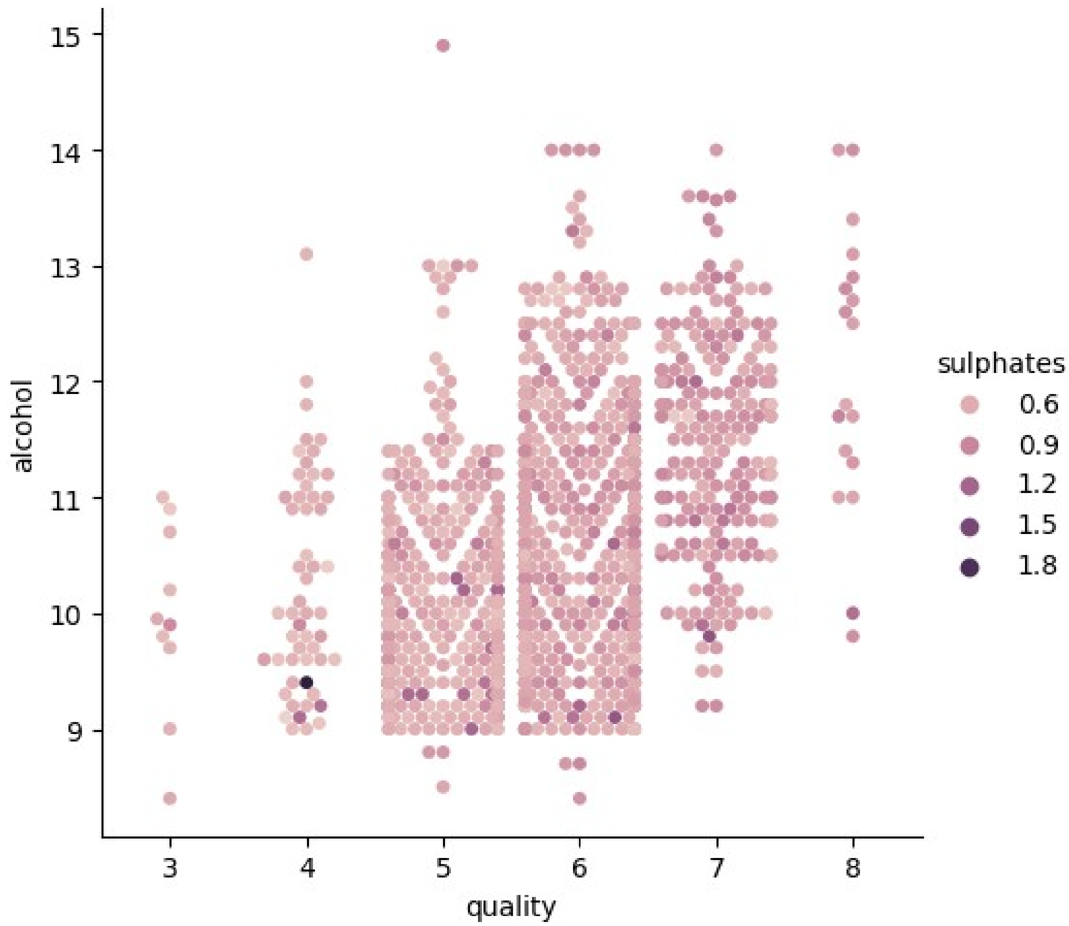
```
sns.catplot(data=df, x="quality", y="alcohol", hue="sulphates",
kind="swarm")
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544:
UserWarning: 66.2% of the points cannot be placed; you may want to
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544:
UserWarning: 49.1% of the points cannot be placed; you may want to
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544:
```

