In [160]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [22]:
```
df = pd.read_csv("winequality-red.csv")
df.head()
```

Out[22]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
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

In [23]:
```
# upto here we are uploded "wine quality-red.csv" to jupyter notebook.
# and make df as a instance of our wine dataset.
```

In [24]:
```
df.shape
```

Out[24]:
```
(1599, 12)
```

In [25]:
```
# here above we can see that the shape of dataset is 1599 ROWS & 12 COLUMNS are
```

In [26]:
```
df.columns
```

Out[26]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [27]:
```
#  here we can see the the 12 diffrent column names present in the dataset.
```

In [28]:
```
df.dtypes
```

Out[28]:
```
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

In [29]:
```
# as above we can see that the data type of each columns , all columns having
```

In [30]:
```
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
```

In [31]:
```
# here we used ".info" which provides us the maximum information about the dat
# total no.of rows present = 1599
# No NULL vales are present in the dataset.
# total no. of columns present = 12
# the datatypes of all the columns
# memory usage
```

In [32]:
```
df.describe()
```

Out[32]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulf dioxic
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.0000(
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.4677(
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.8953(
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.0000(
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.0000(
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.0000(
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.0000(
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.0000(

In [ ]:
```
'''''' ".describe" is a very important function which gives a lot of following

 1. here we can see the total no. counts present in each columns, i.e [1599] a
    means there are no missing values in the given dataset.
 2. It also shows the MEAN, STANDARD DEVIATION of each column.
 3. It also provides us the MIN, MAXIMUM, 25%, 50% (median), 75% Values of eve
 4. IMP POINT:- we know that IF THE MEAN > MEDIAN = data is Right Skewed
                        and  IF THE MEAN < median = data is left Skewed, so we
            (a) "fixed acidity" = mean > median ( right skewed)
            (b) "free sulfurdioxide" = mean > median (right skewed)
            (c) "total sulfurdioxide" = mean > median (right skewed)
            the rest of the columns are having very slight difference.
 5. IMP POINT :- here we also determines the "STANDARD DEVIATION" -
            so we can see that the "std" of "free sulfurdioxide" & " total sul
            so from this we can conclude that there are some OUTLIERS are may
 6. As we can also see that the difference between "75 Percentile & Maximum" i
            so from this also we can assume that there maybe OUTLIERS are pres
    ''''
```

In [36]:
```
# as from the data set we can identifies that the column "quality" is our targe
# so now we are doing some analysis of the Target Column
```

In [37]:
```
df.quality.unique()
```

Out[37]:
```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

In [ ]:
```
# as we can see that the "quality" column is categorical in form having unique
```

In [38]:
```
df.quality.nunique()
```

Out[38]:
```
6
```

In [39]:
```
# and it consist of 6 unique values.
```

In [46]:
```
df.quality.value_counts()
```

Out[46]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

In [47]:
```
# from the above we can identifies that which value is having howmuch count in
# and we can see that the average quantity is present in 5,6,& 7 values.
```

