Importing libraries and Beverage quality dataset

```
%matplotlib
```

```
Using matplotlib backend: Qt5Agg
```

```
from numpy import arange
import numpy
from matplotlib import pyplot as plt
from scipy.stats import norm
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn import model_selection
```

```
plt.rcParams['figure.figsize'] = [16, 7]
```

```
columns=['Fixed Acidity','Volatile Acidity','Citric Acid','Chlorides','Free SO2','Total SO2','Sulphates','pH','Quality',
         'Alcohol','Residual Sugar','Density']
df = pd.read_csv(r"C:/Users/Aadya/Downloads/Beverage_Quality.csv")
```

Data Exploration

```
df.columns = columns
df.head()
```

9]:

	Fixed Acidity	Volatile Acidity	Citric Acid	Chlorides	Free SO2	Total SO2	Sulphates	pH	Quality	Alcohol	Residual Sugar	Density
0	7.4	0.70	0.00	0.076	11.0	34.0	0.56	3.51	5	9.4	1.9	0.9978
1	7.8	0.88	0.00	0.098	25.0	67.0	0.68	3.20	5	9.8	2.6	0.9968
2	7.8	0.76	0.04	0.092	15.0	54.0	0.65	3.26	5	9.8	2.3	0.9970
3	11.2	0.28	0.56	0.075	17.0	60.0	0.58	3.16	6	9.8	1.9	0.9980
4	7.4	0.70	0.00	0.076	11.0	34.0	0.56	3.51	5	9.4	1.9	0.9978

```
#Information about the dataset
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   Chlorides         1599 non-null   float64
 4   Free SO2          1599 non-null   float64
 5   Total SO2         1599 non-null   float64
 6   Sulphates         1599 non-null   float64
 7   pH                1599 non-null   float64
 8   Quality           1599 non-null   int64
 9   Alcohol           1599 non-null   float64
 10  Residual Sugar    1599 non-null   float64
 11  Density           1599 non-null   float64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

```python
#Data types of the dataset columns
df.dtypes
```

```
Fixed Acidity       float64
Volatile Acidity    float64
Citric Acid         float64
Chlorides           float64
Free SO2            float64
Total SO2           float64
Sulphates           float64
pH                  float64
Quality               int64
Alcohol             float64
Residual Sugar      float64
Density             float64
dtype: object
```

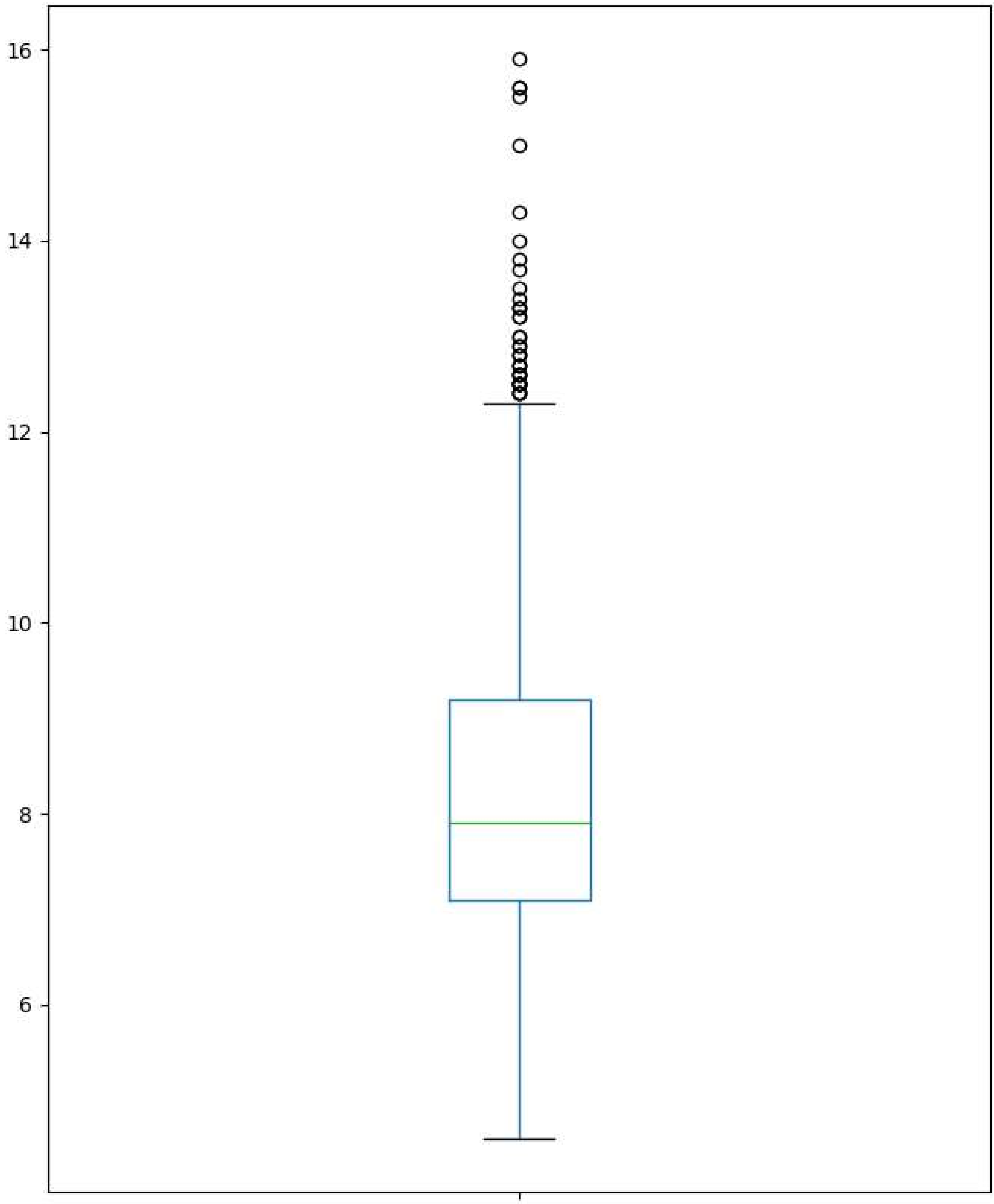
```python
#Memory used by each column in the dataset
df.memory_usage()
```

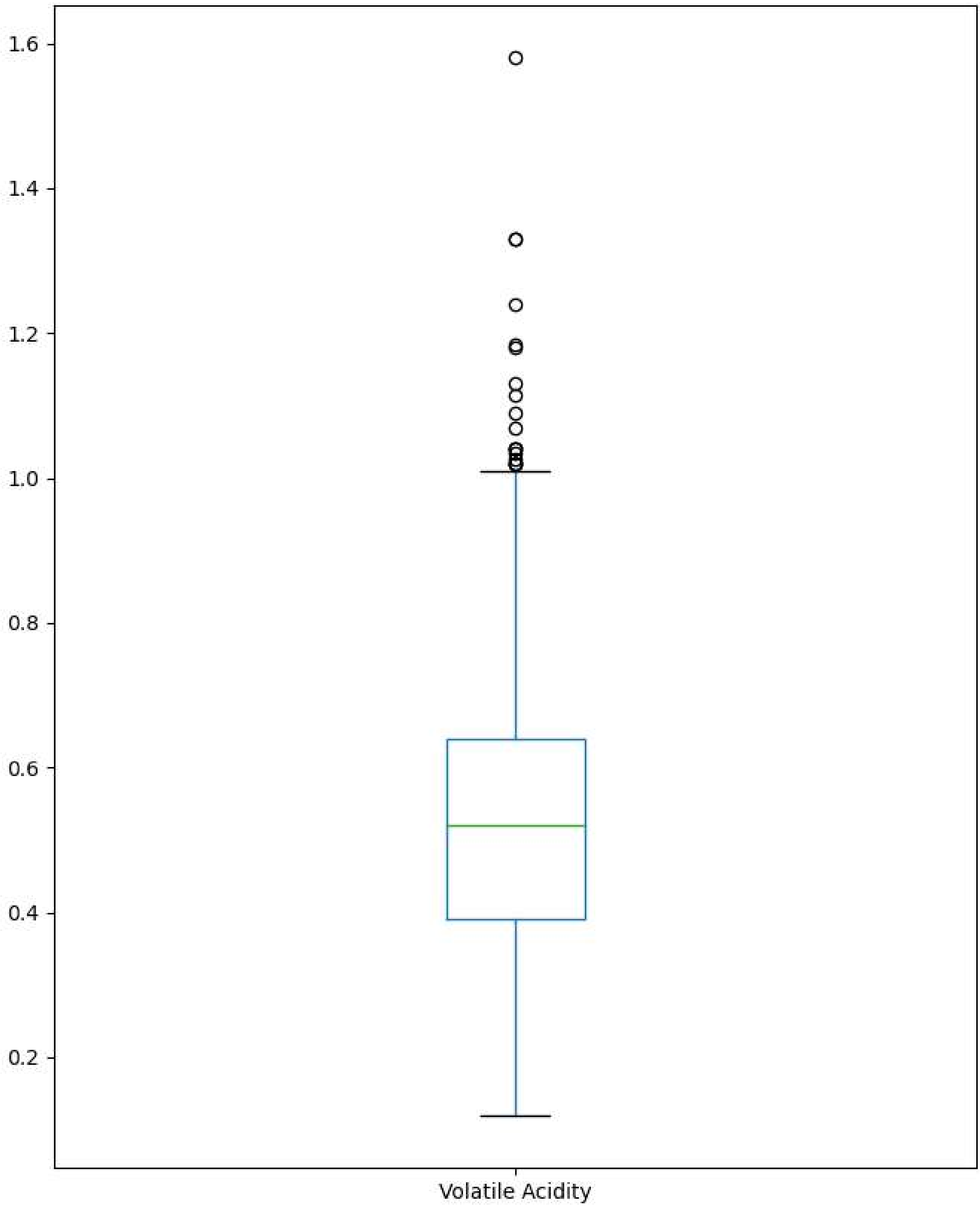
```
Index                 128
Fixed Acidity       12792
Volatile Acidity    12792
Citric Acid         12792
Chlorides           12792
Free SO2            12792
Total SO2           12792
Sulphates           12792
pH                  12792
Quality             12792
Alcohol             12792
Residual Sugar      12792
Density             12792
dtype: int64
```

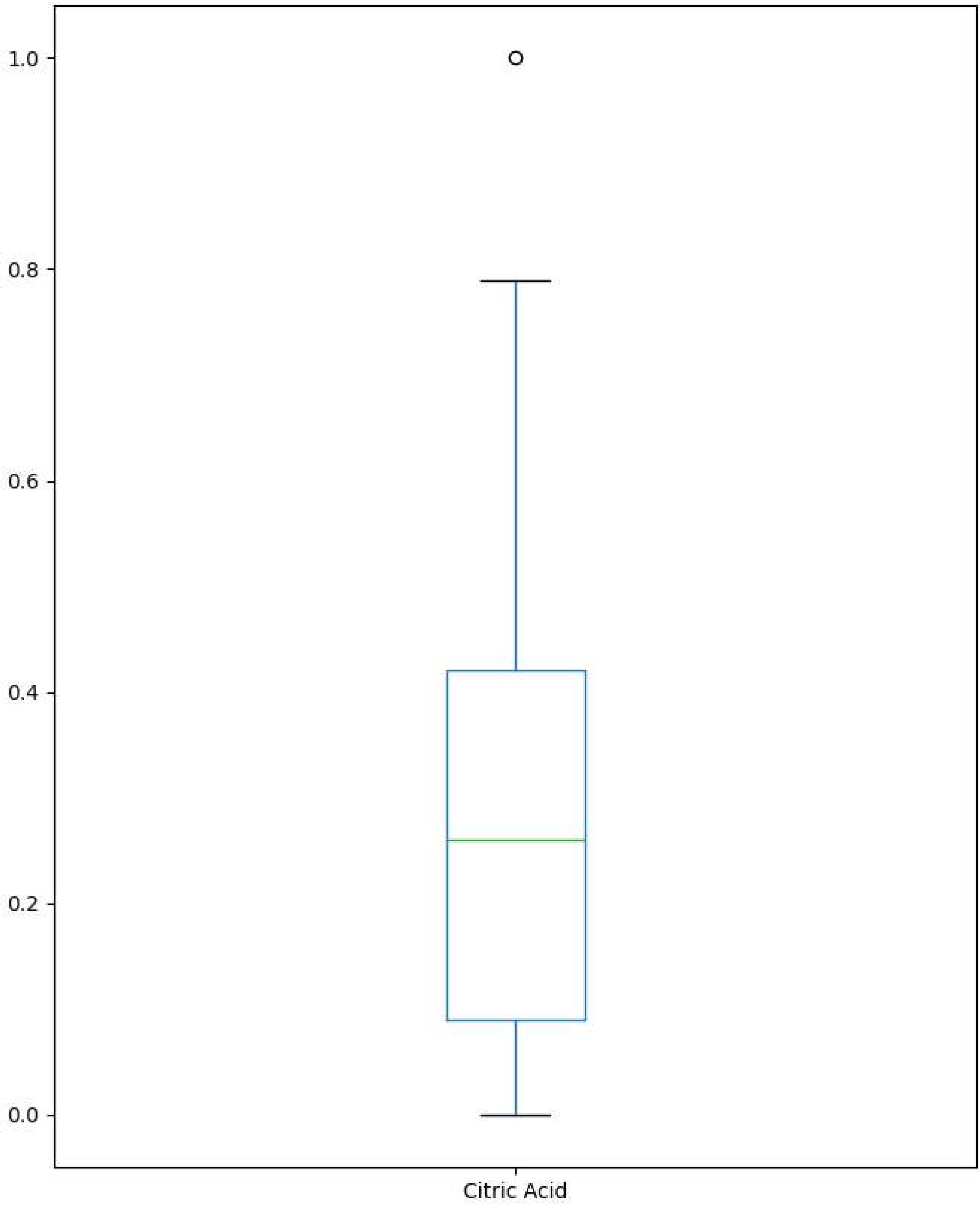
```python
#Total memory used by the dataset
df.memory_usage().sum()
```

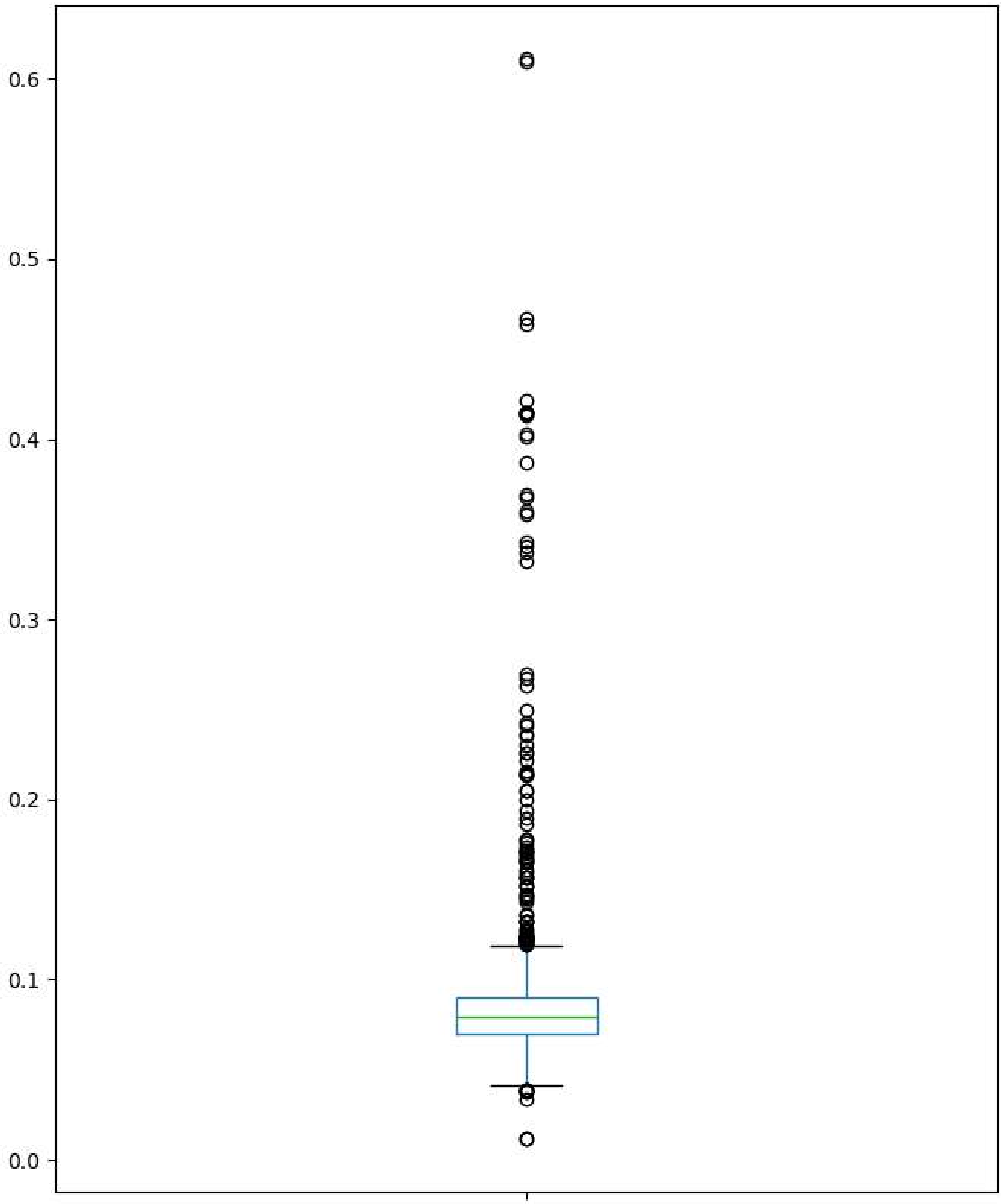
```
153632
```