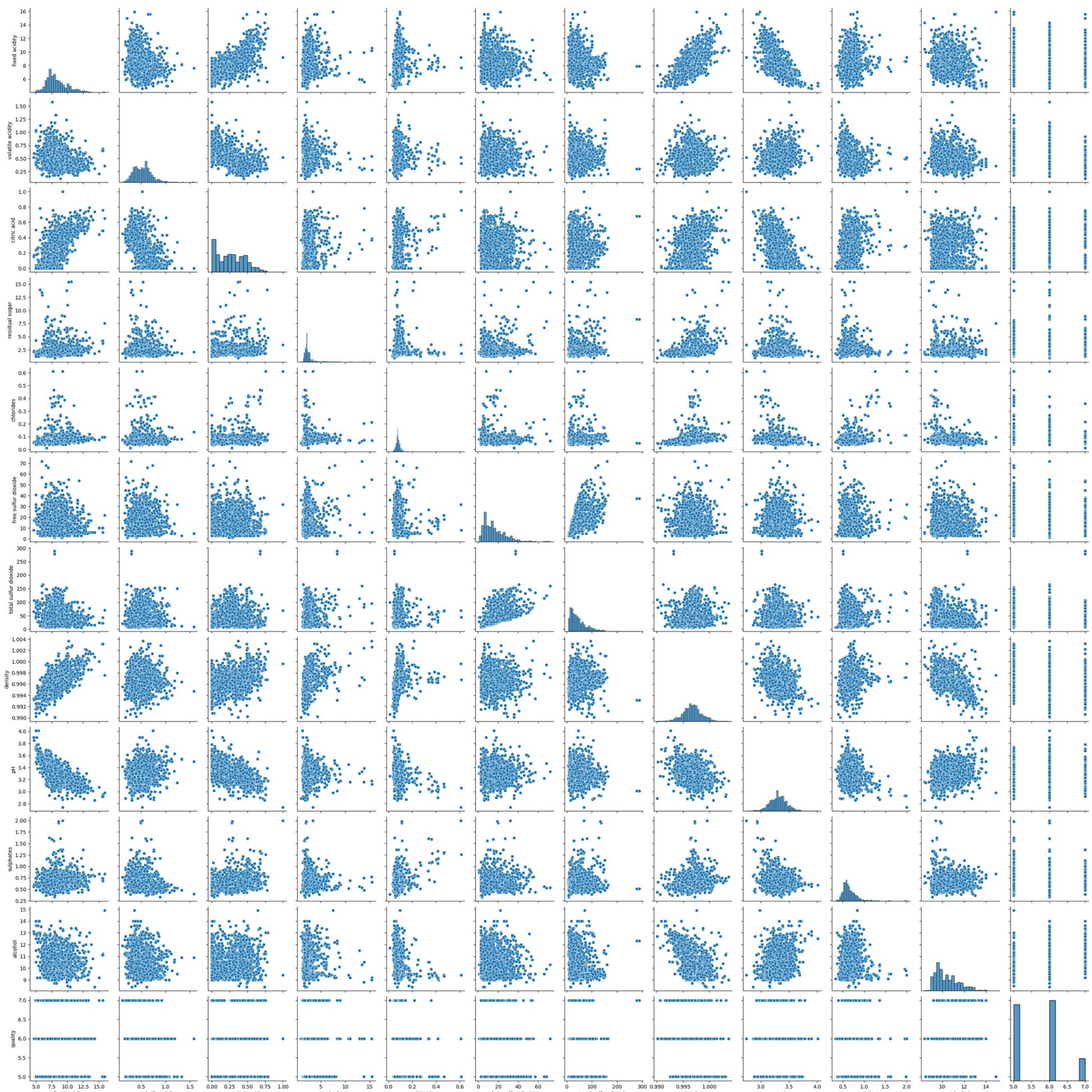
```
UserWarning: 7.5% of the points cannot be placed; you may want to
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544:
UserWarning: 73.7% of the points cannot be placed; you may want to
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544:
UserWarning: 61.0% of the points cannot be placed; you may want to
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544:
UserWarning: 16.6% of the points cannot be placed; you may want to
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544:
UserWarning: 67.8% of the points cannot be placed; you may want to
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544:
UserWarning: 52.5% of the points cannot be placed; you may want to
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544:
UserWarning: 8.0% of the points cannot be placed; you may want to
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb6dff5fa60>
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544:
UserWarning: 74.0% of the points cannot be placed; you may want to
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544:
UserWarning: 61.3% of the points cannot be placed; you may want to
decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

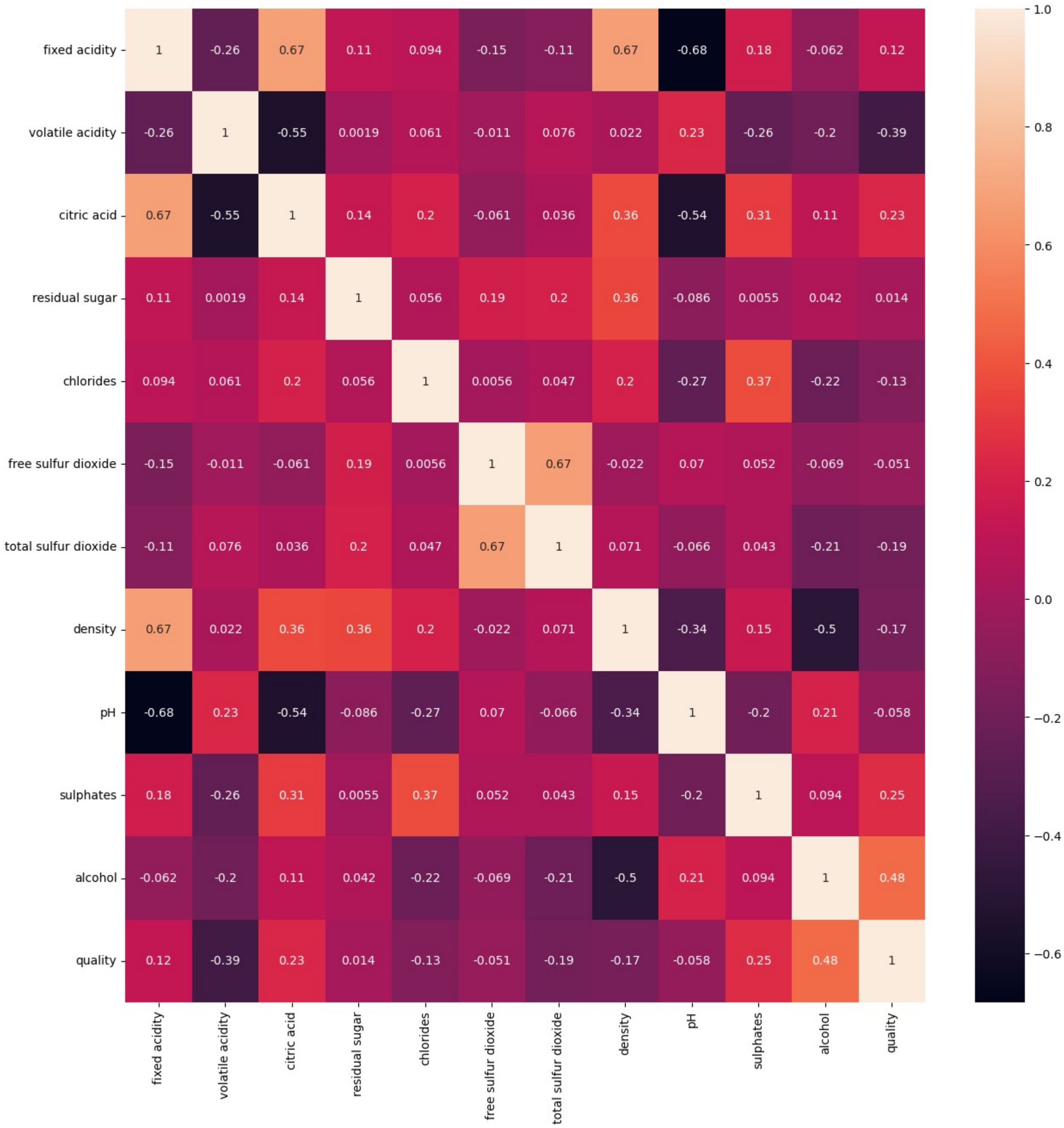



```
sns.pairplot(df)