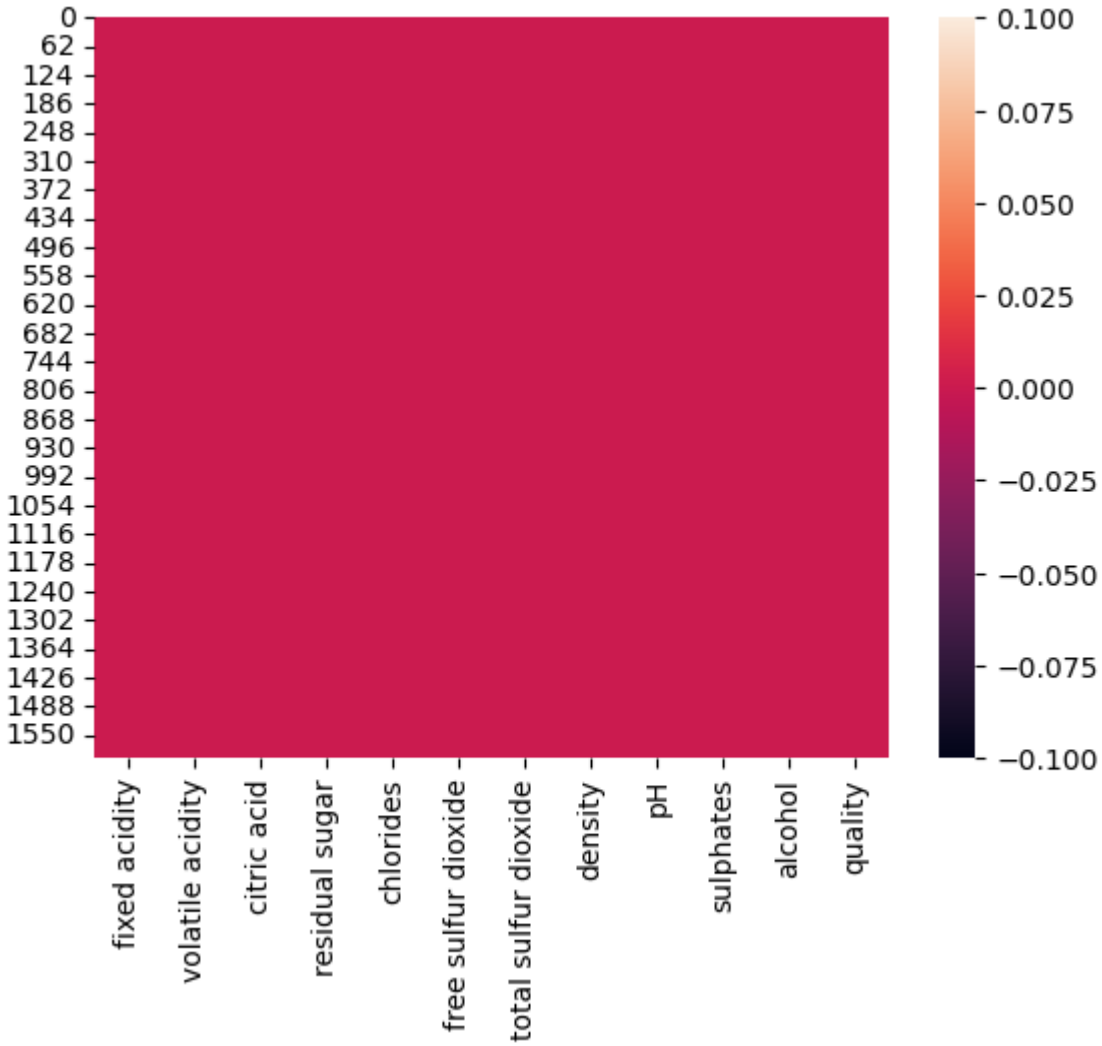
In [48]:
```
# CHECKING NULL VALUES
```

In [49]:
```
df.isnull().sum()
```

Out[49]:
```
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
```

In [50]:
```
sns.heatmap(df.isnull())
```

Out[50]: <AxesSubplot:>



In [51]:
```
# As with the help of numerical and graphical both types we can see that there
```

In [52]:
```
# CORRELATION
# Now here we have to check the correlation between the every column
#  and for this we are using both NON-GRAPHICAL & GRAPHICAL ANALYSIS :-
```