```
#Boxplot
df['Fixed Acidity'].plot.box(figsize=(8, 15)); # Boxplot of a column
```

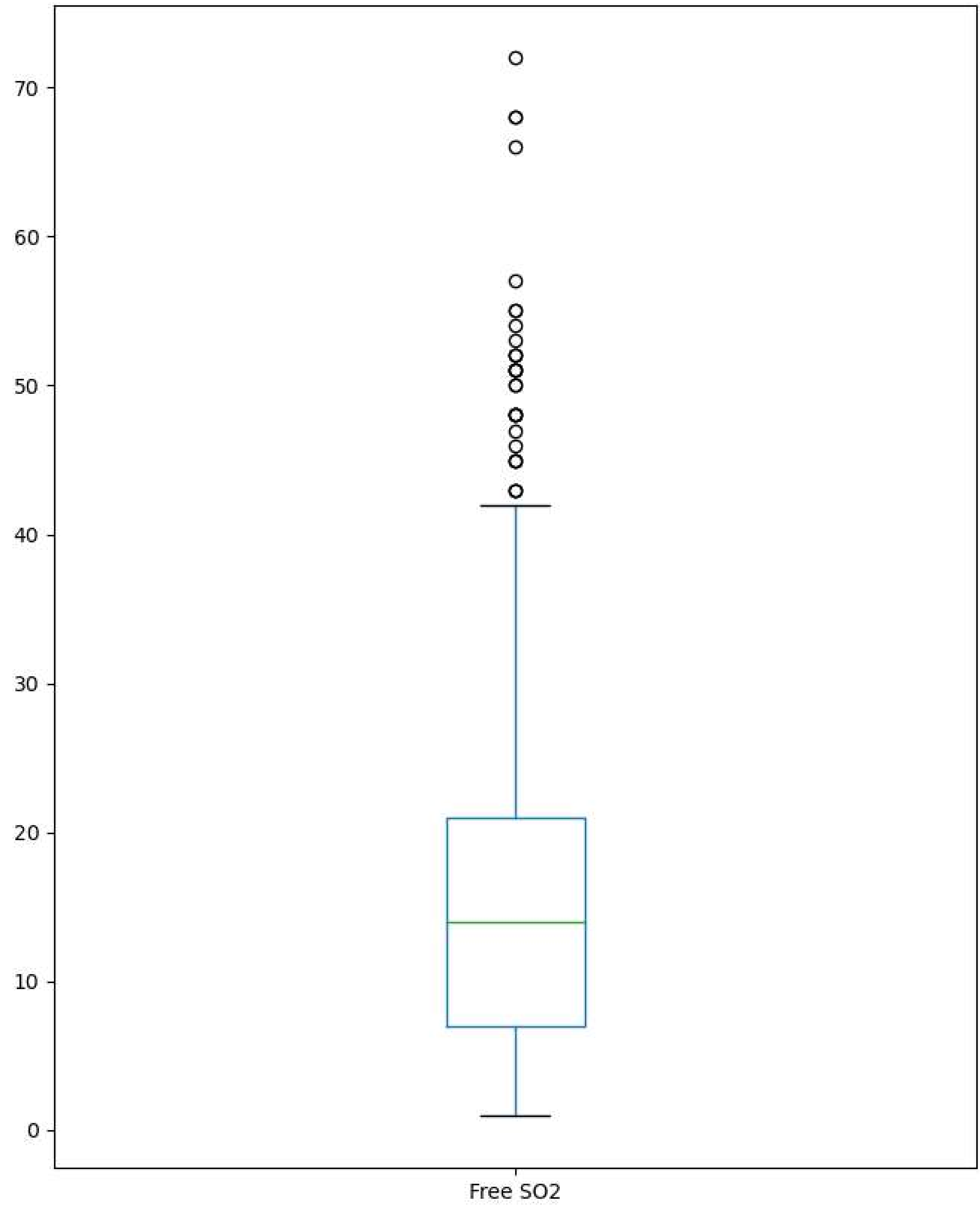

16
14
12
10
8
6
Fixed Acidity

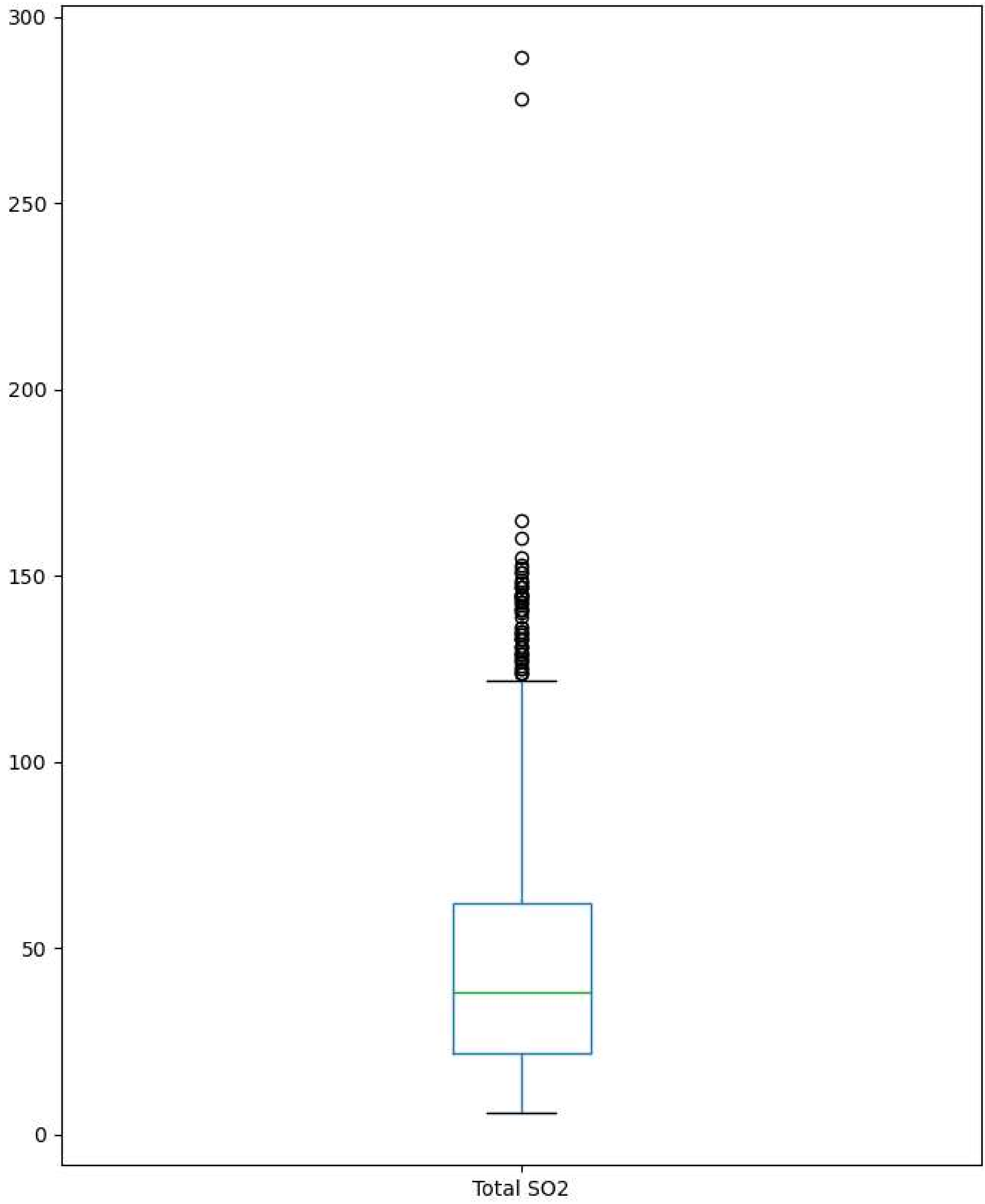

1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
Volatile Acidity

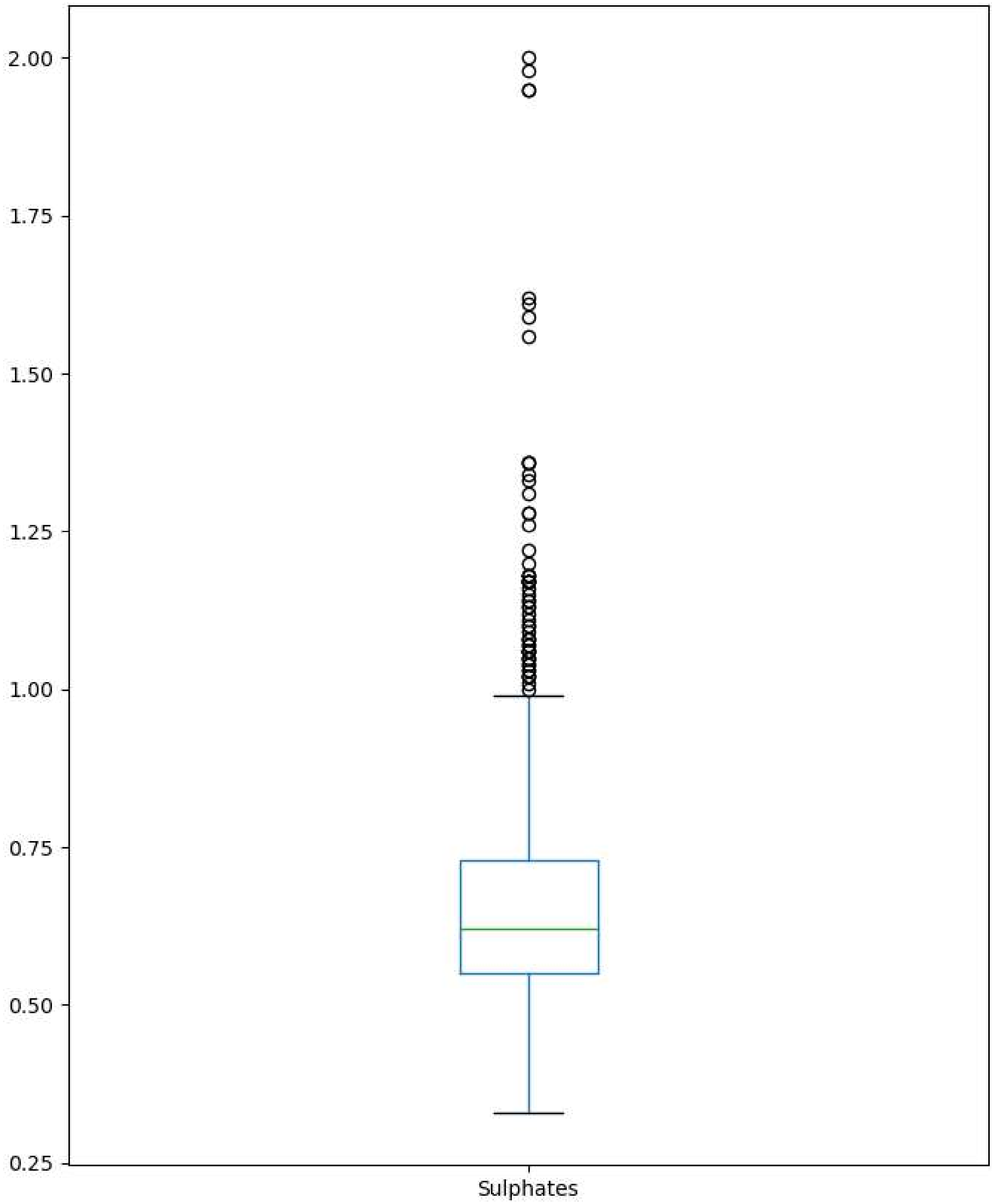

1.0
0.8
0.6
0.4
0.2
0.0
Citric Acid

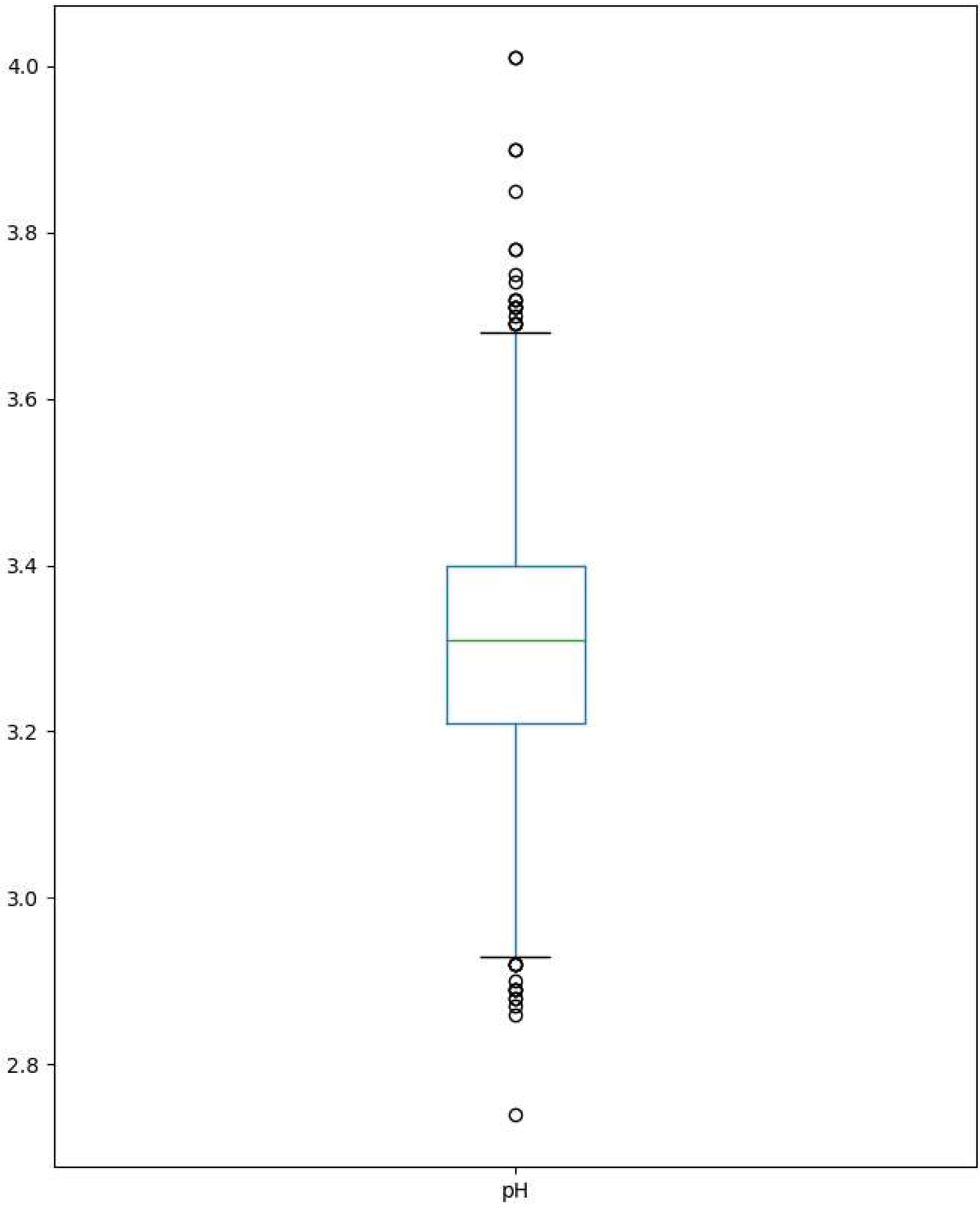


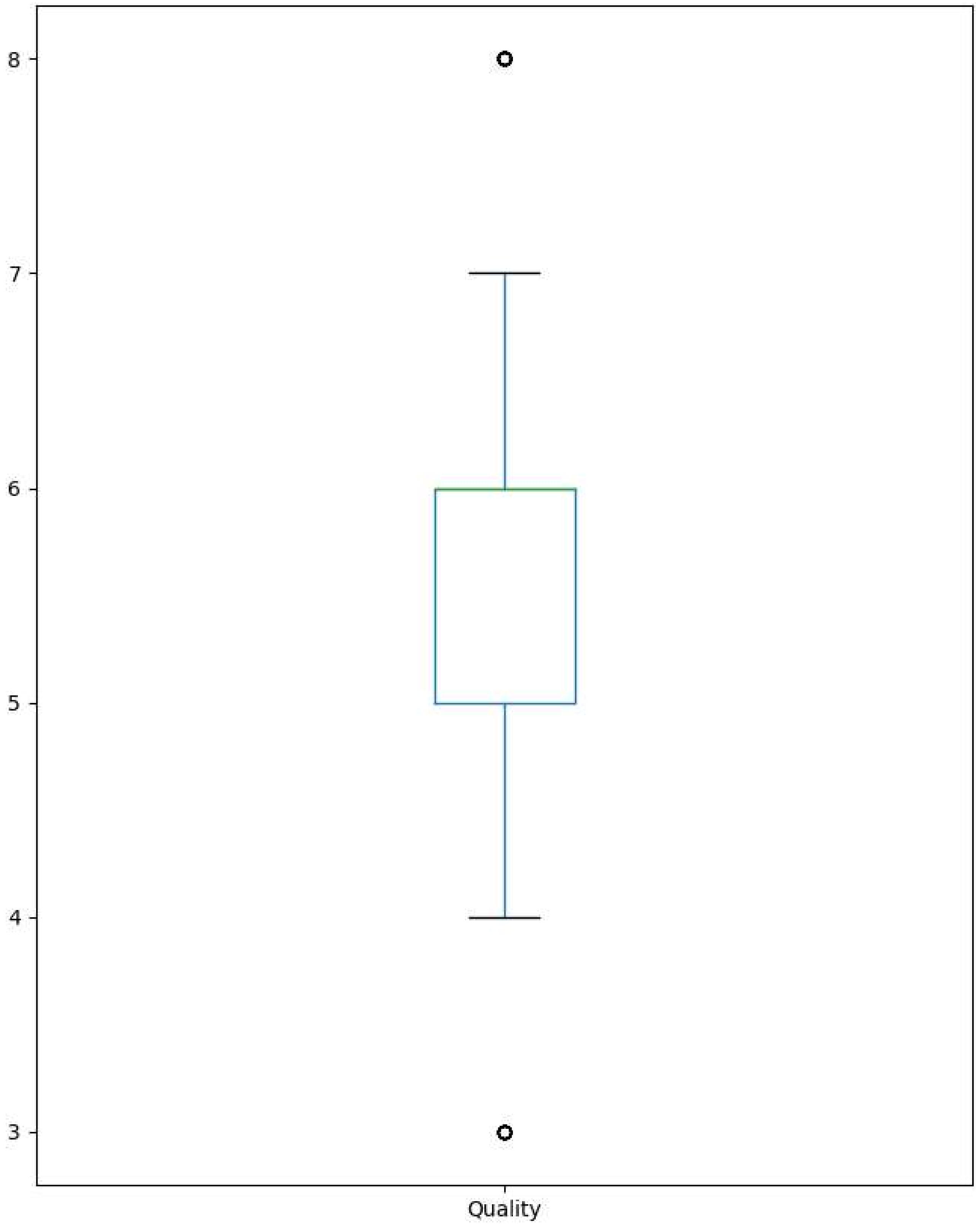
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Chlorides

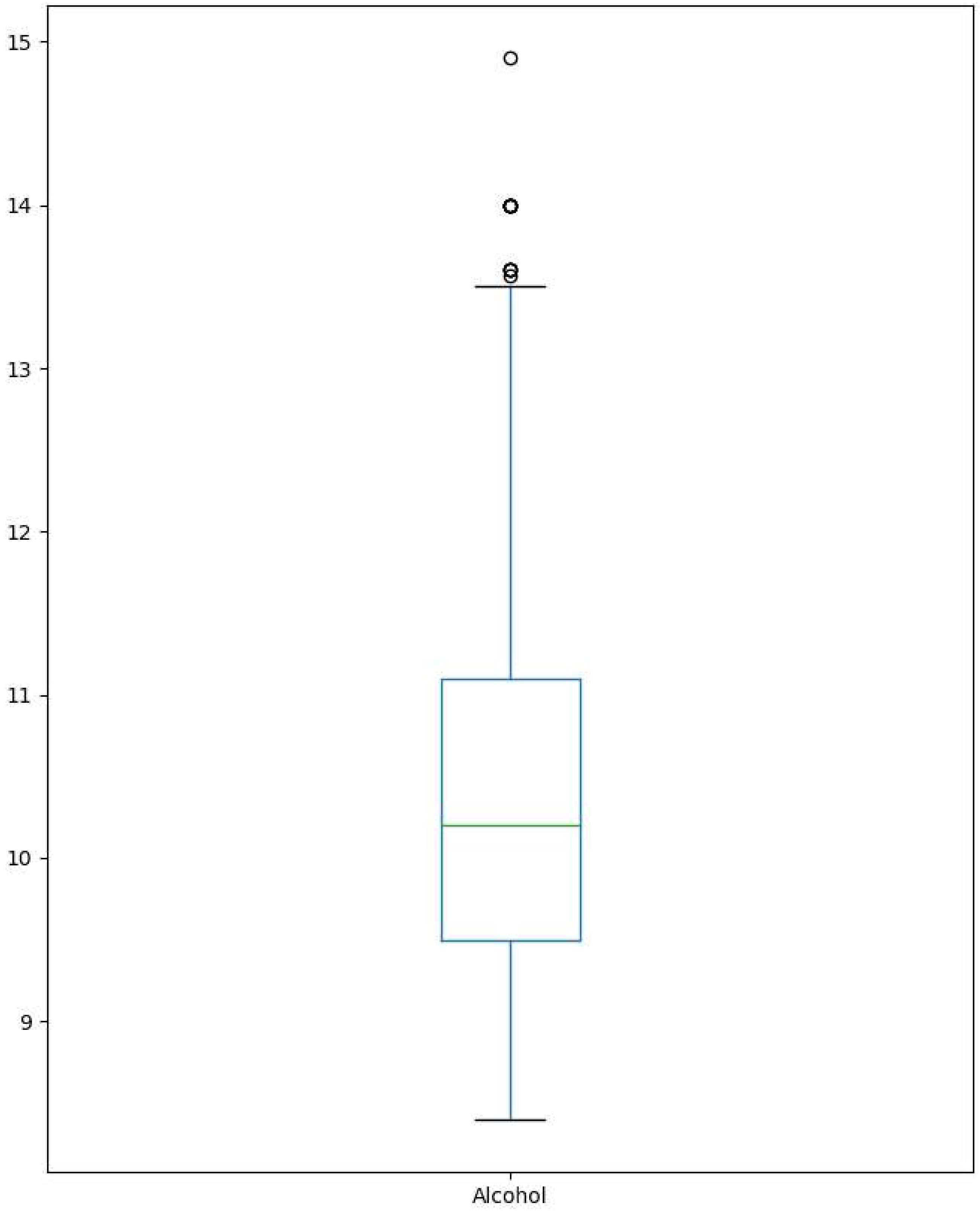

70
60
50
40
30
20
10
0
Free SO2


300
250
200
150
100
50
0
Total SO2

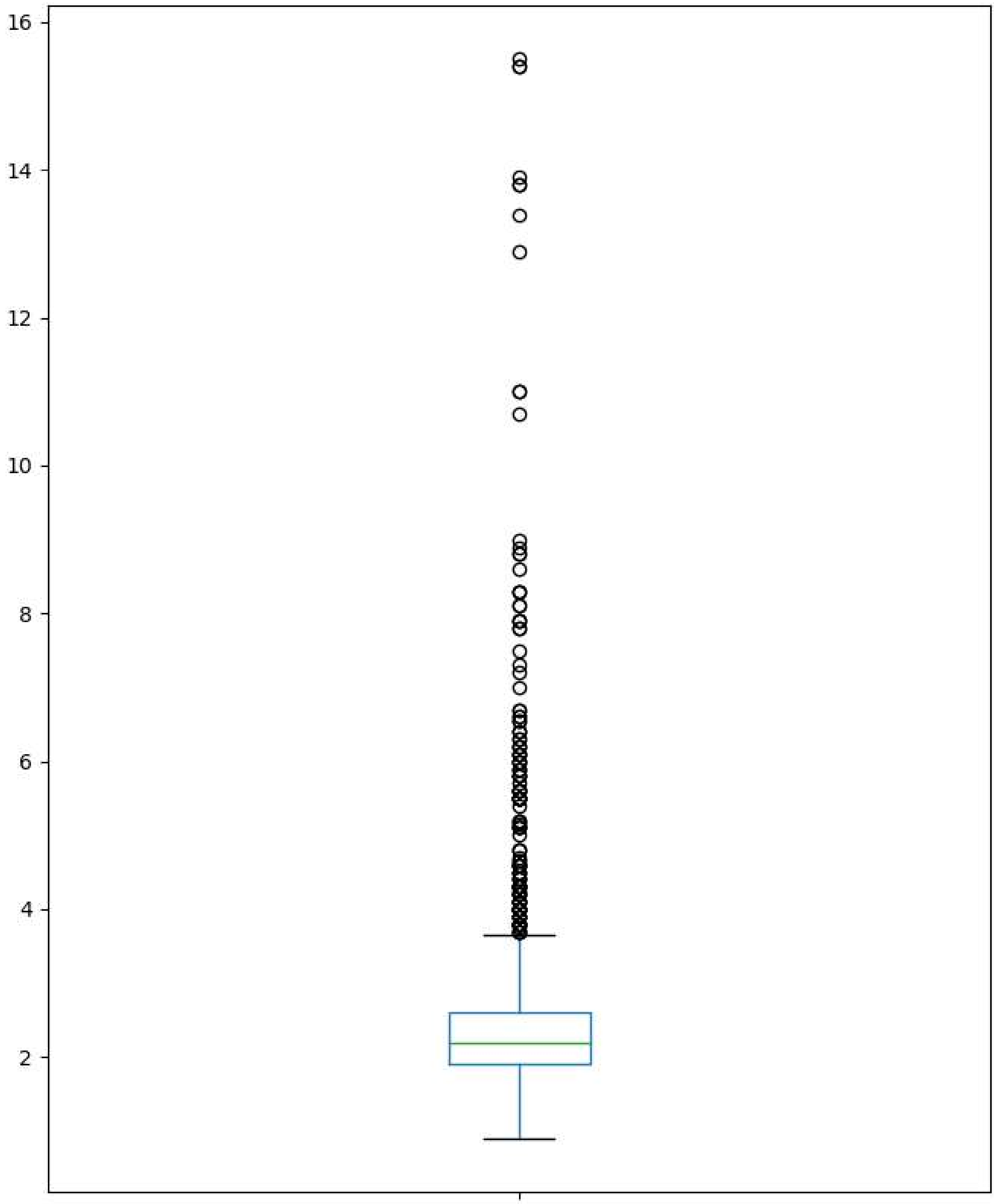

2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
Sulphates

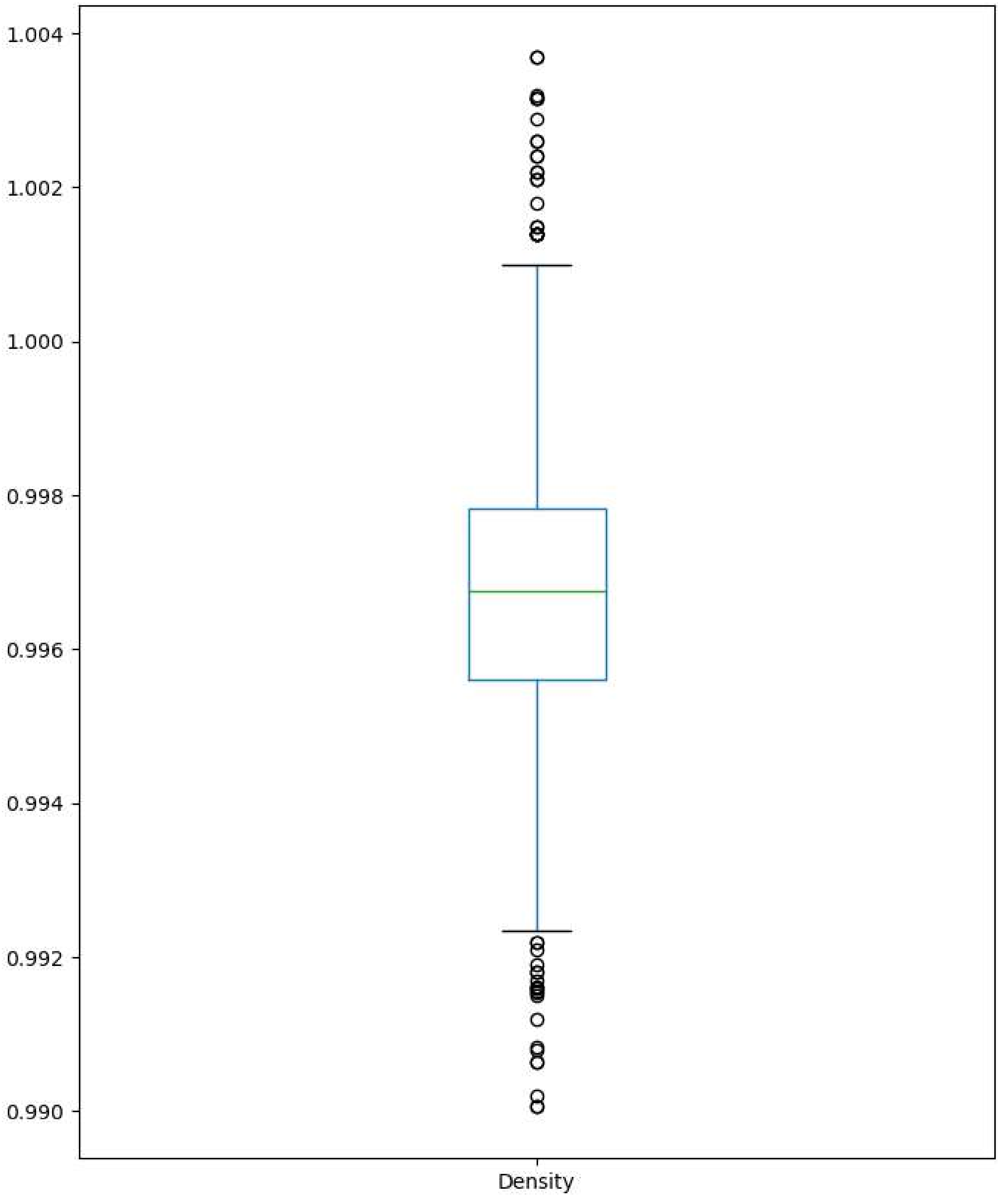

4.0
3.8
3.6
3.4
3.2
3.0
2.8
pH


8
7
6
5
4
3
Quality



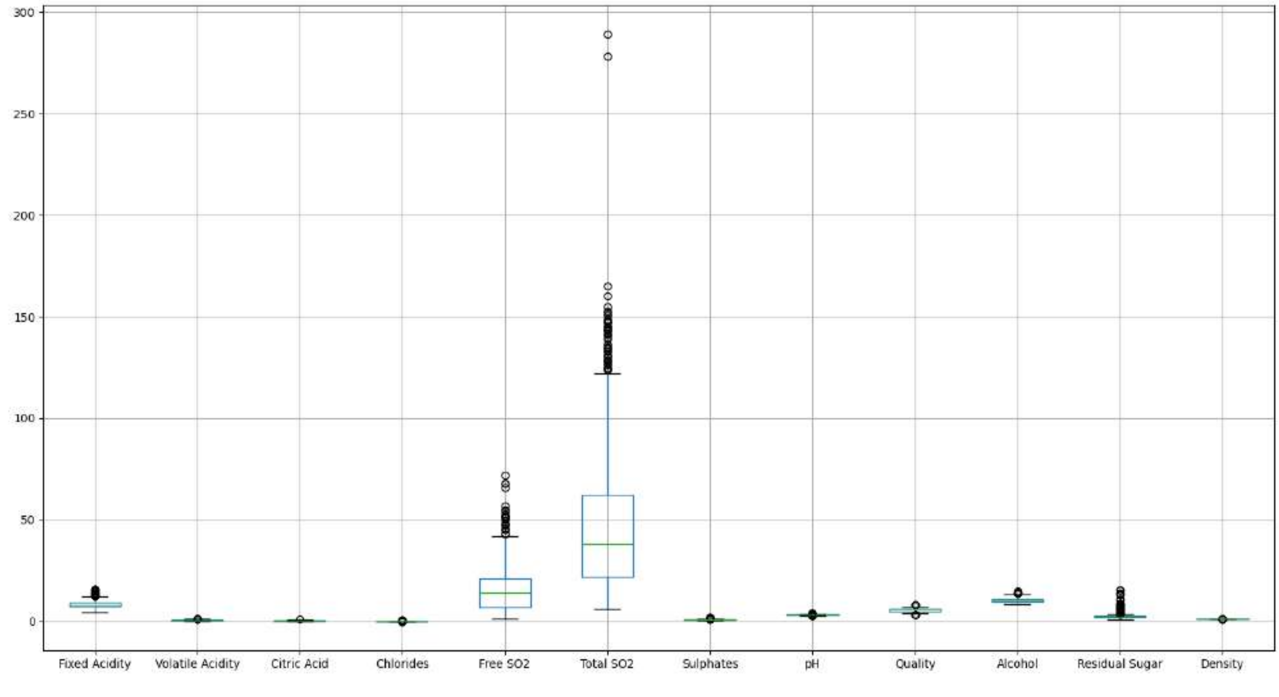
15
14
13
12
11
10
9
Alcohol


16
14
12
10
8
6
4
2
Residual Sugar


1.004
1.002
1.000
0.998
0.996
0.994
0.992
0.990
Density

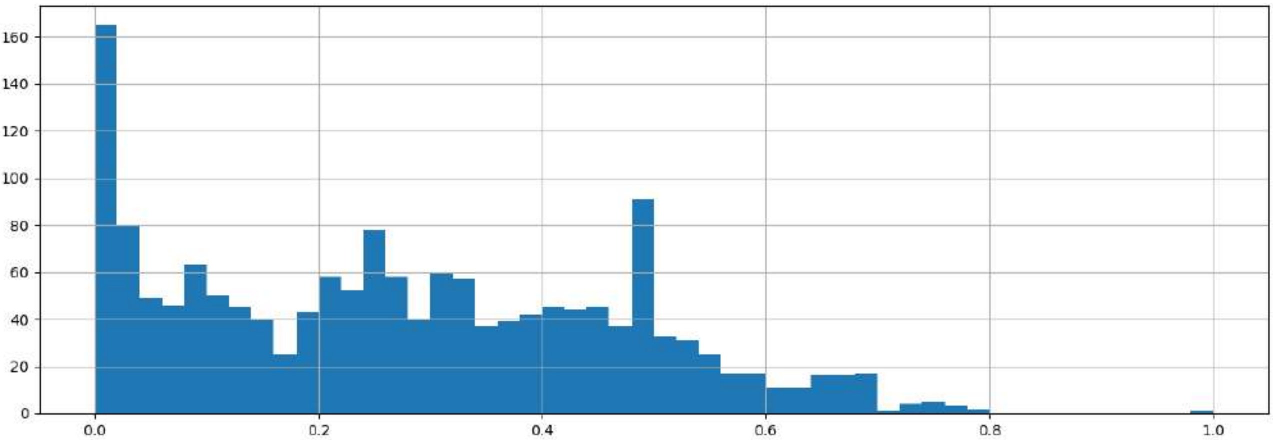
```python
#Boxplot of all the columns with numerical data
df.boxplot(figsize=(20,20))
```