<seaborn.axisgrid.PairGrid at 0x7efa83a890c0>
```




```
plt.figure(figsize=(15, 15))
sns.heatmap(df.corr(), annot=True)
```

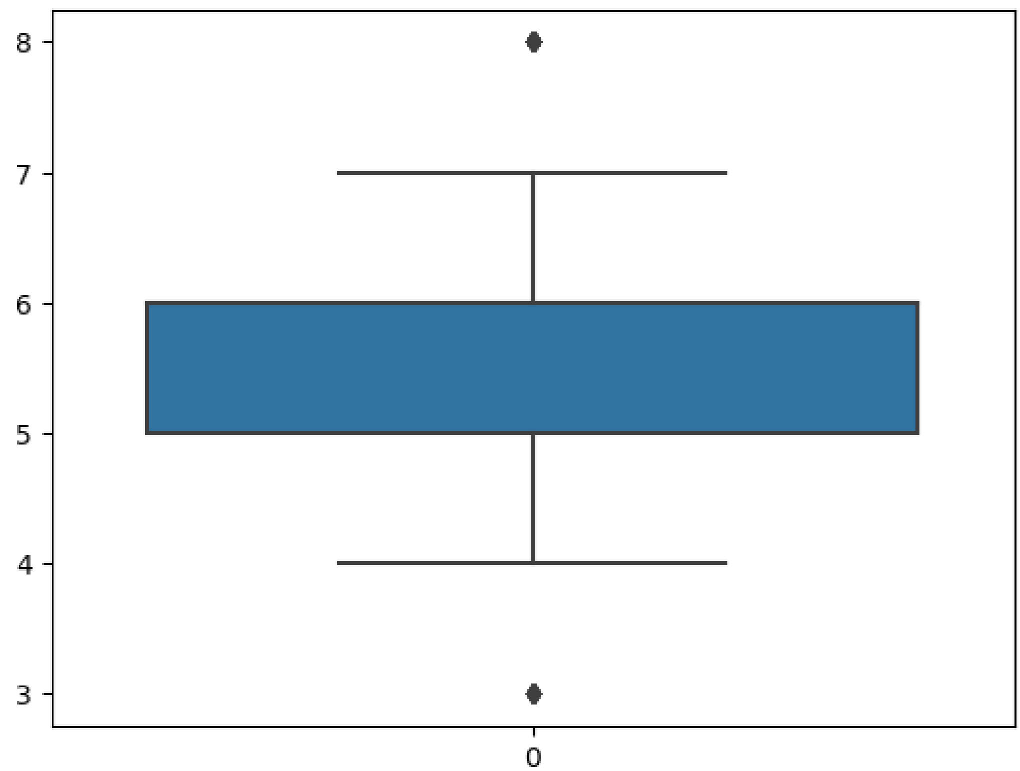
```
<Axes: >
```



Outliers Detection and Replacement

```
sns.boxplot(df.quality)
```

```
<Axes: >
```




```
q1 = df.quality.quantile(0.25) #Q1
q3 = df.quality.quantile(0.75) #Q3
print(q1)
print(q3)

5.0
6.0

IQR =q3-q1
upper_limit = q3+1.5*IQR
upper_limit

7.5

lower_limit = q3-1.5*IQR
lower_limit

4.5

df.median()

fixed acidity            7.90000
volatile acidity         0.52000
citric acid              0.26000
residual sugar           2.20000
chlorides                0.07900
free sulfur dioxide     14.00000
```

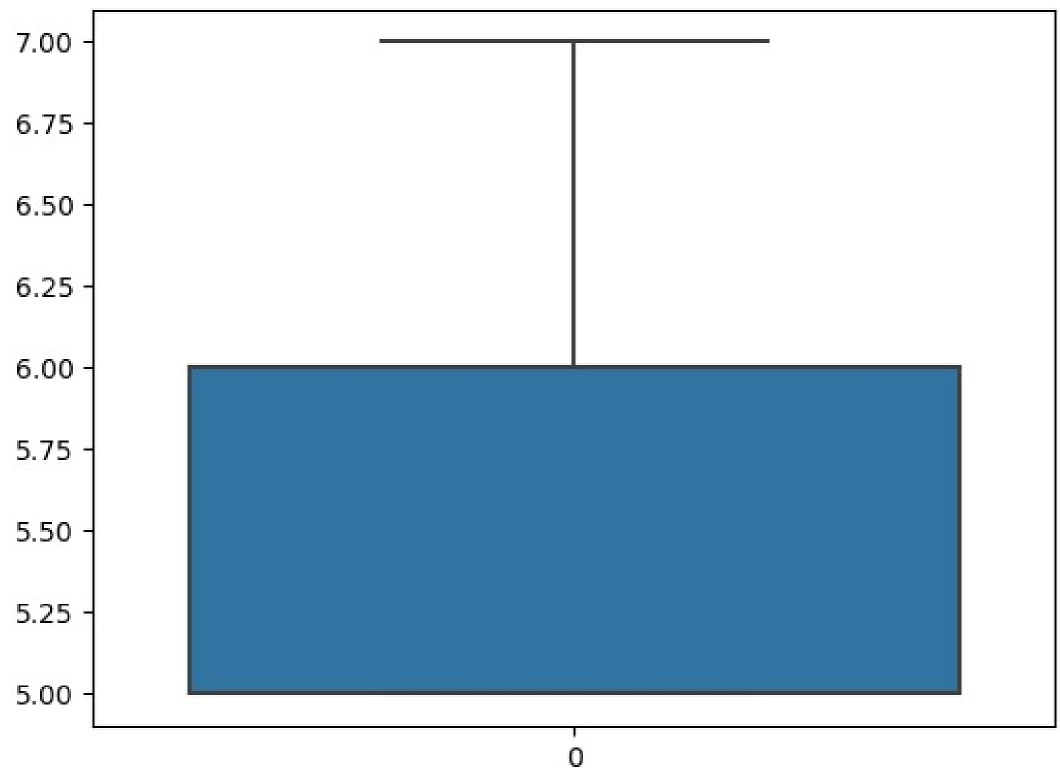
```
total sulfur dioxide    38.00000
density                  0.99675
pH                       3.31000
sulphates                0.62000
alcohol                 10.20000
quality                  6.00000
dtype: float64
```

```
df.quality = np.where(df['quality']>upper_limit,6,df['quality'])
```

```
df.quality = np.where(df['quality']<lower_limit,6,df['quality'])
```

```
sns.boxplot(df.quality)
```

```
<Axes: >
```



Independent(X) and dependent(Y) variable split

```
y = df.quality
```

```
y.head()
```

```
0    5
1    5
2    5
3    6
4    5
Name: quality, dtype: int64
```

```
X = df.drop(columns =['quality'],axis =1)
X.head()
```

```
   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  
0      9.4  
1      9.8  
2      9.8  
3      9.8  
4      9.4  
```

Scaling

```
from sklearn.preprocessing import MinMaxScaler
scale= MinMaxScaler()
```

```
x_scaled=pd.DataFrame(scale.fit_transform(X),columns=X.columns)
x_scaled.head()
```