In [53]:
```
dfcor = df.corr()
dfcor
```

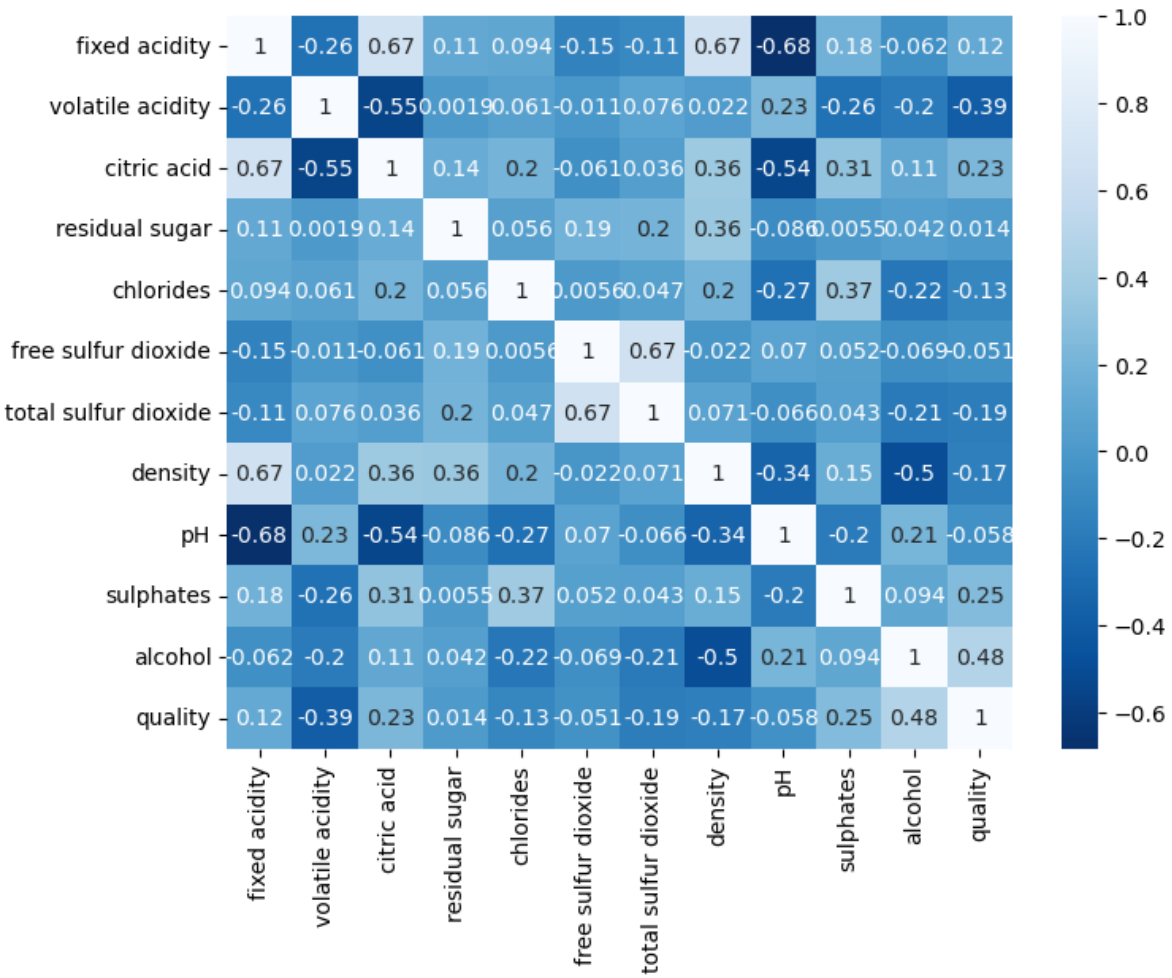
Out[53]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density
fixed acidity	1.000000	-0.256131	0.671703	0.114777	0.093705	-0.153794	-0.113181	0.668047
volatile acidity	-0.256131	1.000000	-0.552496	0.001918	0.061298	-0.010504	0.076470	0.022026
citric acid	0.671703	-0.552496	1.000000	0.143577	0.203823	-0.060978	0.035533	0.364947
residual sugar	0.114777	0.001918	0.143577	1.000000	0.055610	0.187049	0.203028	0.355283
chlorides	0.093705	0.061298	0.203823	0.055610	1.000000	0.005562	0.047400	0.200632
free sulfur dioxide	-0.153794	-0.010504	-0.060978	0.187049	0.005562	1.000000	0.667666	-0.021946
total sulfur dioxide	-0.113181	0.076470	0.035533	0.203028	0.047400	0.667666	1.000000	0.071269
density	0.668047	0.022026	0.364947	0.355283	0.200632	-0.021946	0.071269	1.000000
pH	-0.682978	0.234937	-0.541904	-0.085652	-0.265026	0.070377	-0.066495	-0.341699
sulphates	0.183006	-0.260987	0.312770	0.005527	0.371260	0.051658	0.042947	0.148506
alcohol	-0.061668	-0.202288	0.109903	0.042075	-0.221141	-0.069408	-0.205654	-0.496180
quality	0.124052	-0.390558	0.226373	0.013732	-0.128907	-0.050656	-0.185100	-0.174919

In [54]:
```
#  as above we can see that the +1 = is for highly positively correlation and
#                             -1 = is for highly negative correlation
# as it is difficult to identify the correlation between the data by NON-GRAPHI
# so we are also finds correlation by using GRAPHICAL REPRESENTATION (HEATMAP)
```

In [61]:
```
plt.figure(figsize=(8,6))
sns.heatmap(dfcor,annot=True,cmap="Blues_r")
```