```
<AxesSubplot:>
```



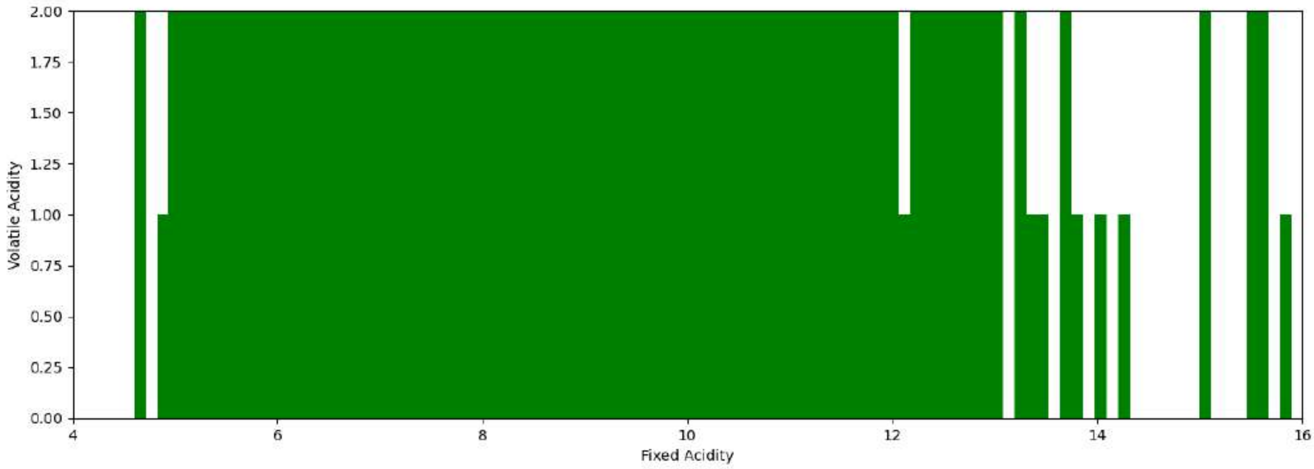
300
250
200
150
100
50
0
Fixed Acidity
Volatile Acidity
Citric Acid
Chlorides
Free SO2
Total SO2
Sulphates
pH
Quality
Alcohol
Residual Sugar
Density