```
   fixed acidity  volatile acidity  citric acid  residual sugar
chlorides  \
0       0.247788          0.397260         0.00        0.068493
0.106845
1       0.283186          0.520548         0.00        0.116438
0.143573
2       0.283186          0.438356         0.04        0.095890
0.133556
3       0.584071          0.109589         0.56        0.068493
0.105175
4       0.247788          0.397260         0.00        0.068493
0.106845

   free sulfur dioxide  total sulfur dioxide   density        pH
sulphates  \
0             0.140845              0.098940  0.567548  0.606299
0.137725
1             0.338028              0.215548  0.494126  0.362205
0.209581
2             0.197183              0.169611  0.508811  0.409449
0.191617
3             0.225352              0.190813  0.582232  0.330709
0.149701
4             0.140845              0.098940  0.567548  0.606299
0.137725

    alcohol
0  0.153846
1  0.215385
2  0.215385
3  0.215385
4  0.153846
```

Train test split

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x_scaled,y,test_size
=0.3,random_state=0)

x_train.shape
```

```
(1119, 11)
```

```
x_train.head()
```

```
      fixed acidity  volatile acidity  citric acid  residual sugar  
chlorides  \
92         0.353982          0.253425         0.29        0.075342   
0.163606   
1017       0.300885          0.041096         0.37        0.000000   
0.061770   
1447       0.194690          0.376712         0.00        0.068493   
0.113523   
838        0.486726          0.130137         0.35        0.047945   
0.105175   
40         0.238938          0.226027         0.36        0.342466   
0.103506   

      free sulfur dioxide  total sulfur dioxide   density        pH  \
92               0.253521              0.448763  0.523495  0.149606   
1017             0.492958              0.363958  0.000000  0.118110   
1447             0.295775              0.116608  0.509545  0.519685   
838              0.112676              0.077739  0.488253  0.393701   
40               0.154930              0.286219  0.567548  0.464567   

      sulphates   alcohol  
92     0.988024  0.215385  
1017   0.065868  0.661538  
1447   0.245509  0.200000  
838    0.299401  0.430769  
40     0.299401  0.323077  
```

```
y_train.shape
```

```
(1119,)
```

```
y_train.head()
```

```
92      5
1017    6
1447    5
838     7
40      5
Name: quality, dtype: int64
```

Linear Regression Model Building

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(x_train,y_train) # fitting the model on the training data
```

```
LinearRegression()
```

```
y_pred =model.predict(x_test)
y_pred
```

```
array([5.81412436, 5.3495239 , 6.47775328, 5.48902545, 6.11370699,
       5.07350715, 5.57863931, 6.1060134 , 5.31321748, 5.10476097,
       5.43505498, 5.47643313, 5.76714443, 5.52903534, 5.64110872,
       6.24477685, 6.68839727, 5.70190757, 6.09407664, 5.33726522,
       6.37596046, 5.4217376 , 5.70738721, 6.06680998, 5.50083739,
       5.09234752, 5.22832156, 6.3108958 , 5.27582959, 6.41911796,
       5.92163708, 5.73538743, 5.72412849, 5.48324871, 5.87893531,
       6.04997097, 5.2983249 , 5.70476589, 6.36741188, 5.85172931,
       5.45500004, 6.06147147, 6.62054449, 6.6424488 , 5.92869669,
       5.0225266 , 5.62234176, 5.88992795, 5.67706288, 6.01714988,
       5.42971047, 5.24446315, 5.82749025, 6.26519331, 5.74498206,
       5.37061533, 5.09990523, 5.46561722, 6.45167399, 5.61594653,
       5.43819663, 5.73244644, 5.95473134, 6.14516999, 5.29004245,
       6.10871319, 5.53493477, 6.06453878, 6.20449923, 5.91255367,
       5.49719885, 5.47454141, 5.29643145, 5.56678125, 5.62012216,
       6.28441938, 5.50366186, 5.33074174, 5.70532336, 6.19008027,
       5.69829824, 5.33893321, 5.96056082, 6.01446632, 5.62162274,
       5.5829836 , 6.17682194, 5.3077459 , 5.4465042 , 5.23474367,
       5.16216383, 5.40466331, 5.96010871, 5.54433236, 6.1716567 ,
       5.49719885, 6.14610874, 5.55430126, 5.53394743, 5.94967384,
       6.76598497, 5.7975484 , 5.79876595, 6.07543446, 5.62533771,
       5.30494217, 5.62533771, 5.53567141, 6.06331346, 5.44626938,
       6.07031515, 5.32659073, 6.10180947, 5.2395352 , 5.76777039,
       5.8257891 , 5.74126695, 6.41028152, 6.08162423, 5.70738721,
       5.19184908, 5.85955509, 5.61243318, 5.2063629 , 6.08197422,
       6.30964508, 5.31349219, 5.37798026, 5.98375761, 6.36186641,
       5.39762918, 5.43091915, 6.0447371 , 5.9312238 , 6.31802106,
       5.28127697, 5.92461914, 5.43157277, 6.00174549, 5.41931859,
       6.16063295, 5.20249173, 5.62234176, 5.50158183, 5.73267647,
       5.93079104, 6.10180947, 6.40815125, 6.20010271, 5.37742516,
       5.84158125, 5.87339177, 5.53567141, 5.68719671, 5.76556519,
       5.55618151, 5.31633291, 5.46669341, 6.18063644, 6.17201785,
       5.96297734, 5.66309954, 6.24768194, 5.23562768, 6.03593169,
       5.30147743, 6.04614315, 6.39382198, 5.46849213, 6.00881373,
       5.9170964 , 5.58162001, 5.21088658, 6.46528991, 6.09502666,
       6.04095355, 6.15421749, 6.71484912, 6.25471501, 4.9758616 ,
       5.16061628, 6.76857012, 5.56978607, 6.15697849, 6.56051929,
       5.14296265, 5.59212011, 5.79876595, 5.04454755, 5.96023258,
       6.07725721, 5.81332905, 5.28735194, 5.52605617, 5.40807798,
       5.22580767, 5.43505498, 5.48008614, 5.3466483 , 5.17663914,
       5.1997637 , 6.20199184, 5.23533643, 5.5294994 , 5.53021788,
       5.18909408, 5.25507849, 6.30184611, 5.97579132, 5.41788159,
       5.94622537, 5.82781115, 5.56622327, 6.28924008, 5.28780449,
       5.56386042, 5.9390689 , 5.92511454, 5.67057769, 5.03283135,
       5.29167648, 5.92296144, 6.3024462 , 5.74132814, 5.53534732,
       5.95635964, 6.30433344, 5.8758209 , 5.18333005, 5.60875483,
       5.21625908, 5.99254659, 5.44601651, 5.9619835 , 6.1294308 ,
```