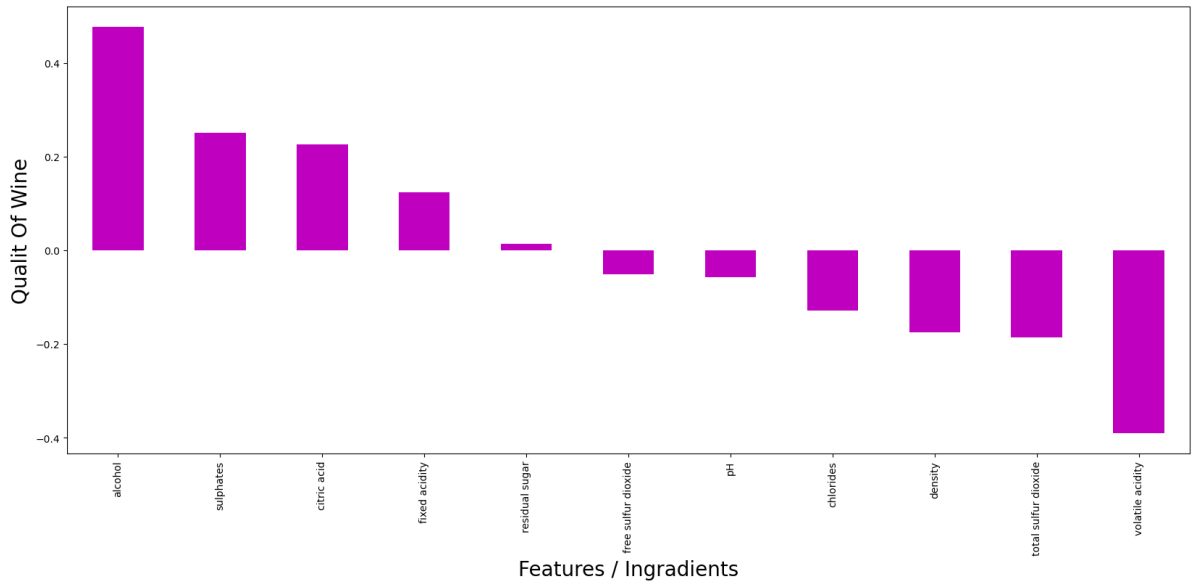
Out[61]: <AxesSubplot:>



In [ ]:
```
 '''' as from above now it is easy to identify the highly positive and negative
1. the colour from dark blue to white is showing the incresing the (+)ve correl
2. the dark blue colour is showing the highly (-)ve correlation.
3. As from the heatmap we can finds that the correlation between "free sulfurd
4. we can also finds that the "alcohol" & "quality" is also (+)tively correlate
5. "citric acid" + "density" is also (+)vly correlated with "fixed acidity"
   weather "ph" is (-)vely correlated with "fixed acidity" ''''
```

In [63]:
```
# FINDING CORRELATION BETWEEN TARGET & FEATURES
```

In [74]:
```
plt.figure(figsize=(20,8))
df.corr()['quality'].sort_values(ascending=False).drop(['quality']).plot(kind=
plt.xlabel('Features / Ingradients',fontsize=20)
plt.ylabel('Qualit Of Wine',fontsize =20)
plt.title=("Wine Quality")
plt.show()
```



In [ ]:
```
#  here from the above we can see that the "alcohol" is highly positive correl
#  and the "volatile acidity" is highly negative correlated with the "quality"
# "residual suagr", "free sulphur dioxide" and "ph" are having 'Neutral Relati
```

In [ ]:
```
#  Now as earlier we can seen that there is a highly probability of OUTLIERS pr
#  so now here we are identify and conforming that weather the outliers are pre
#  CHECKING OUTLIERS BY USING BOXPLOT METHOD
```

In [75]:
```
df.describe()
```

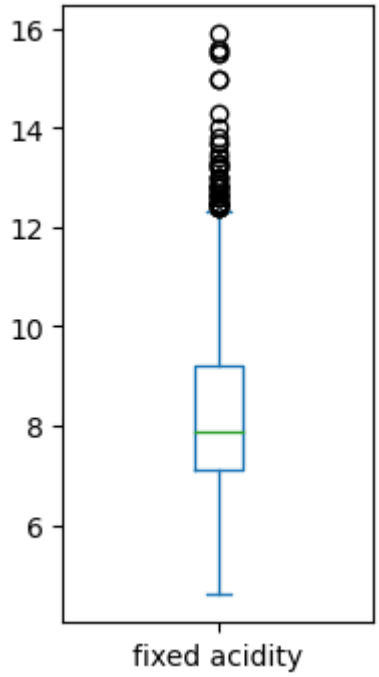
Out[75]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulf dioxi
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.0000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.4677
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.8953
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.0000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.0000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.0000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.0000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.0000