```
#Histogram
df['Citric Acid'].hist(bins=50, figsize=(15, 5));
```



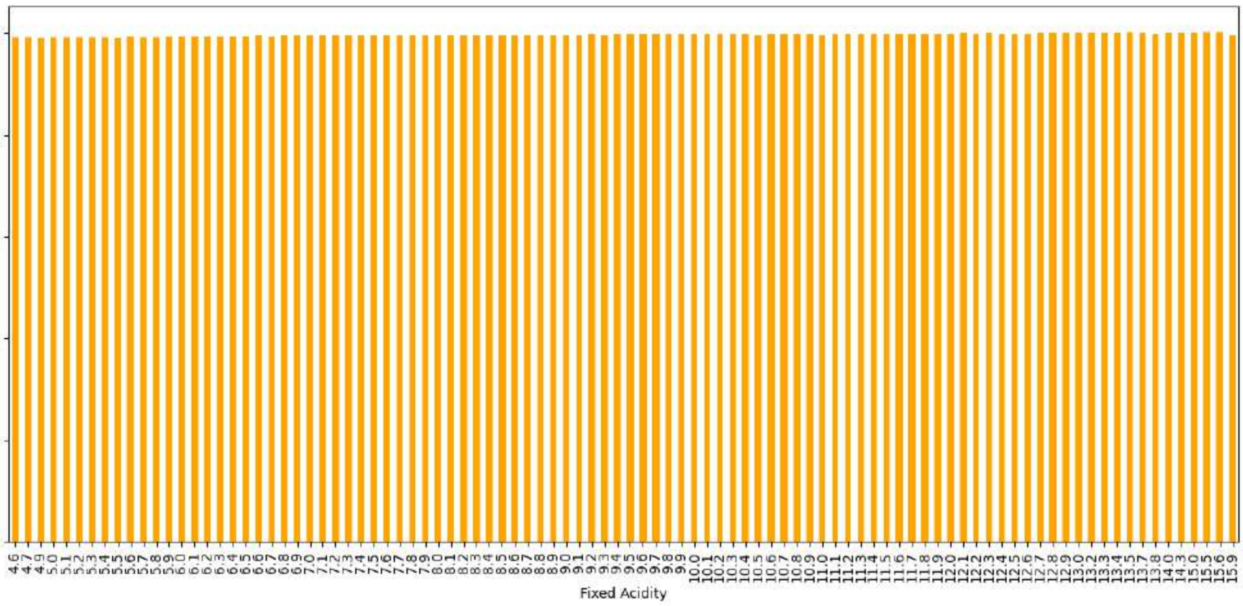
160
140
120
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0