```
       5.72898092, 6.13698513, 6.46432177, 6.40902954, 6.15541632,
       5.49719547, 5.2630472 , 5.23114681, 5.33021939, 5.90499735,
       4.96933685, 5.59161118, 5.41607987, 5.29025337, 6.23602078,
       5.00050763, 5.46712602, 5.12116089, 5.47598104, 5.609658  ,
       5.61272925, 5.10142296, 5.34482248, 5.24413651, 5.74238891,
       5.52156688, 6.76443382, 5.27605682, 5.70661931, 5.43620723,
       5.07770638, 5.60345169, 5.15940215, 5.75407039, 6.24785819,
       5.72329949, 5.70131181, 5.60028051, 6.14755725, 6.05820374,
       5.92568287, 6.2267951 , 5.20243973, 6.4074623 , 6.23148812,
       5.40405593, 6.63333168, 5.70131181, 6.30118643, 5.70380944,
       5.30966644, 5.54670896, 5.45696272, 6.22416388, 5.83259793,
       5.70931116, 5.77444104, 6.01109824, 5.49076131, 5.34622457,
       5.80767969, 5.56458515, 5.31370338, 5.37358383, 5.16358941,
       5.82414196, 5.49303048, 5.4214723 , 5.60968123, 6.06923179,
       5.94724506, 5.56238274, 5.37515823, 5.58556082, 5.5278609 ,
       5.74737261, 5.88715673, 5.53894745, 5.55430126, 5.28351979,
       6.40291845, 6.06006101, 6.22842679, 5.40317045, 6.25358863,
       5.5278609 , 5.8226611 , 6.16506917, 6.10447762, 5.62097458,
       5.57091508, 6.48706033, 5.57847748, 5.23373271, 5.5790941 ,
       6.58346407, 5.36932188, 5.43900552, 5.96023258, 5.23732323,
       5.87207377, 5.94079883, 5.55312321, 5.4618405 , 5.53631614,
       5.18951783, 5.52028526, 5.53493477, 6.43794683, 5.80767969,
       5.49493554, 5.54214224, 5.31362325, 5.54536579, 6.41486175,
       5.66889413, 6.05314651, 5.48793503, 5.89945033, 6.5193756 ,
       5.34235388, 6.07725721, 6.33763533, 5.70946507, 6.1138922 ,
       5.28215636, 6.23602078, 5.3091233 , 6.07831446, 5.67592927,
       5.3880802 , 5.70476589, 6.34607821, 5.20307939, 5.57944911,
       5.61896678, 5.73072167, 5.50550819, 6.09529138, 5.54392804,
       5.43721133, 5.87339177, 6.24189688, 5.72898092, 5.92184484,
       6.15240535, 5.88069523, 5.33903921, 5.51510032, 5.5433225 ,
       5.7202405 , 6.23840213, 5.56229642, 5.96889356, 5.66396589,
       6.53766792, 5.51651784, 5.23336446, 6.32292427, 5.78914554,
       5.89225775, 5.60764465, 5.74737261, 5.88498313, 5.44290115,
       5.48591873, 5.44290115, 5.34221171, 5.52037344, 5.76373184,
       5.46561722, 5.94792913, 5.96784034, 5.43542817, 5.94098034,
       5.56386042, 5.21834304, 4.9287341 , 6.00344755, 5.66502337,
       5.26020903, 6.13182802, 5.22431444, 5.46263836, 6.23105214,
       6.05476993, 5.73629077, 6.20228657, 5.78191123, 5.31569298,
       5.82725785, 5.83235242, 6.21927397, 5.40766916, 5.41607987,
       5.48818828, 5.96889356, 5.40466331, 6.21390118, 5.83711374,
       6.61799195, 5.81609807, 6.33302354, 5.59595683, 5.58033979,
       5.81919981, 5.50979607, 6.15541632, 5.39576439, 5.88684538,
       5.64140743, 5.55312321, 6.81305086, 5.74534328, 5.63070355,
       6.02446658, 5.21104497, 6.10937418, 6.3541574 , 6.20539917,
       5.29921766, 5.79824595, 5.30970779, 6.01109824, 5.50201433,
       5.45777872, 6.58105116, 5.41931859, 5.18475125, 5.20412579,
       5.07350715, 5.99467291, 5.32056192, 5.61910841, 5.20542908,
       5.88702858, 5.51689136, 5.48238114, 5.482462  , 5.83259793,
       5.55740713, 5.05044581, 5.20890116, 5.7202405 , 5.30970779])
```

```
## Compare

qty = pd.DataFrame({'Actual quality':y_test,'Predictd
quality':y_pred})
qty

      Actual quality  Predictd quality
1109               6          5.814124
1032               5          5.349524
1002               7          6.477753
487                6          5.489025
979                5          6.113707
...              ...               ...
801                5          5.557407
61                 5          5.050446
431                5          5.208901
1210               6          5.720240
713                5          5.309708

[480 rows x 2 columns]
```

Evaluate the Linear Regression Model

```
from sklearn import metrics

# R- Square
# evaluating testing accuracy
print(metrics.r2_score(y_test,y_pred))

0.31453512543285256

# MSE (Mean square Error)

print(metrics.mean_squared_error(y_test,y_pred))

0.2822514675132806

# RMSE (Root Mean Square Error)
print(np.sqrt(metrics.mean_squared_error(y_test,y_pred)))

0.5312734394954077
```

So, Linear Regression is not a good model

Logistic Regression Model

```
from sklearn.linear_model import LogisticRegression
model1 = LogisticRegression()
```