In [76]:
```
#  here we can see the huge difference betwwen "75 percentile & maximum" of so
#  so one by one we are plotting boxplot for those columns & checking the prese
```

In [87]:
```
plt.figure(figsize=(2,4))
df['fixed acidity'].plot.box()
```

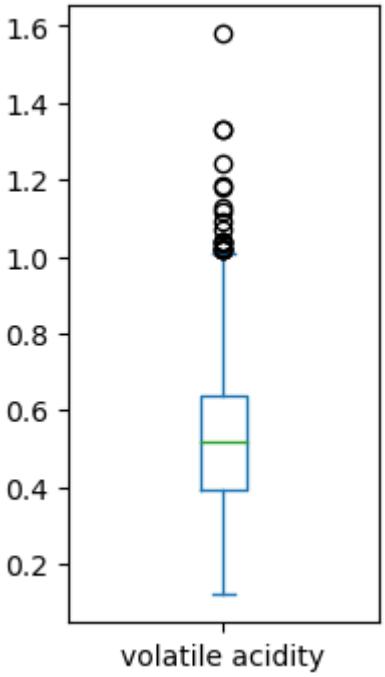
Out[87]: <AxesSubplot:>



In [88]:
```
# here we can see the presence of OUTLIERS in "fixed acidity" column.
#  similarly we can check all the cloumns one by one.
```

In [104]:
```
plt.figure(figsize=(2,4))
df['volatile acidity'].plot.box()
```

Out[104]: <AxesSubplot:>

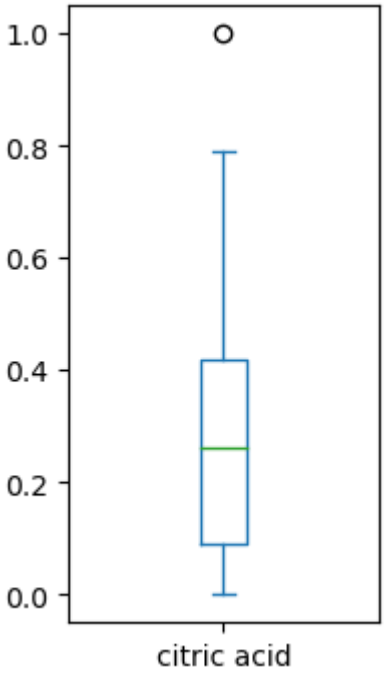


In [106]:
```
df.columns
```

Out[106]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

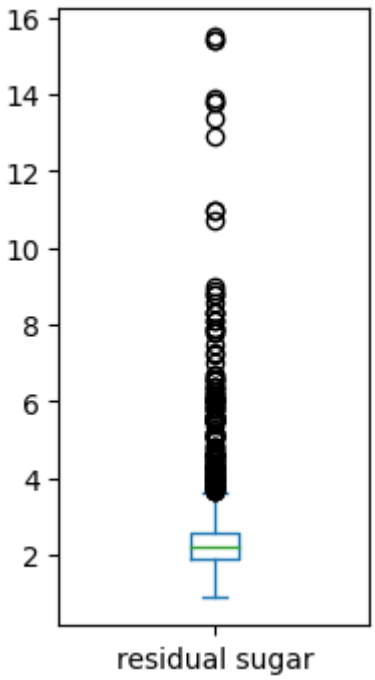
In [107]:
```
plt.figure(figsize=(2,4))
df['citric acid'].plot.box()
```

Out[107]: <AxesSubplot:>



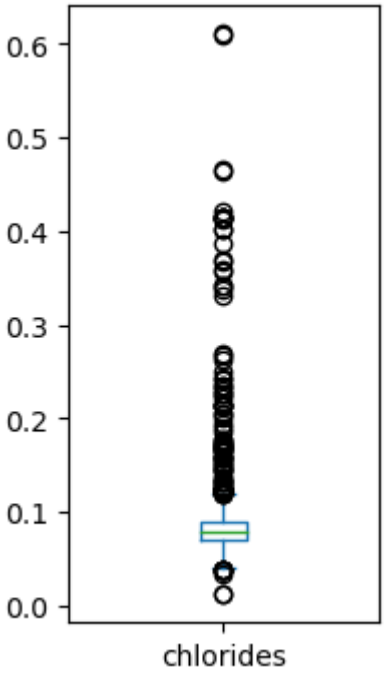
In [108]:
```
plt.figure(figsize=(2,4))
df['residual sugar'].plot.box()
```