```
ax = df['Fixed Acidity'].hist(bins=100, grid=False, color='green', figsize=(15, 5)) # grid turned off and color changed
ax.set_xlabel('Fixed Acidity')
ax.set_ylabel('Volatile Acidity')
ax.set_xlim(4,16) #limiting display range to 0-6 for the x-axis
ax.set_ylim(0,2); #limiting display range to 0-2 for the y-axis
```

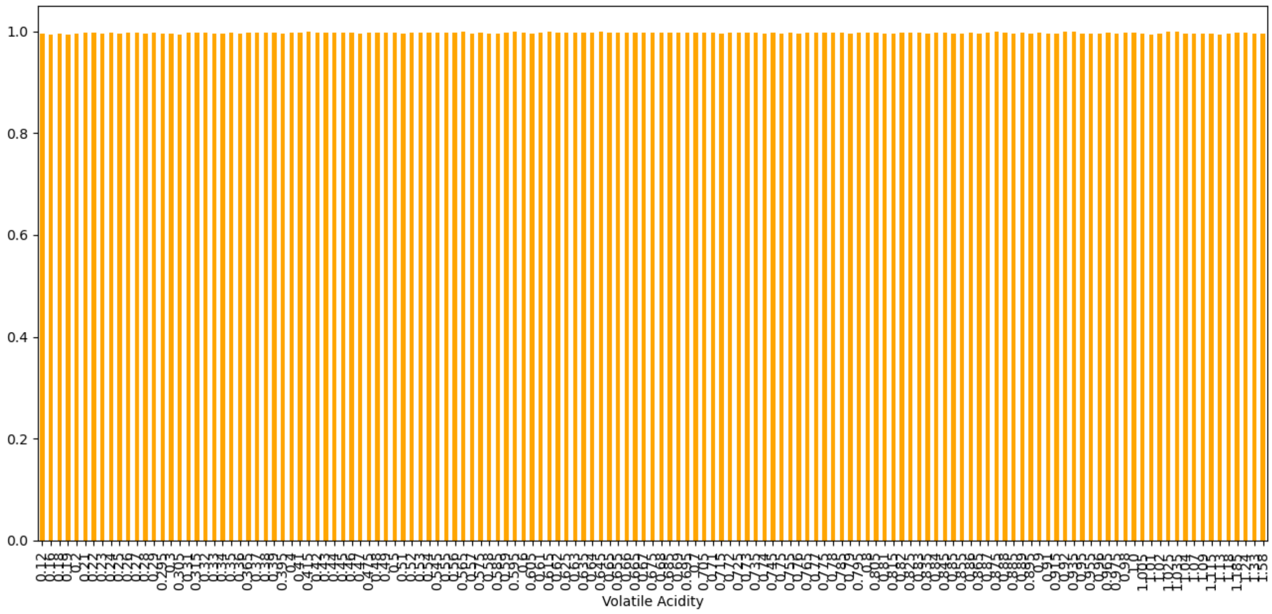


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Volatile Acidity
4
6
8
10
12
14
16
Fixed Acidity

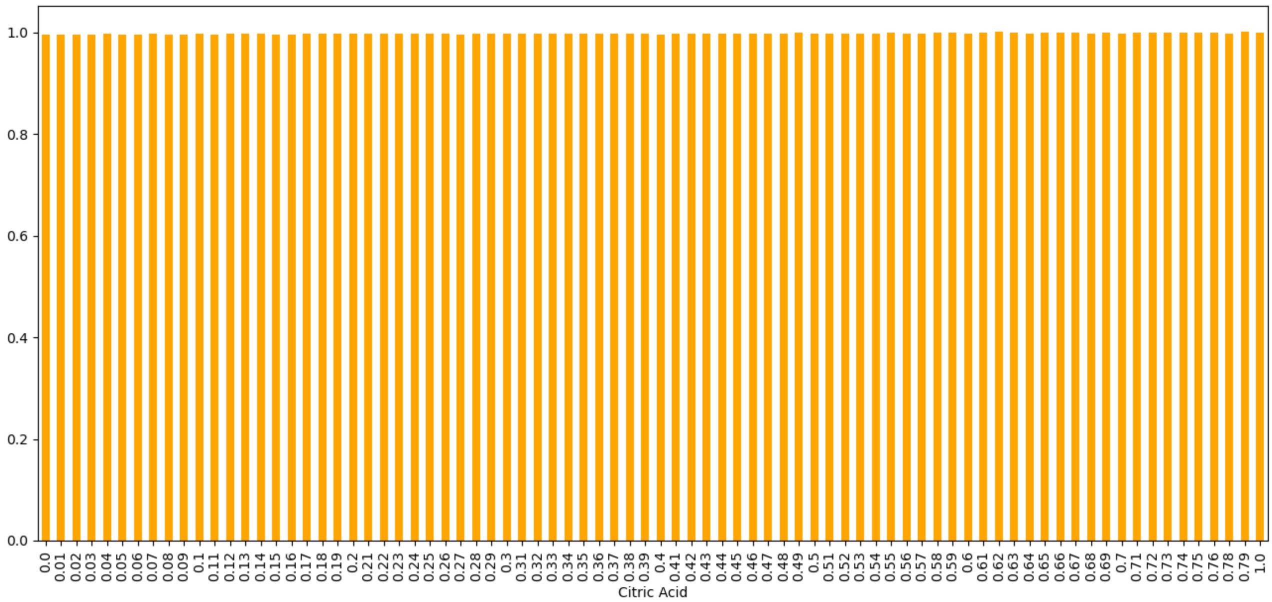
```
#Barplot with Density as dependent variable
df_avg_density = df.groupby('Fixed Acidity')['Density'].mean()
df_avg_density[:].plot.bar(color='orange');
```



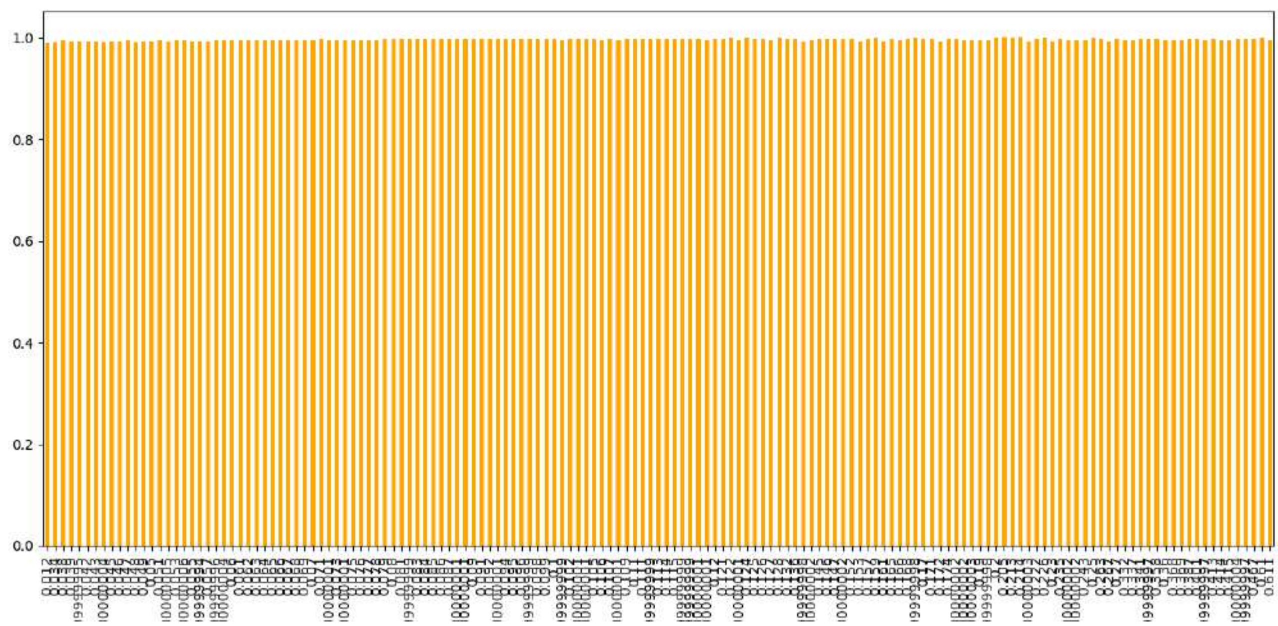
1.0
0.8
0.6
0.4
0.2
0.0
Fixed Acidity

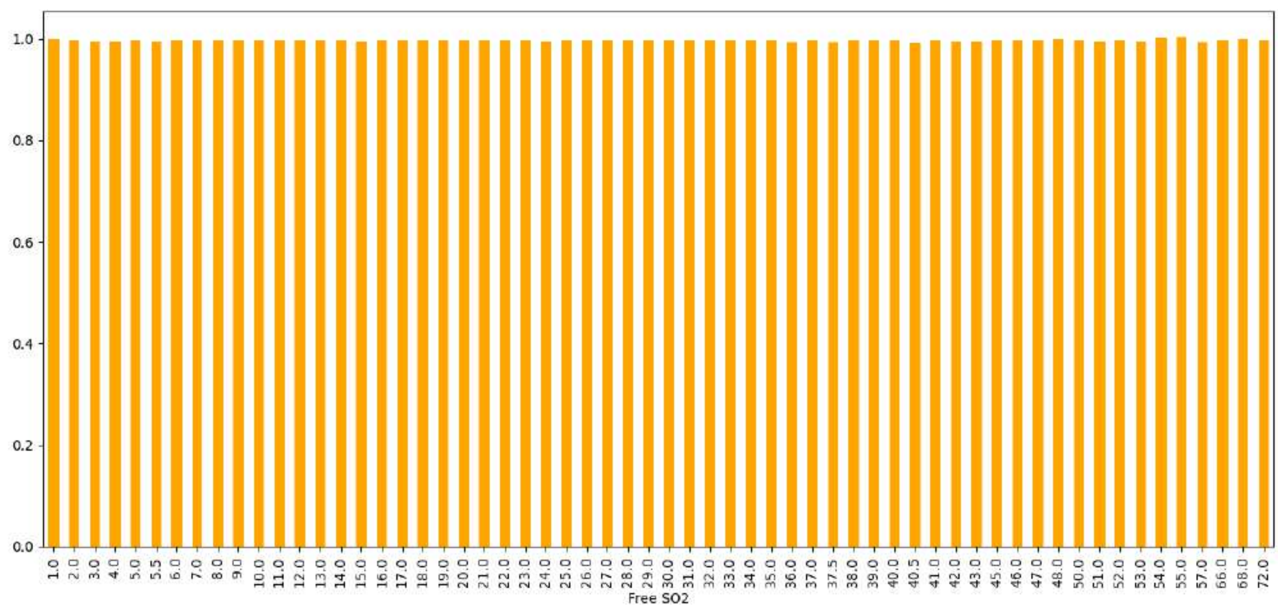


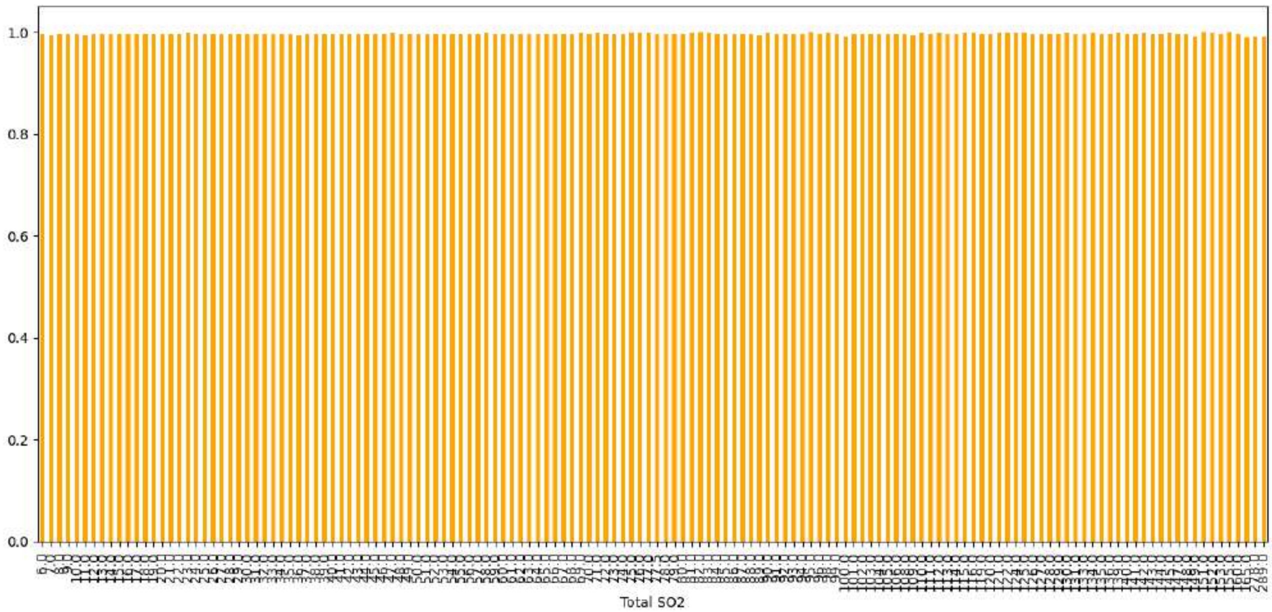
1.0
0.8
0.6
0.4
0.2
0.0
Volatile Acidity



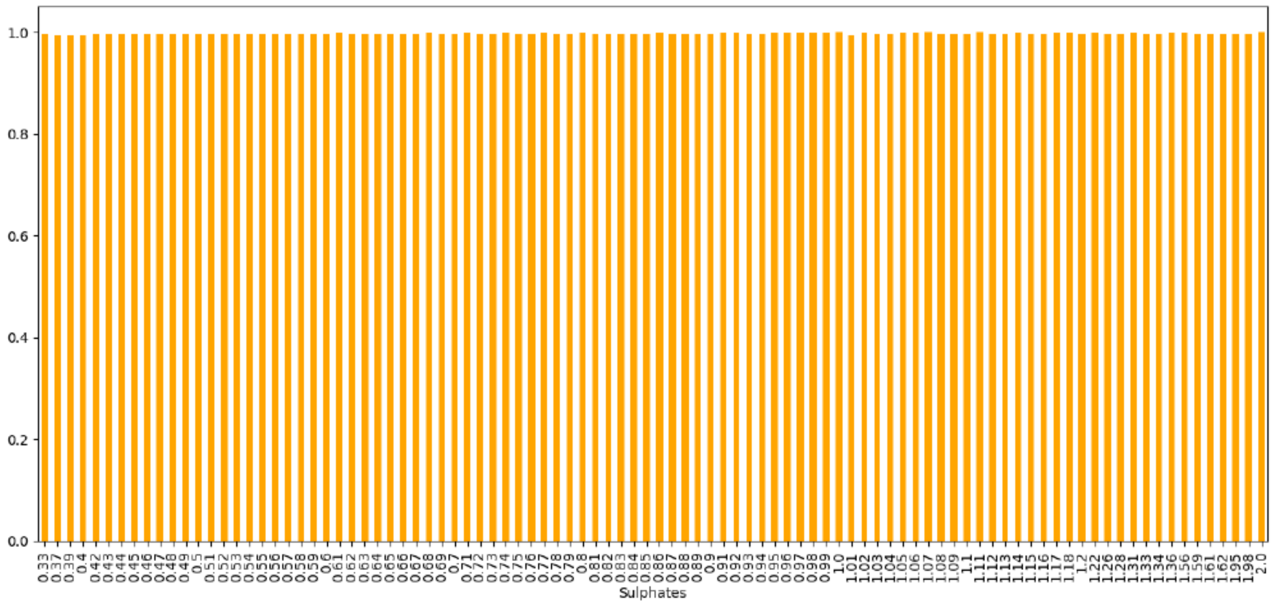
0.0
0.2
0.4
0.6
0.8
1.0
0.0 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.1 0.11 0.12 0.13 0.14 0.15 0.16 0.17 0.18 0.19 0.2 0.21 0.22 0.23 0.24 0.25 0.26 0.27 0.28 0.29 0.3 0.31 0.32 0.33 0.34 0.35 0.36 0.37 0.38 0.39 0.4 0.41 0.42 0.43 0.44 0.45 0.46 0.47 0.48 0.49 0.5 0.51 0.52 0.53 0.54 0.55 0.56 0.57 0.58 0.59 0.6 0.61 0.62 0.63 0.64 0.65 0.66 0.67 0.68 0.69 0.7 0.71 0.72 0.73 0.74 0.75 0.76 0.78 0.79 1.0
Citric Acid

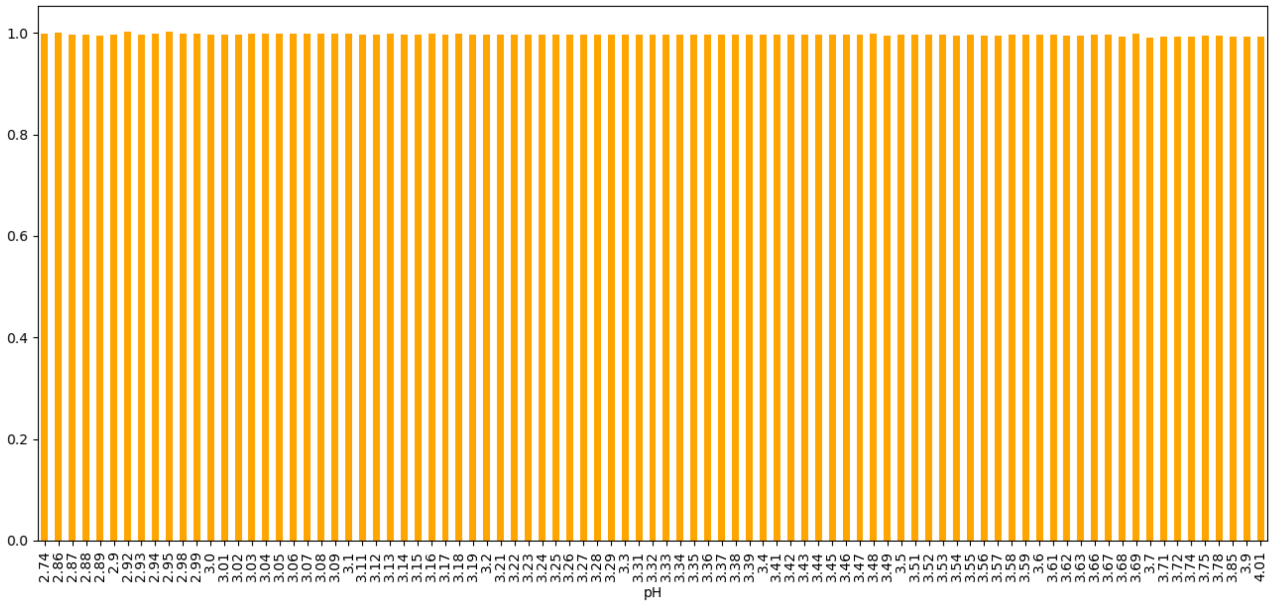

1.0
0.8
0.6
0.4
0.2
0.0

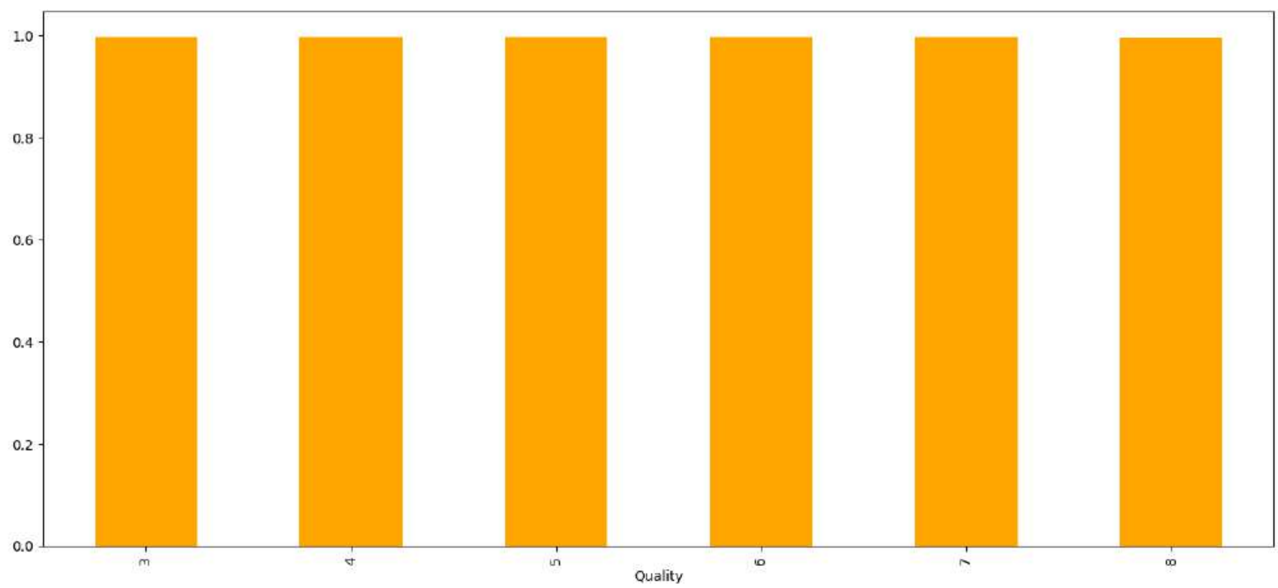

1.0
0.8
0.6
0.4
0.2
0.0
Free SO2



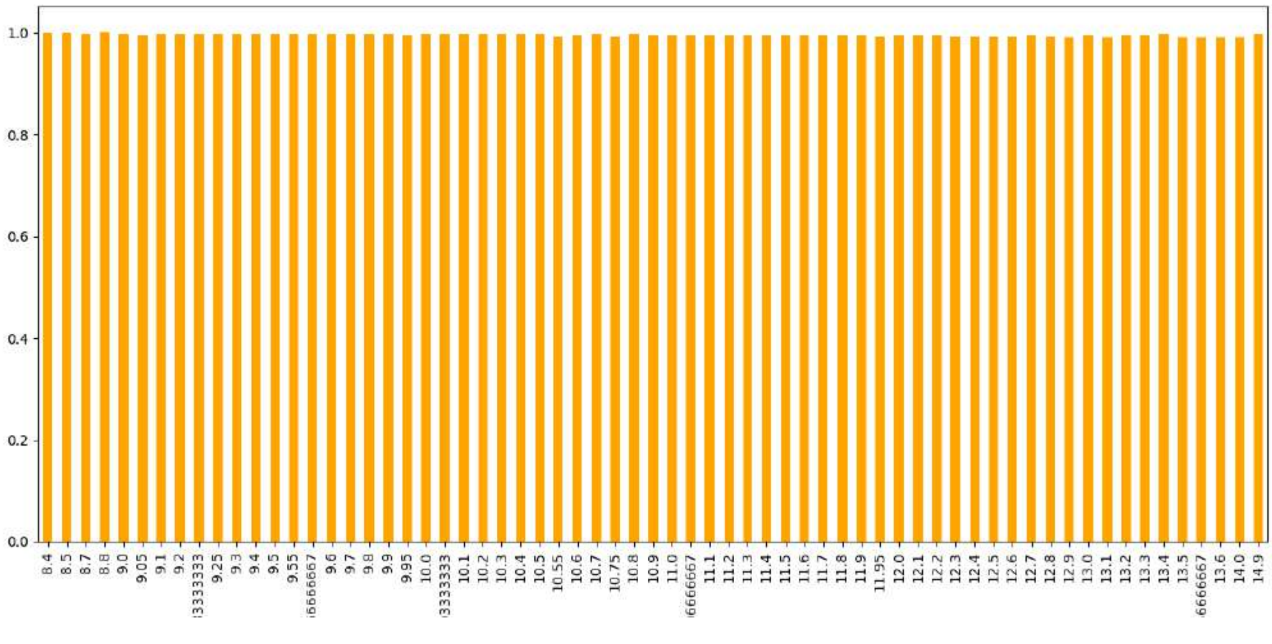
1.0
0.8
0.6
0.4
0.2
0.0
Total SO2


Sulphates

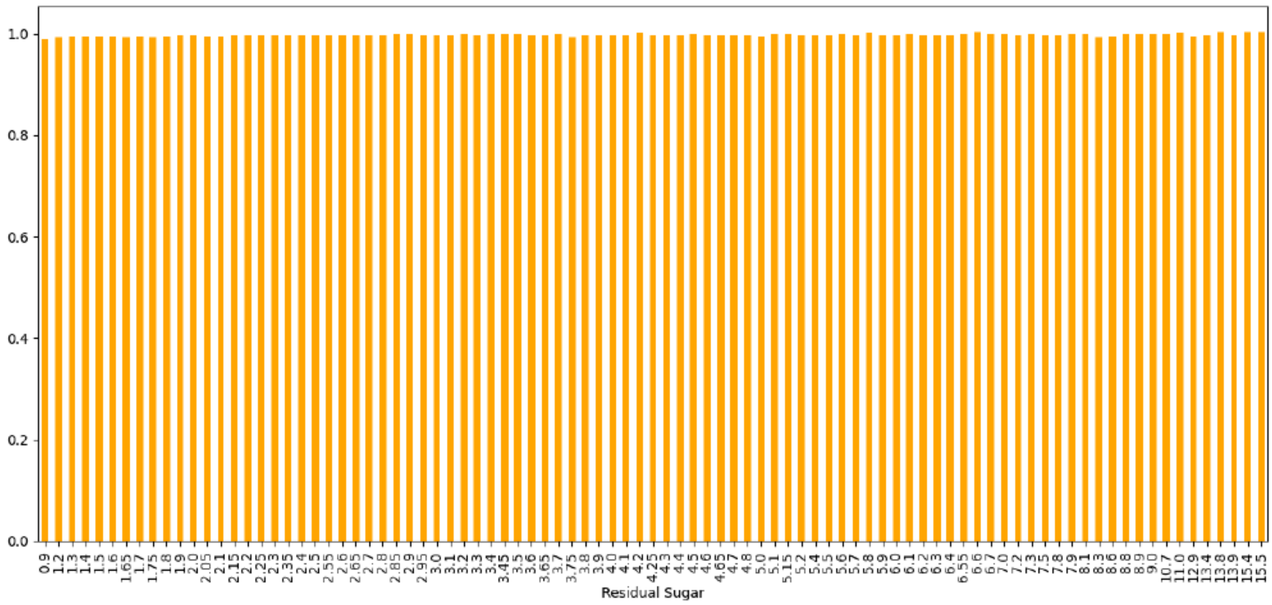

pH



1.0
0.8
0.6
0.4
0.2
0.0
3
4
5
6
7
8
Quality



1.0
0.8
0.6
0.4
0.2
0.0
8.4
8.5
8.7
8.8
9.0
9.05
9.1
9.2
3333333
9.25
9.3
9.4
9.5
9.55
6666667
9.6
9.7
9.8
9.9
9.95
10.0
3333333
10.1
10.2
10.3
10.4
10.5
10.55
10.6
10.7
10.75
10.8
10.9
11.0
6666667
11.1
11.2
11.3
11.4
11.5
11.6
11.7
11.8
11.9
11.95
12.0
12.1
12.2
12.3
12.4
12.5
12.6
12.7
12.8
12.9
13.0
13.1
13.2
13.3
13.4
13.5
6666667
13.6
14.0
14.9



1.0
0.8
0.6
0.4
0.2
0.0
0.9 1.2 1.3 1.4 1.5 1.6 1.65 1.7 1.75 1.8 1.9 2.0 2.05 2.1 2.15 2.2 2.25 2.3 2.35 2.4 2.5 2.55 2.6 2.65 2.7 2.8 2.85 2.9 2.95 3.0 3.1 3.2 3.3 3.4 3.45 3.5 3.6 3.65 3.7 3.75 3.8 3.9 4.0 4.1 4.2 4.25 4.3 4.4 4.5 4.6 4.65 4.7 4.8 5.0 5.1 5.15 5.2 5.4 5.5 5.6 5.7 5.8 5.9 6.0 6.1 6.2 6.3 6.4 6.55 6.6 6.7 7.0 7.2 7.3 7.5 7.8 7.9 8.1 8.3 8.6 8.8 8.9 9.0 10.7 11.0 12.9 13.4 13.8 13.9 15.4 15.5
Residual Sugar

Data Cleaning