```
model1.fit(x_train,y_train)
pred = model.predict(x_test)
pred
```

```
array([6, 5, 7, 5, 6, 5, 5, 7, 5, 5, 5, 5, 5, 6, 6, 7, 7, 5, 5, 5, 6,
6,
       5, 7, 6, 5, 5, 7, 5, 6, 6, 6, 6, 5, 6, 7, 5, 6, 7, 6, 5, 6, 7,
6,
       6, 5, 5, 7, 6, 7, 6, 5, 6, 6, 6, 5, 5, 5, 7, 6, 5, 6, 6, 7, 5,
6,
       5, 6, 7, 5, 5, 6, 5, 5, 6, 6, 6, 5, 5, 6, 6, 5, 6, 7, 5, 5, 7,
5,
       6, 5, 5, 5, 6, 5, 7, 5, 7, 5, 6, 6, 7, 7, 6, 7, 5, 5, 5, 6, 6,
5,
       6, 5, 7, 5, 6, 6, 6, 7, 6, 5, 5, 6, 6, 5, 6, 7, 5, 5, 6, 6, 5,
5,
       6, 5, 6, 5, 6, 5, 6, 5, 6, 5, 5, 5, 5, 6, 7, 7, 6, 6, 7, 7, 6,
5,
       5, 6, 6, 5, 6, 6, 5, 6, 7, 6, 5, 5, 6, 7, 6, 6, 5, 6, 5, 7, 6,
7,
       7, 7, 6, 5, 5, 7, 6, 6, 7, 5, 6, 6, 5, 7, 6, 6, 6, 6, 6, 5, 5,
5,
       5, 5, 5, 6, 5, 5, 5, 5, 5, 7, 6, 6, 7, 7, 6, 6, 5, 6, 6, 7, 6,
5,
       5, 6, 7, 6, 6, 6, 7, 6, 5, 6, 5, 6, 5, 6, 7, 6, 6, 7, 7, 6, 5,
5,
       5, 5, 6, 5, 6, 5, 5, 7, 5, 5, 5, 5, 6, 6, 5, 5, 5, 5, 5, 7, 6,
6,
       5, 5, 5, 5, 6, 7, 5, 6, 5, 5, 6, 6, 7, 5, 7, 6, 6, 7, 6, 6, 6,
5,
       5, 6, 7, 6, 6, 6, 6, 6, 5, 6, 5, 6, 5, 5, 6, 6, 6, 6, 7, 6, 6,
5,
       5, 6, 7, 6, 6, 5, 5, 7, 6, 7, 5, 7, 6, 5, 7, 6, 6, 6, 7, 5, 5,
5,
       7, 5, 5, 7, 5, 6, 6, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 7, 6,
6,
       6, 5, 7, 5, 6, 6, 6, 6, 5, 7, 5, 6, 6, 5, 6, 7, 5, 5, 6, 6, 6,
6,
       5, 5, 7, 6, 6, 6, 6, 6, 5, 6, 5, 5, 6, 6, 7, 6, 7, 6, 5, 7, 6,
6,
       6, 7, 6, 6, 5, 6, 5, 6, 5, 5, 6, 6, 5, 6, 6, 5, 5, 6, 6, 5, 6,
5,
       6, 6, 6, 6, 7, 6, 5, 6, 6, 6, 5, 5, 6, 7, 5, 6, 6, 7, 6, 7, 5,
5,
       6, 5, 6, 5, 6, 6, 5, 6, 6, 6, 6, 5, 6, 7, 6, 6, 5, 5, 6, 5, 5,
7,
       5, 5, 5, 5, 6, 5, 5, 5, 6, 5, 5, 5, 6, 5, 5, 6, 5, 5])
```

```
y_test
```

```
1109    6
1032    5
1002    7
487     6
979     5
       ..
801     5
61      5
431     5
1210    6
713     5
Name: quality, Length: 480, dtype: int64
```

Evalution of Logistic Regression model

```
## Accuracy Score

from sklearn.metrics import accuracy_score,
confusion_matrix,classification_report,roc_auc_score,roc_curve

accuracy_score(y_test,pred)

0.6479166666666667

pd.crosstab(y_test,pred)

col_0      5    6   7
quality
5        149   58   1
6         61  154  12
7          2   35   8

#Accuracy Score
(149+154+8)/480

0.6479166666666667

# classification report

print(classification_report(y_test,pred))

              precision    recall  f1-score   support

           5       0.70      0.72      0.71       208
           6       0.62      0.68      0.65       227
           7       0.38      0.18      0.24        45

    accuracy                           0.65       480
   macro avg       0.57      0.52      0.53       480
```

```
weighted avg       0.64      0.65      0.64       480
```

```
probability  = model1.predict_proba(x_test)[:,1]
probability
```

```
array([0.53150532, 0.39597011, 0.49084053, 0.44770254, 0.55791885,
       0.20347566, 0.46650276, 0.57446715, 0.34644901, 0.22818244,
       0.38489914, 0.4706061 , 0.55349917, 0.42302441, 0.50900012,
       0.52094817, 0.36440498, 0.50427077, 0.6265948 , 0.3079797 ,
       0.66082196, 0.37326387, 0.53275227, 0.34537031, 0.41472884,
       0.17791759, 0.22485426, 0.52760594, 0.21670255, 0.56687242,
       0.50162091, 0.51475507, 0.55322487, 0.43496075, 0.67513758,
       0.50784619, 0.30202844, 0.49568572, 0.55821228, 0.61759303,
       0.39952842, 0.62922234, 0.31294952, 0.44415069, 0.55352207,
       0.18622512, 0.47651326, 0.51584474, 0.47173833, 0.48926058,
       0.37791298, 0.28982122, 0.4924904 , 0.50544626, 0.46378063,
       0.30191333, 0.15225221, 0.37852586, 0.47551953, 0.47733845,
       0.40868743, 0.52329431, 0.64449682, 0.55558065, 0.33754362,
       0.61559307, 0.4755842 , 0.63502227, 0.50931364, 0.60569177,
       0.43767133, 0.40065408, 0.3720695 , 0.43626506, 0.46420318,
       0.61657683, 0.4611961 , 0.30768558, 0.38989673, 0.52383521,
       0.46648586, 0.26785646, 0.55827759, 0.54403331, 0.51097572,
       0.49486054, 0.48184196, 0.32561775, 0.39044022, 0.30337893,
       0.19783483, 0.36928185, 0.64184452, 0.44820401, 0.5304037 ,
       0.43767133, 0.55100917, 0.43442139, 0.45314934, 0.52627588,
       0.36769907, 0.51556901, 0.56856141, 0.42795691, 0.51063727,
       0.35509467, 0.51063727, 0.43582437, 0.61611001, 0.37426135,
       0.63510968, 0.31998448, 0.56538462, 0.23453008, 0.6082299 ,
       0.51877964, 0.54800198, 0.48494845, 0.64504663, 0.53275227,
       0.20132684, 0.5380445 , 0.47583905, 0.23578568, 0.55606292,
       0.54816221, 0.22873864, 0.30127447, 0.4893277 , 0.52743557,
       0.33503855, 0.44352416, 0.59398279, 0.62071257, 0.47808185,
       0.23814757, 0.50749717, 0.36970221, 0.47324408, 0.38829854,
       0.68078924, 0.25681657, 0.47651326, 0.45376001, 0.54177381,
       0.52933272, 0.56538462, 0.49306699, 0.59397912, 0.33624617,
       0.52563598, 0.50271957, 0.43582437, 0.54199002, 0.56719961,
       0.51848393, 0.28667407, 0.44818676, 0.65543534, 0.58419703,
       0.5924445 , 0.49896605, 0.5214487 , 0.32105837, 0.59239165,
       0.36681826, 0.63921295, 0.46841464, 0.44007502, 0.58880311,
       0.49492308, 0.4369482 , 0.30632078, 0.43150299, 0.67428198,
       0.5159442 , 0.51301431, 0.36602798, 0.66372173, 0.15024981,
       0.23784006, 0.35935013, 0.46283212, 0.5344851 , 0.39422073,
       0.22690015, 0.46946844, 0.56856141, 0.16855589, 0.5446174 ,
       0.62092534, 0.51406556, 0.38981352, 0.42037539, 0.32420057,
       0.21475193, 0.38489914, 0.44268216, 0.38976206, 0.21209832,
       0.2534914 , 0.60398891, 0.25731228, 0.46414581, 0.43601184,
       0.22056909, 0.32904426, 0.53843849, 0.53858537, 0.34277108,
       0.48160125, 0.33634758, 0.46132477, 0.43903506, 0.34959141,
       0.48513992, 0.52394974, 0.49876087, 0.5087562 , 0.15965457,
```

```
        0.34382867, 0.5358172 , 0.5279745 , 0.46426808, 0.4714453 ,
        0.50318762, 0.49996977, 0.54511067, 0.27276582, 0.45833834,
        0.25835893, 0.65664969, 0.45523992, 0.54303038, 0.55967067,
        0.55625452, 0.55830186, 0.47680356, 0.47921261, 0.59244841,
        0.43817818, 0.35553023, 0.3296707 , 0.37620307, 0.5524995 ,
        0.16657507, 0.40575898, 0.32842204, 0.32138276, 0.46229747,
        0.20211231, 0.46627482, 0.20064209, 0.48538934, 0.48148719,
        0.43099882, 0.1932154 , 0.36644074, 0.2719787 , 0.4671032 ,
        0.47590899, 0.41336766, 0.35213815, 0.47066282, 0.42830653,
        0.2700167 , 0.4921495 , 0.24537988, 0.5379711 , 0.44814461,
        0.44460804, 0.48770209, 0.47184309, 0.52185802, 0.51799119,
        0.62926781, 0.50941237, 0.24248865, 0.39505467, 0.54045912,
        0.40614242, 0.44575354, 0.48770209, 0.55555987, 0.47661048,
        0.29861897, 0.42296716, 0.40726488, 0.53085419, 0.49497586,
        0.51221867, 0.62901533, 0.53872858, 0.44525015, 0.367229  ,
        0.63046678, 0.39647179, 0.29425336, 0.33451465, 0.26121464,
        0.61387338, 0.51464509, 0.38450215, 0.42266203, 0.53644704,
        0.65980175, 0.52427556, 0.38949469, 0.35232572, 0.39107439,
        0.44767695, 0.61638882, 0.42712509, 0.43442139, 0.34232281,
        0.40266753, 0.58611226, 0.5188394 , 0.42857268, 0.49795538,
        0.39107439, 0.4307108 , 0.51965867, 0.59839841, 0.53775125,
        0.38361376, 0.4362972 , 0.42074019, 0.23595533, 0.3337573 ,
        0.43076653, 0.36892376, 0.324472  , 0.5446174 , 0.34519008,
        0.49405271, 0.51786256, 0.49423931, 0.41673561, 0.49218751,
        0.23602226, 0.38350526, 0.4755842 , 0.64316573, 0.63046678,
        0.42418184, 0.45111492, 0.34908776, 0.47008826, 0.48186694,
        0.54399137, 0.55606815, 0.41985758, 0.52577026, 0.39556197,
        0.38943576, 0.62092534, 0.66046354, 0.56864703, 0.54463695,
        0.30574811, 0.46229747, 0.34663094, 0.56908312, 0.4605853 ,
        0.38948131, 0.49568572, 0.5103037 , 0.25172872, 0.49883187,
        0.48687764, 0.51624886, 0.47545672, 0.66095441, 0.48125739,
        0.32834732, 0.50271957, 0.57103008, 0.55625452, 0.63545629,
        0.59290446, 0.57966898, 0.29614956, 0.47283183, 0.39706761,
        0.55718141, 0.6034336 , 0.39268931, 0.52858971, 0.44185418,
        0.47728637, 0.3613848 , 0.24187051, 0.45936178, 0.51819761,
        0.53585007, 0.42624614, 0.44767695, 0.54512089, 0.34157811,
        0.38670478, 0.34157811, 0.39534807, 0.50935646, 0.42144687,
        0.37852586, 0.63225881, 0.56222574, 0.41038172, 0.5986983 ,
        0.48513992, 0.22648282, 0.15012169, 0.61611072, 0.5833698 ,
        0.25681506, 0.51058033, 0.26249644, 0.40142808, 0.65512133,
        0.53617015, 0.5349615 , 0.44933895, 0.48712111, 0.31003693,
        0.61906831, 0.48142796, 0.54451505, 0.31855612, 0.32842204,
        0.39567667, 0.52858971, 0.36928185, 0.53178322, 0.5181658 ,
        0.4527915 , 0.56747712, 0.48147361, 0.42634741, 0.48166013,
        0.54271393, 0.43099337, 0.59244841, 0.3959078 , 0.50119993,
        0.45785718, 0.49423931, 0.3359988 , 0.32818398, 0.5054023 ,
        0.60664132, 0.21422976, 0.58823342, 0.42439346, 0.64922113,
        0.31518466, 0.52339775, 0.30275806, 0.53872858, 0.42224183,
        0.45360159, 0.36985584, 0.38829854, 0.25753186, 0.24079774,
```

```
       0.20347566, 0.51612485, 0.36859038, 0.47251759, 0.19698817,
       0.60385457, 0.44153189, 0.43028571, 0.39426744, 0.49497586,
       0.4743997 , 0.17385998, 0.262994  , 0.55718141, 0.30275806])

#Logistic Reg model is better than Linear Reg model as seen from the
accuracy score from this dataset
```

Decision Tree Classifier

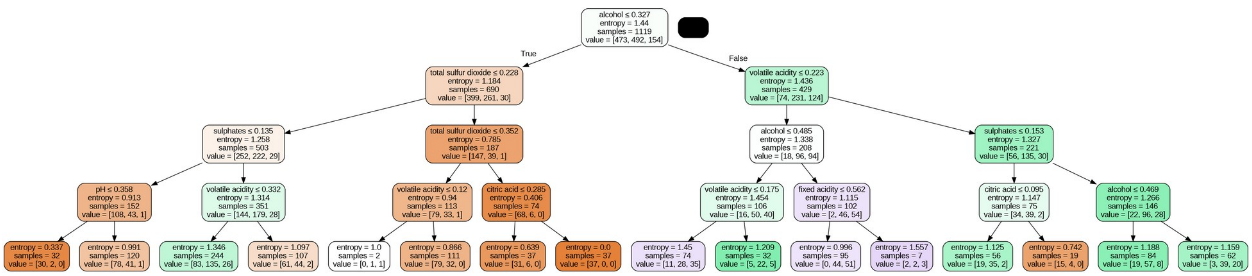
```
from sklearn.tree import DecisionTreeClassifier

model2 =
DecisionTreeClassifier(max_depth=4,splitter='best',criterion='entropy'
)

model2.fit(x_train,y_train)