```
#Check if there are missing values in the dataset
df.isnull().sum().sum()
```

0]: 0

```
#Check if there are duplicate rows in the dataset
df.duplicated().sum()
```

1]: 240

```
#Removing duplicates from the dataset
df.drop_duplicates(keep="first",inplace=True)
```

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

3]: 0

```
#Check if duplicate rows have been removed successfully from the dataset
df.duplicated().sum()
```

4]: 0

```
#No. of rows in the dataset after cleaning
print(len(df.axes[0]))
```

```
1359
```

```python
#Statistics for all the dataset columns
df.describe()
```

	Fixed Acidity	Volatile Acidity	Citric Acid	Chlorides	Free SO2	Total SO2	Sulphates	pH	Quality	Alcohol	Residual Sugar
count	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000
mean	8.310596	0.529478	0.272333	0.088124	15.893304	46.825975	0.658705	3.309787	5.623252	10.432315	2.523400
std	1.736990	0.183031	0.195537	0.049377	10.447270	33.408946	0.170667	0.155036	0.823578	1.082065	1.352314
min	4.600000	0.120000	0.000000	0.012000	1.000000	6.000000	0.330000	2.740000	3.000000	8.400000	0.900000
25%	7.100000	0.390000	0.090000	0.070000	7.000000	22.000000	0.550000	3.210000	5.000000	9.500000	1.900000
50%	7.900000	0.520000	0.260000	0.079000	14.000000	38.000000	0.620000	3.310000	6.000000	10.200000	2.200000
75%	9.200000	0.640000	0.430000	0.091000	21.000000	63.000000	0.730000	3.400000	6.000000	11.100000	2.600000
max	15.900000	1.580000	1.000000	0.611000	72.000000	289.000000	2.000000	4.010000	8.000000	14.900000	15.500000

```python
#Variance
df.var()
```

```
Fixed Acidity          3.017134
Volatile Acidity       0.033500
Citric Acid            0.038235
Chlorides              0.002438
Free SO2             109.145456
Total SO2           1116.157653
Sulphates              0.029127
pH                     0.024036
Quality                0.678281
Alcohol                1.170866
Residual Sugar         1.828752
Density                0.000003
dtype: float64
```

```python
#Skewness
df.skew()
```

```
Fixed Acidity       0.941041
Volatile Acidity    0.729279
Citric Acid         0.312726
Chlorides           5.502487
Free SO2            1.226579
Total SO2           1.540368
Sulphates           2.406505
pH                  0.232032
Quality             0.192407
Alcohol             0.859841
Residual Sugar      4.548153
Density             0.044778
dtype: float64
```

```python
#Kurtosis
df.kurtosis()
```

```
Fixed Acidity        1.049673
Volatile Acidity     1.249243
Citric Acid         -0.788921
Chlorides           38.624653
Free SO2             1.892691
Total SO2            4.042257
Sulphates           11.102282
pH                   0.879790
Quality              0.340256
Alcohol              0.159739
Residual Sugar      29.364592
Density              0.830659
dtype: float64
```

Data Selection

```
#Column-wise correlation in the dataset
df.corr()
```

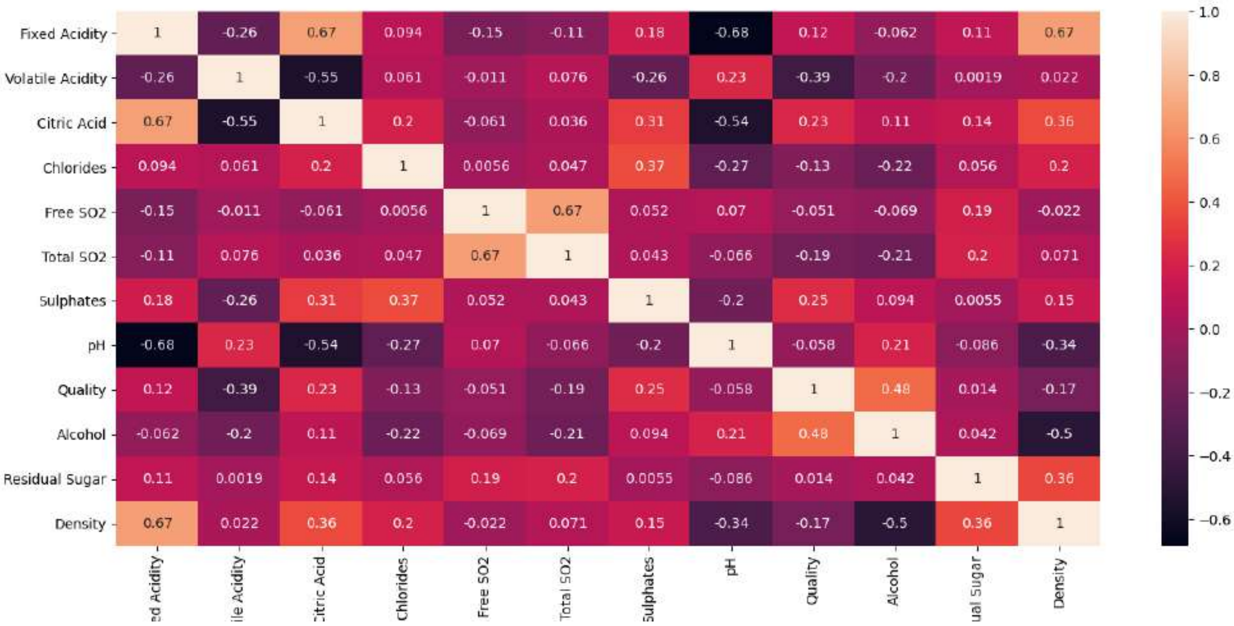
Fixed Acidity	1.000000	-0.255124	0.667437	0.085886	-0.140580	-0.103777	0.190269	-0.686685	0.119024	-0.061596	0.111025	0.670195
Volatile Acidity	-0.255124	1.000000	-0.551248	0.055154	-0.020945	0.071701	-0.256948	0.247111	-0.395214	-0.197812	-0.002449	0.023943
Citric Acid	0.667437	-0.551248	1.000000	0.210195	-0.048004	0.047358	0.326062	-0.550310	0.228057	0.105108	0.143892	0.357962
Chlorides	0.085886	0.055154	0.210195	1.000000	0.000749	0.045773	0.394557	-0.270893	-0.130988	-0.223824	0.026656	0.193592
Free SO2	-0.140580	-0.020945	-0.048004	0.000749	1.000000	0.667246	0.054126	0.056631	-0.050463	-0.080125	0.160527	-0.018071
Total SO2	-0.103777	0.071701	0.047358	0.045773	0.667246	1.000000	0.035291	-0.079257	-0.177855	-0.217829	0.201038	0.078141
Sulphates	0.190269	-0.256948	0.326062	0.394557	0.054126	0.035291	1.000000	-0.214134	0.248835	0.091621	-0.011837	0.146036
pH	-0.686685	0.247111	-0.550310	-0.270893	0.056631	-0.079257	-0.214134	1.000000	-0.055245	0.213418	-0.083143	-0.355617
Quality	0.119024	-0.395214	0.228057	-0.130988	-0.050463	-0.177855	0.248835	-0.055245	1.000000	0.480343	0.013640	-0.184252
Alcohol	-0.061596	-0.197812	0.105108	-0.223824	-0.080125	-0.217829	0.091621	0.213418	0.480343	1.000000	0.063281	-0.504995
Residual Sugar	0.111025	-0.002449	0.143892	0.026656	0.160527	0.201038	-0.011837	-0.083143	0.013640	0.063281	1.000000	0.324522
Density	0.670195	0.023943	0.357962	0.193592	-0.018071	0.078141	0.146036	-0.355617	-0.184252	-0.504995	0.324522	1.000000

```
#Import seaborn library
import seaborn as sns
```

Pearson correlation

```
sns.heatmap(df.corr('pearson'),annot=True)
```

<AxesSubplot:>



Fixed Acidity
Volatile Acidity
Citric Acid
Chlorides
Free SO2
Total SO2
Sulphates
pH
Quality
Alcohol
Residual Sugar
Density

```
All the three correlation methods namely Pearson, Spearman and Kendall give varying results.

Target column: Density

Pearson correlation results in:

Columns:
Citric Acid and Residual Sugar are similar
Chlorides and Sulphates are similar

Since,
Residual Sugar is more correlated to Density compared to Citric Acid
Sulphates is more correlated to Density compared to Chlorides

Thus, columns Citric Acid and Chlorides will be dropped to remove redundancy.
```

```python
df=df.drop(['Citric Acid','Chlorides'],axis=1)
```

```python
df.replace('', numpy.nan, inplace=True)
```

```python
df.dropna(inplace=True)
```

```python
df.head()
```

26]:

	Fixed Acidity	Volatile Acidity	Free SO2	Total SO2	Sulphates	pH	Quality	Alcohol	Residual Sugar	Density
0	7.4	0.70	11.0	34.0	0.56	3.51	5	9.4	1.9	0.9978
1	7.8	0.88	25.0	67.0	0.68	3.20	5	9.8	2.6	0.9968
2	7.8	0.76	15.0	54.0	0.65	3.26	5	9.8	2.3	0.9970
3	11.2	0.28	17.0	60.0	0.58	3.16	6	9.8	1.9	0.9980
5	7.4	0.66	13.0	40.0	0.56	3.51	5	9.4	1.8	0.9978

Data Splitting and Model Building (Multiple Linear regression)

```
X = df.iloc[:, :-1]
y = df['Density']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
from sklearn.linear_model import LinearRegression

regressor = LinearRegression()
regressor.fit(X, y)
```

```
9]: LinearRegression()
```

```
y_pred = regressor.predict(X_test)
```

```python
print(y_pred)
```

```
[0.99640749 0.99785688 0.99708326 0.99539482 0.99612931 0.995194
 0.99693205 0.99699212 0.99745241 0.99430827 0.99449966 0.99323472
 0.99681841 0.99762721 0.99377624 0.99446739 0.99703345 0.99789105
 1.00360707 0.99515538 0.9952441  0.99645341 0.99386123 0.99971813
 0.99422765 1.00023135 0.9959406  0.99459511 0.99593451 0.99851908
 0.99473729 0.99648162 0.99837395 0.99565292 0.9922853  0.99421875
 0.99670407 0.99772534 0.99660006 0.99771934 0.99495205 0.99941717
 0.99510624 0.99712425 0.99525415 0.99689882 0.99828792 0.99523136
 0.99586954 0.99575379 0.9965849  0.99792298 1.00428919 0.99582539
 0.99664799 0.99969216 0.99708493 0.99820925 0.996148   0.99343401
 0.99749114 0.99632346 0.99663821 0.99789704 0.99963601 0.99666104
 0.99459784 0.99562009 0.99480111 0.99460536 0.99692844 0.99979836
 0.99809219 0.99627198 0.99704802 0.99822532 0.99671455 0.99800649
 0.99470689 0.99266523 0.99804928 0.99635738 0.99508558 0.99578954
 0.99510302 0.999549   0.99710988 1.00018898 0.99540078 0.99582883
 0.99455876 0.99098862 0.99606226 0.99542813 0.99745843 0.9976648
 0.99617895 0.99815673 0.99712086 0.99466019 0.99293655 0.99917448
 0.99637922 0.99509361 1.00182092 0.99625012 0.99921633 0.99934813
 0.99979403 0.99580573 0.99991449 0.99755893 0.99690107 0.99902839
 0.99833173 0.99682315 0.99514039 0.99661044 0.99719153 0.99542585
 0.99859561 0.99782631 0.99708716 0.99593546 0.99746571 0.99787647
 0.99547383 0.9951078  0.99720052 0.99631605 0.9973698  0.99795241
 0.99606462 0.99811306 0.99480199 0.99382957 0.99818819 0.99522122
 0.99525686 0.99798227 0.9974018  0.99689847 1.00179894 0.99659585
 0.99663465 0.99725311 0.99496596 0.99756373 0.9986689  0.99395131
 0.99681282 0.99790575 0.99706939 0.99964701 0.99931397 0.99800097
 0.99149985 0.99938661 0.99512582 0.99496219 0.99627816 0.99851102
 0.9972537  0.99717722 0.99638492 0.99592906 0.99714893 0.99429821
 0.99594665 0.99903405 0.99736444 0.99769083 0.99652224 0.99658628
 0.99736231 0.99709652 0.99725698 0.99735359 0.99583705 0.99726851
 0.99541789 0.99833479 0.99576897 0.99790523 0.99854828 0.99751013
 0.99484604 0.9953428  0.99922858 0.99563559 0.99715141 0.99595074
 0.99479104 0.99539834 0.99675423 0.9960853  0.99364291 0.9971918
 0.995797   0.99537605 0.99660393 0.99698644 0.99467083 0.99742502
 0.99892243 0.99690542 0.9959947  0.99622429 0.996841   0.99646689
 0.99700943 0.99621766 0.99705581 0.99704641 0.99619398 0.99870503
 0.99787223 1.00102815 0.99460995 0.9949689  1.00036938 0.99644242
 0.99566    0.99770419 0.9934913  0.99442809 0.99743532 0.99968851
 0.99856396 0.9959146  0.99475071 0.99483642 0.99615518 0.99118962
 0.99971855 0.9981558  0.9974608  0.99915894 0.9961147  0.9964988
 0.9968256  0.99480935 0.9982535  0.99894865 0.99518061 0.99838636
 0.99572676 0.9973213  0.99784747 1.00168698 0.99683371 0.99854839
 0.99748425 0.99821544 0.99547791 0.99806656 0.99559497 0.99723412
 0.9954647  0.99688276 0.99541718 0.99431642 0.99436769 0.99815486
 0.99675343 0.99722784 0.99404113 0.99664855 1.00109986 0.99882043
 0.9985345  0.99409147]
```

```
print(y_test)
```

```
1133    0.99684
760     0.99779
1173    0.99716
1079    0.99316
935     0.99543
         ...
754     0.99656
289     1.00100
684     0.99800
772     0.99782
1272    0.99385
Name: Density, Length: 272, dtype: float64
```

```
regressor.score(X_test, y_test)
```

```
0.8392217121633154
```

```
#Co-efficients of the Logistic regression equation
regressor.coef_
```

```
array([ 9.33624238e-04,  6.91164973e-04, -7.48772624e-06,  2.09753090e-06,
        1.54531658e-03,  4.81418488e-03, -2.65303379e-05, -9.40764728e-04,
        4.10503352e-04])
```

```
#y-intercept of the Logistic regression equation
regressor.intercept_
```

```
0.9805806482257146
```

```
Multiple Linear regression equation using Pearson

y -> target variable i.e. Density
a -> y-intercept of Density
b0 -> co-efficient of Fixed Acidity
b1 -> co-efficient of Volatile Acidity
b2 -> co-efficient of Free SO2
b3 -> co-efficient of Total SO2
b4 -> co-efficient of Sulphates
b5 -> co-efficient of pH
b6 -> co-efficient of Quality
b7 -> co-efficient of Alcohol
b8 -> co-efficient of Residual Sugar

General equation: y = a + b0x0 + b1x1 + ... + bnxn

Actual equation: Density = 0.98 + 0.17(Fixed Acidity) + 0.12(Volatile Acidity) - 0.01(Free SO2) + 0.004(Total SO2) +
                           0.08(Sulphates) + 0.24(pH) - 0.02(Quality) - 0.20(Alcohol) + 0.07(Residual Sugar)
```

Model evaluation through k-fold cross validation and evaluation metrics

```
#K-fold cross-validation
#Logistic Regression
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
k = 5
kf = model_selection.KFold(n_splits=k, random_state=None)
model = LinearRegression()
result = cross_val_score(model , X, y, cv = kf)
print("Avg accuracy: {}".format(result.mean()))
```

```
Avg accuracy: 0.7519965612347168
```

```
#Root mean square error
import sklearn
sklearn.metrics.mean_squared_error(y_test,y_pred)
```

7]: 6.726161410167201e-07

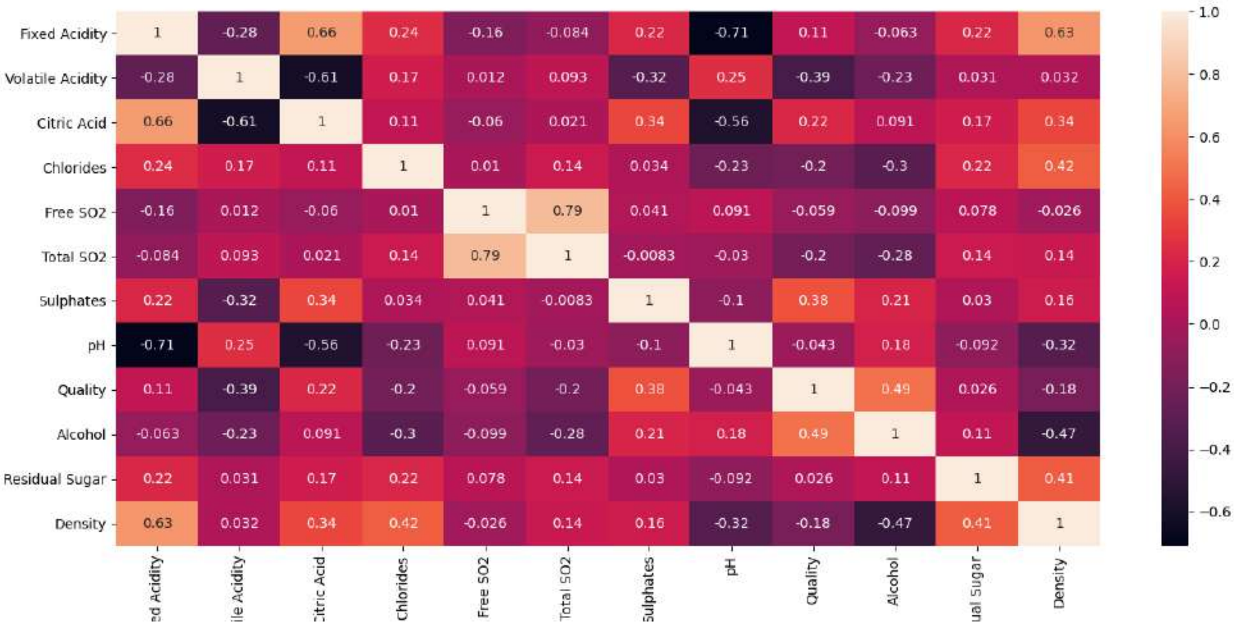
```
#R2 score
import sklearn
sklearn.metrics.r2_score(y_test,y_pred)
```

8]: 0.8392217121633154

Spearman correlation

```
sns.heatmap(df.corr('spearman'),annot=True)
```

```
<AxesSubplot:>
```


Fixed Acidity
Volatile Acidity
Citric Acid
Chlorides
Free SO2
Total SO2
Sulphates
pH
Quality
Alcohol
Residual Sugar
Density
1 -0.28 0.66 0.24 -0.16 -0.084 0.22 -0.71 0.11 -0.063 0.22 0.63
-0.28 1 -0.61 0.17 0.012 0.093 -0.32 0.25 -0.39 -0.23 0.031 0.032
0.66 -0.61 1 0.11 -0.06 0.021 0.34 -0.56 0.22 0.091 0.17 0.34
0.24 0.17 0.11 1 0.01 0.14 0.034 -0.23 -0.2 -0.3 0.22 0.42
-0.16 0.012 -0.06 0.01 1 0.79 0.041 0.091 -0.059 -0.099 0.078 -0.026
-0.084 0.093 0.021 0.14 0.79 1 -0.0083 -0.03 -0.2 -0.28 0.14 0.14
0.22 -0.32 0.34 0.034 0.041 -0.0083 1 -0.1 0.38 0.21 0.03 0.16
-0.71 0.25 -0.56 -0.23 0.091 -0.03 -0.1 1 -0.043 0.18 -0.092 -0.32
0.11 -0.39 0.22 -0.2 -0.059 -0.2 0.38 -0.043 1 0.49 0.026 -0.18
-0.063 -0.23 0.091 -0.3 -0.099 -0.28 0.21 0.18 0.49 1 0.11 -0.47
0.22 0.031 0.17 0.22 0.078 0.14 0.03 -0.092 0.026 0.11 1 0.41
0.63 0.032 0.34 0.42 -0.026 0.14 0.16 -0.32 -0.18 -0.47 0.41 1
ed Acidity
ile Acidity
Citric Acid
Chlorides
Free SO2
Total SO2
Sulphates
pH
Quality
Alcohol
ual Sugar
Density
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

```
All the three correlation methods namely Pearson, Spearman and Kendall give varying results.

Target column: Density

Spearman correlation results in:

Columns:
Chlorides and Residual Sugar are similar
Total SO2 and Sulphates are similar

Since,
Residual Sugar is more correlated to Density compared to Chlorides
Sulphates is more correlated to Density compared to Total SO2

Thus, columns Chlorides and Total SO2 will be dropped to remove redundancy.
```

```python
df=df.drop(['Chlorides','Total SO2'],axis=1)
```

```python
df.replace('', numpy.nan, inplace=True)
```

```python
df.dropna(inplace=True)
```

```python
df.head()
```

99]:

	Fixed Acidity	Volatile Acidity	Citric Acid	Free SO2	Sulphates	pH	Quality	Alcohol	Residual Sugar	Density
0	7.4	0.70	0.00	11.0	0.56	3.51	5	9.4	1.9	0.9978
1	7.8	0.88	0.00	25.0	0.68	3.20	5	9.8	2.6	0.9968
2	7.8	0.76	0.04	15.0	0.65	3.26	5	9.8	2.3	0.9970
3	11.2	0.28	0.56	17.0	0.58	3.16	6	9.8	1.9	0.9980
5	7.4	0.66	0.00	13.0	0.56	3.51	5	9.4	1.8	0.9978

Data Splitting and Model Building (Multiple Linear regression)

```python
X = df.iloc[:, :-1]
y = df['Density']
```

```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```python
from sklearn.linear_model import LinearRegression

regressor = LinearRegression()
regressor.fit(X, y)
```

2]: LinearRegression()

```python
y_pred = regressor.predict(X_test)
```

```
print(y_pred)
```

```
[0.99641235 0.99771424 0.99708552 0.99507838 0.99605905 0.99521188
 0.99695708 0.99702597 0.99747365 0.99431458 0.99450555 0.99322787
 0.99679304 0.9976465  0.99382576 0.99449355 0.99705424 0.99789065
 1.00374857 0.99509661 0.99524284 0.99639808 0.99386617 0.9997066
 0.9942679  1.00013511 0.99594241 0.99461485 0.99591618 0.99840809
 0.9947532  0.99649293 0.99833489 0.99571622 0.99213626 0.99428812
 0.99673519 0.99760788 0.99661742 0.99755979 0.99502285 0.9994279
 0.99511761 0.99714569 0.9953156  0.99696282 0.99833023 0.99525282
 0.99588651 0.99575115 0.99660022 0.99780053 1.00433185 0.99584864
 0.99666516 0.99959372 0.99707077 0.99825498 0.99613112 0.99352637
 0.99752508 0.9963029  0.99667853 0.99792644 0.99958575 0.99661786
 0.99462063 0.99565943 0.99473309 0.99456179 0.99691436 0.99975875
 0.99800438 0.99629402 0.99708326 0.99829789 0.99671322 0.99798456
 0.99466909 0.99253419 0.99804141 0.99640068 0.9950955  0.99583849
 0.9951192  0.99955604 0.99713475 1.00017949 0.99540457 0.99582806
 0.99459548 0.99093224 0.99611701 0.99548155 0.99743585 0.99771486
 0.9961739  0.99818404 0.99712553 0.99472399 0.99291189 0.99915891
 0.99634119 0.99514754 1.00180767 0.99629169 0.99918716 0.99934481
 0.99984419 0.99582763 0.99991188 0.99753935 0.99682846 0.99904088
 0.99833761 0.9968467  0.99519513 0.9966319  0.99723398 0.9954188
 0.99853973 0.99785595 0.99704325 0.99595258 0.99747564 0.99790455
 0.99552961 0.99508546 0.99715792 0.99633493 0.9974051  0.99798007
 0.99605416 0.9981362  0.99485498 0.99380895 0.99819151 0.99528467
 0.99524336 0.99801965 0.99742529 0.99688674 1.00183541 0.99659777
 0.99661135 0.99724929 0.99498    0.99759726 0.99869519 0.99392222
 0.99680942 0.99786009 0.9970171  0.99961965 0.99934062 0.99800614
 0.99140237 0.99936423 0.99514558 0.99498393 0.99629203 0.99850996
 0.99732606 0.99714463 0.99636438 0.99592532 0.99714559 0.99427042
 0.99594544 0.99905492 0.99732942 0.99769481 0.99654037 0.99661967
 0.99737955 0.9969948  0.99728689 0.99735945 0.9958051  0.99721152
 0.99507838 0.99824383 0.99583501 0.99779206 0.99855179 0.99734141
 0.99487671 0.9953734  0.99922877 0.99567449 0.99717774 0.99595574
 0.99481893 0.99545046 0.99673326 0.9961201  0.99362737 0.99717165
 0.99584558 0.9953814  0.99662423 0.99706113 0.99469025 0.99735623
 0.99895105 0.9969199  0.99601765 0.99626023 0.99684438 0.99647763
 0.99701743 0.9962448  0.99701157 0.99705229 0.99620043 0.99871637
 0.99779817 1.00101103 0.99460418 0.99496085 1.00039614 0.99647236
 0.9955972  0.99769237 0.99346898 0.99445351 0.99736843 0.99975078
 0.99856763 0.99593527 0.99476987 0.99479839 0.99609137 0.99118894
 0.99972328 0.99817038 0.99744093 0.9991811  0.99605519 0.99652116
 0.99684707 0.99483439 0.99825231 0.99890373 0.9951237  0.99835944
 0.99574311 0.99730916 0.99780249 1.00169162 0.99682037 0.99850708
 0.99749047 0.99823358 0.99545007 0.99803237 0.99564849 0.99718571
 0.99551223 0.99694067 0.99545902 0.99432574 0.99436096 0.99820026
 0.99679308 0.99728158 0.99404541 0.99676714 1.00106936 0.99872497
 0.99836034 0.99410029]
```