DecisionTreeClassifier(criterion='entropy', max_depth=4)

y_pred = model2.predict(x_test)
y_pred

array([6, 5, 7, 5, 6, 5, 5, 7, 5, 5, 5, 5, 5, 6, 6, 7, 7, 5, 5, 5, 6,
6,
       5, 7, 6, 5, 5, 7, 5, 6, 6, 6, 6, 5, 6, 7, 5, 6, 7, 6, 5, 6, 7,
6,
       6, 5, 5, 7, 6, 7, 6, 5, 6, 6, 6, 5, 5, 5, 7, 6, 5, 6, 6, 7, 5,
6,
       5, 6, 7, 5, 5, 6, 5, 5, 6, 6, 6, 5, 5, 6, 6, 5, 6, 7, 5, 5, 7,
5,
       6, 5, 5, 5, 6, 5, 7, 5, 7, 5, 6, 6, 7, 7, 6, 7, 5, 5, 5, 6, 6,
5,
       6, 5, 7, 5, 6, 6, 6, 7, 6, 5, 5, 6, 6, 5, 6, 7, 5, 5, 6, 6, 5,
5,
       6, 5, 6, 5, 6, 5, 6, 5, 6, 5, 5, 5, 5, 6, 7, 7, 6, 6, 7, 7, 6,
5,
       5, 6, 6, 5, 6, 6, 5, 6, 7, 6, 5, 5, 6, 7, 6, 6, 5, 6, 5, 7, 6,
7,
       7, 7, 6, 5, 5, 7, 6, 6, 7, 5, 6, 6, 5, 7, 6, 6, 6, 6, 6, 5, 5,
5,
       5, 5, 5, 6, 5, 5, 5, 5, 5, 7, 6, 6, 7, 7, 6, 6, 5, 6, 6, 7, 6,
5,
       5, 6, 7, 6, 6, 6, 7, 6, 5, 6, 5, 6, 5, 6, 7, 6, 6, 7, 7, 6, 5,
5,
       5, 5, 6, 5, 6, 5, 5, 7, 5, 5, 5, 5, 6, 6, 5, 5, 5, 5, 5, 7, 6,
6,
       5, 5, 5, 5, 6, 7, 5, 6, 5, 5, 6, 6, 7, 5, 7, 6, 6, 7, 6, 6, 6,
5,
       5, 6, 7, 6, 6, 6, 6, 6, 5, 6, 5, 6, 5, 5, 6, 6, 6, 6, 7, 6, 6,
```

```
5,
       5, 6, 7, 6, 6, 5, 5, 7, 6, 7, 5, 7, 6, 5, 7, 6, 6, 6, 7, 5, 5,
5,
       7, 5, 5, 7, 5, 6, 6, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 7, 6,
6,
       6, 5, 7, 5, 6, 6, 6, 6, 5, 7, 5, 6, 6, 5, 6, 7, 5, 5, 6, 6, 6,
6,
       5, 5, 7, 6, 6, 6, 6, 6, 5, 6, 5, 5, 6, 6, 7, 6, 7, 6, 5, 7, 6,
6,
       6, 7, 6, 6, 5, 6, 5, 6, 5, 5, 6, 6, 5, 6, 6, 5, 5, 6, 6, 5, 6,
5,
       6, 6, 6, 6, 7, 6, 5, 6, 6, 6, 5, 5, 6, 7, 5, 6, 6, 7, 6, 7, 5,
5,
       6, 5, 6, 5, 6, 6, 5, 6, 6, 6, 6, 5, 6, 7, 6, 6, 5, 5, 6, 5, 5,
7,
       5, 5, 5, 5, 6, 5, 5, 5, 6, 5, 5, 5, 6, 5, 5, 6, 5, 5])

y_pred_train = model2.predict(x_train)
```

Evaluation of Decision Tree classifier

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

print('Testing Accuracy = ', accuracy_score(y_test,y_pred))
print('Training Accuracy = ', accuracy_score(y_train,y_pred_train))

Testing Accuracy =  0.575
Training Accuracy =  0.6336014298480787

pd.crosstab(y_test,y_pred)

col_0      5    6   7
quality
5        130   68  10
6         60  123  44
7          2   20  23

print(classification_report(y_test,y_pred))

              precision    recall  f1-score   support

           5       0.68      0.62      0.65       208
           6       0.58      0.54      0.56       227
           7       0.30      0.51      0.38        45

    accuracy                           0.57       480
   macro avg       0.52      0.56      0.53       480
```

```
weighted avg       0.60      0.57      0.58       480


from six import StringIO
from IPython.display import Image
import pydotplus
from sklearn.tree import export_graphviz

dot_data =StringIO()
export_graphviz(model2,out_file=dot_data,feature_names= X.columns,
                filled=True,rounded= True,special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```



Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
model3 =RandomForestClassifier(criterion='entropy')

model3.fit(x_train,y_train)

RandomForestClassifier(criterion='entropy')

r_y_predict = model3.predict(x_test)
r_y_predict_train = model3.predict(x_train)

print('Testing Accuracy = ', accuracy_score(y_test,r_y_predict))
print('Training Accuracy = ',
accuracy_score(y_train,r_y_predict_train))

Testing Accuracy =  0.6833333333333333
Training Accuracy =  1.0

pd.crosstab(y_test,r_y_predict)

col_0      5    6   7
quality
5        155   48   5
6         61  148  18
7          0   20  25
```

```python
#Accuracy Score
(155+148+25)/480
```

```
0.6833333333333333
```

```python
print(classification_report(y_test,r_y_predict))
```

```
              precision    recall  f1-score   support

           5       0.72      0.75      0.73       208
           6       0.69      0.65      0.67       227
           7       0.52      0.56      0.54        45

    accuracy                           0.68       480
   macro avg       0.64      0.65      0.65       480
weighted avg       0.68      0.68      0.68       480
```

```python
#Better Accuracy, Better model
```