```python
print(y_test)
```

```
1133    0.99684
760     0.99779
1173    0.99716
1079    0.99316
935     0.99543
         ...
754     0.99656
289     1.00100
684     0.99800
772     0.99782
1272    0.99385
Name: Density, Length: 272, dtype: float64
```

```python
regressor.score(X_test, y_test)
```

```
0.8413634711146213
```

```python
#Co-efficients of the Logistic regression equation
regressor.coef_
```

```
array([ 9.17653833e-04,  7.85180479e-04,  1.85308769e-04, -3.29203335e-06,
        1.53263542e-03,  4.75313662e-03, -3.08249199e-05, -9.50242204e-04,
        4.14085650e-04])
```

```python
#y-intercept of the Logistic regression equation
regressor.intercept_
```

```
0.9809690544230507
```

```
Multiple Linear regression equation using Pearson

y -> target variable i.e. Density
a -> y-intercept of Density
b0 -> co-efficient of Fixed Acidity
b1 -> co-efficient of Volatile Acidity
b2 -> co-efficient of Citric Acid
b3 -> co-efficient of Free SO2
b4 -> co-efficient of Sulphates
b5 -> co-efficient of pH
b6 -> co-efficient of Quality
b7 -> co-efficient of Alcohol
b8 -> co-efficient of Residual Sugar

General equation: y = a + b0x0 + b1x1 + ... + bnxn

Actual equation: Density = 0.98 + 0.17(Fixed Acidity) + 0.14(Volatile Acidity) + 0.03(Citric Acid) - 0.006(Free SO2) +
                           0.08(Sulphates) + 0.24(pH) - 0.02(Quality) - 0.17(Alcohol) + 0.07(Residual Sugar)
```

Model evaluation through k-fold cross validation and evaluation metrics

```
#K-fold cross-validation
#Logistic Regression
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
k = 5
kf = model_selection.KFold(n_splits=k, random_state=None)
model = LinearRegression()
result = cross_val_score(model , X, y, cv = kf)
print("Avg accuracy: {}".format(result.mean()))
```

```
Avg accuracy: 0.7526190991866091
```

```
#Root mean square error
import sklearn
sklearn.metrics.mean_squared_error(y_test,y_pred)
```

3]: 6.636560901280165e-07

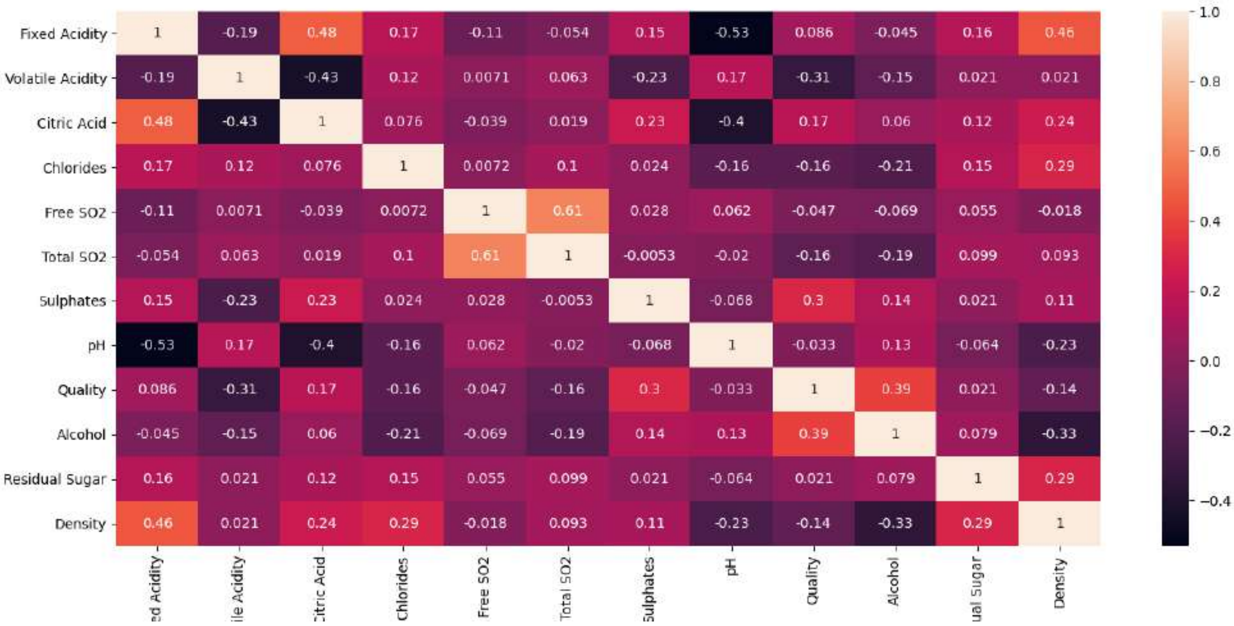
```
#R2 score
import sklearn
sklearn.metrics.r2_score(y_test,y_pred)
```

4]: 0.8413634711146213

Kendall correlation

```
sns.heatmap(df.corr('kendall'),annot=True)
```

```
<AxesSubplot:>
```



Fixed Acidity
Volatile Acidity
Citric Acid
Chlorides
Free SO2
Total SO2
Sulphates
pH
Quality
Alcohol
Residual Sugar
Density
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

```
All the three correlation methods namely Pearson, Spearman and Kendall give varying results.

Target column: Density

Spearman correlation results in:

Columns:
Chlorides and Citric Acid are similar
Residual Sugar and Citric Acid are similar

Since,
Chlorides is more correlated to Density compared to Citric Acid
Residual Sugar is more correlated to Density compared to Citric Acid

Thus, columns Chlorides and Total SO2 will be dropped to remove redundancy.
```

```
df=df.drop(['Citric Acid'],axis=1)
```

```
df.replace('', numpy.nan, inplace=True)
```

```
df.dropna(inplace=True)
```

```
df.head()
```

28]:

	Fixed Acidity	Volatile Acidity	Chlorides	Free SO2	Total SO2	Sulphates	pH	Quality	Alcohol	Residual Sugar	Density
0	7.4	0.70	0.076	11.0	34.0	0.56	3.51	5	9.4	1.9	0.9978
1	7.8	0.88	0.098	25.0	67.0	0.68	3.20	5	9.8	2.6	0.9968
2	7.8	0.76	0.092	15.0	54.0	0.65	3.26	5	9.8	2.3	0.9970
3	11.2	0.28	0.075	17.0	60.0	0.58	3.16	6	9.8	1.9	0.9980
5	7.4	0.66	0.075	13.0	40.0	0.56	3.51	5	9.4	1.8	0.9978

Data Splitting and Model Building (Multiple Linear regression)

```
X = df.iloc[:, :-1]
y = df['Density']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
from sklearn.linear_model import LinearRegression

regressor = LinearRegression()
regressor.fit(X, y)
```

```
1]: LinearRegression()
```

```
y_pred = regressor.predict(X_test)
```

```
print(y_pred)
```

```
[0.99641881 0.99783831 0.99710946 0.99539333 0.99612633 0.99514071
 0.99696633 0.99693059 0.9974363  0.99430623 0.99452827 0.99323093
 0.99682425 0.9975929  0.99371917 0.99446207 0.99690995 0.99780184
 1.00372893 0.99512877 0.99522288 0.99648529 0.99385153 0.99962917
 0.99429644 1.00026037 0.99593071 0.99459703 0.99587082 0.99852296
 0.99476504 0.99645932 0.99836694 0.99569062 0.99230285 0.99418153
 0.9966547  0.9976377  0.9965427  0.99738016 0.99496675 0.99947788
 0.99505385 0.99704807 0.99532533 0.99693783 0.99832348 0.99527203
 0.99588176 0.99579178 0.99663833 0.99789525 1.00417565 0.99576174
 0.99671826 0.99975903 0.99708058 0.99821356 0.99617422 0.99339839
 0.99748019 0.99627904 0.99656538 0.9979009  0.99969523 0.99664732
 0.99457056 0.99558306 0.9947741  0.99477641 0.996942   0.99975734
 0.99810048 0.9962019  0.99697584 0.99814082 0.99674287 0.99800499
 0.99478638 0.99267271 0.99799313 0.99639011 0.99501187 0.9957357
 0.99509539 0.99951169 0.99703373 1.00018688 0.99543532 0.99584553
 0.99453578 0.99096125 0.99606316 0.9954024  0.99745739 0.99762138
 0.99614044 0.99811581 0.99708983 0.99463245 0.99291815 0.9991374
 0.99635355 0.99508212 1.00179311 0.9961983  0.99922763 0.99934556
 0.99985347 0.99579963 0.99986151 0.99757961 0.99689351 0.99900495
 0.99827752 0.99660504 0.99513862 0.99651417 0.99723145 0.99539154
 0.99863333 0.99787641 0.99707577 0.99597813 0.99746312 0.9978509
 0.99544586 0.99509646 0.99720654 0.99627363 0.99727344 0.99792936
 0.99599941 0.99830583 0.99475572 0.99390502 0.99814273 0.99527933
 0.99520258 0.99797624 0.99720649 0.99689825 1.00170836 0.99654804
 0.99664019 0.99722105 0.99502864 0.9975619  0.99890512 0.99384744
 0.99683127 0.99811971 0.99703394 0.9997296  0.99935193 0.99803122
 0.99148024 0.99932927 0.99514705 0.99498988 0.99627844 0.99853014
 0.99763313 0.99719524 0.99642448 0.99590647 0.99713768 0.99428361
 0.99594901 0.99893899 0.99751312 0.99769788 0.9965227  0.99659598
 0.99735886 0.99710652 0.99733303 0.9973305  0.99583986 0.99733397
 0.99541931 0.99833939 0.9957313  0.99787645 0.998505   0.99719595
 0.99480345 0.99532519 0.99922576 0.99562084 0.99710022 0.9959757
 0.9948505  0.99569124 0.99675188 0.99605874 0.99360189 0.99721369
 0.99581908 0.99538559 0.99654692 0.9974942  0.99470794 0.99746762
 0.99888046 0.99690956 0.99593264 0.99613939 0.99681007 0.99641549
 0.99701326 0.99611411 0.99704482 0.9971057  0.99620405 0.99870852
 0.99779887 1.00087756 0.99456428 0.99501589 1.00031693 0.99642723
 0.99566825 0.99774755 0.99344493 0.99441864 0.99741822 0.99964968
 0.99852047 0.99587051 0.99470922 0.99484526 0.99616308 0.99121301
 0.99972016 0.99817247 0.99742794 0.9991201  0.99611154 0.99650855
 0.9967816  0.99481845 0.99826466 0.99907847 0.99516158 0.99840326
 0.9957903  0.99733968 0.99781513 1.00165794 0.99683321 0.99857616
 0.99749346 0.99821482 0.99548914 0.99802023 0.99555665 0.99726387
 0.99538619 0.99681787 0.99532774 0.99436838 0.99434112 0.99812672
 0.9967118  0.99717227 0.99401908 0.99711037 1.00106611 0.99882585
 0.99855554 0.99411842]
```

```python
print(y_test)
```

```
1133    0.99684
760     0.99779
1173    0.99716
1079    0.99316
935     0.99543
          ...
754     0.99656
289     1.00100
684     0.99800
772     0.99782
1272    0.99385
Name: Density, Length: 272, dtype: float64
```

```python
regressor.score(X_test, y_test)
```

```
0.841340122483591
```

```python
#Co-efficients of the Logistic regression equation
regressor.coef_
```

```
array([ 9.44052119e-04,  6.28191476e-04,  1.88002896e-03, -7.54213082e-06,
        2.36173937e-06,  1.30295076e-03,  5.00615623e-03, -1.11454645e-05,
       -9.28591371e-04,  4.06644422e-04])
```

```python
#y-intercept of the Logistic regression equation
regressor.intercept_
```

```
0.9796706388896447
```

```
Multiple Linear regression equation using Pearson

y -> target variable i.e. Density
a -> y-intercept of Density
b0 -> co-efficient of Fixed Acidity
b1 -> co-efficient of Volatile Acidity
b2 -> co-efficient of Chlorides
b3 -> co-efficient of Free SO2
b4 -> co-efficient of Total SO2
b5 -> co-efficient of Sulphates
b6 -> co-efficient of pH
b7 -> co-efficient of Quality
b8 -> co-efficient of Alcohol
b9 -> co-efficient of Residual Sugar

General equation: y = a + b0x0 + b1x1 + ... + bnxn

Actual equation: Density = 0.98 + 0.17(Fixed Acidity) + 0.11(Volatile Acidity) + 0.08(Chlorides) - 0.02(Free SO2) +
                          0.005(Total SO2) + 0.07(Sulphates) + 0.25(pH) - 0.008(Quality) - 0.18(Alcohol) + 0.07(Residual Sugar)
```

Model evaluation through k-fold cross validation and evaluation metrics

```
#K-fold cross-validation
#Logistic Regression
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
k = 5
kf = model_selection.KFold(n_splits=k, random_state=None)
model = LinearRegression()
result = cross_val_score(model , X, y, cv = kf)
print("Avg accuracy: {}".format(result.mean()))
```

```
Avg accuracy: 0.7515064948161496
```

```
#Root mean square error
import sklearn
sklearn.metrics.mean_squared_error(y_test,y_pred)
```

```
9]: 6.637537691511789e-07
```

```
#R2 score
import sklearn
sklearn.metrics.r2_score(y_test,y_pred)
```

```
0]: 0.841340122483591
```

```
Comparing the accuracy given by the three methods i.e. Pearson, Spearman and Kendall through Multiple Linear regression using
sklearn:

Pearson: 0.8392217121633154
Spearman: 0.8413634711146213
Kendall: 0.841340122483591

Thus, Kendall correlation along with Logistic Regression model gives the most accurate results of the three on evaluation.
```

```
Comparing the average accuracy given by the three methods i.e. Pearson, Spearman and Kendall through K-fold cross validation:

Pearson: 0.7519965612347168
Spearman: 0.7526190991866091
Kendall: 0.7515064948161496

Thus, Spearman correlation along with Logistic Regression model gives the most accurate results of the three on evaluation.
```

```
Comparing the RMSE (Root Mean Squar Error) for the three methods i.e. Pearson, Spearman and Kendall:

Pearson: 6.726161410167201e-07 ~ 0.006
Spearman: 6.636560901280165e-07 ~ 0.006
Kendall: 6.637537691511789e-07 ~ 0.006

Thus, Pearson correlation gives the least RMSE evaluation metric.
```

```
Comparing the R2 score for the three methods i.e. Pearson, Spearman and Kendall:

Pearson: 0.8392217121633154
Spearman: 0.8413634711146213
Kendall: 0.841340122483591

Thus, Kendall correlation gives the best R2 score evaluation metric.
```