Out[108]: <AxesSubplot:>



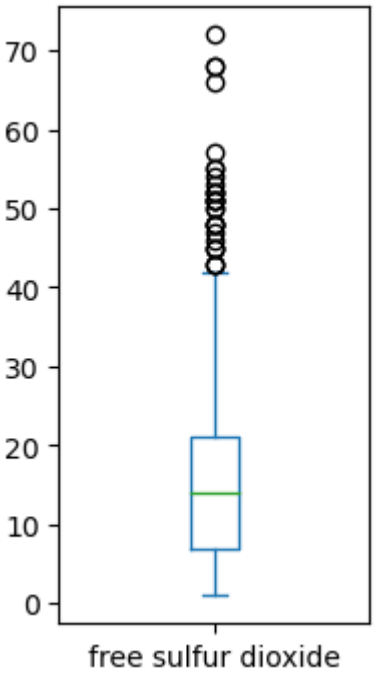
In [109]:
```
plt.figure(figsize=(2,4))
df['chlorides'].plot.box()
```

Out[109]: <AxesSubplot:>



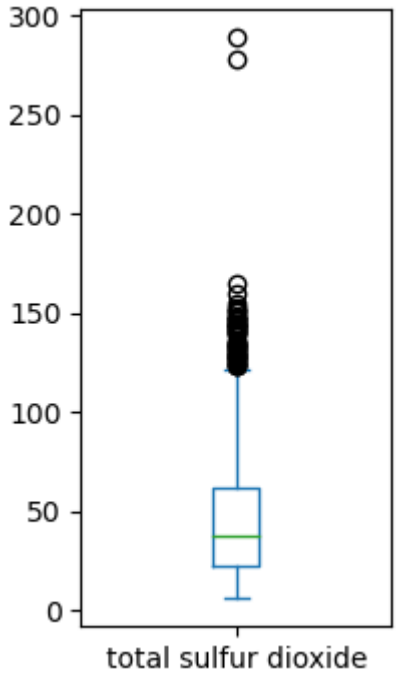
In [110]:
```
plt.figure(figsize=(2,4))
df['free sulfur dioxide'].plot.box()
```

Out[110]: <AxesSubplot:>



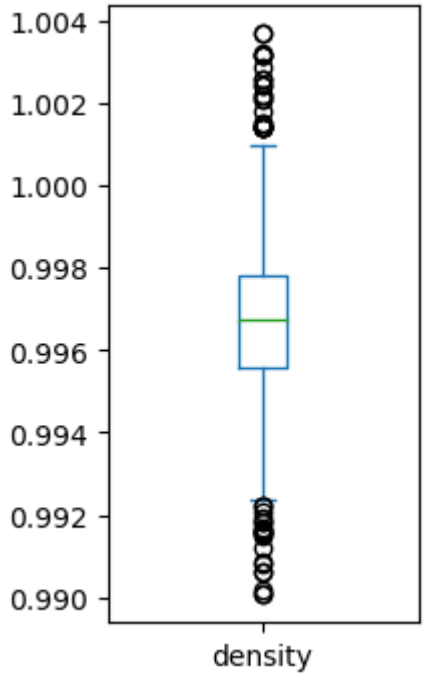
In [111]:
```
plt.figure(figsize=(2,4))
df['total sulfur dioxide'].plot.box()
```

Out[111]: <AxesSubplot:>



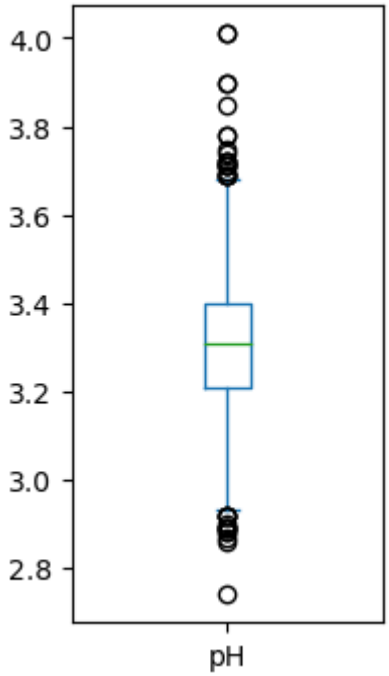
In [112]:
```
plt.figure(figsize=(2,4))
df['density'].plot.box()
```

Out[112]: <AxesSubplot:>



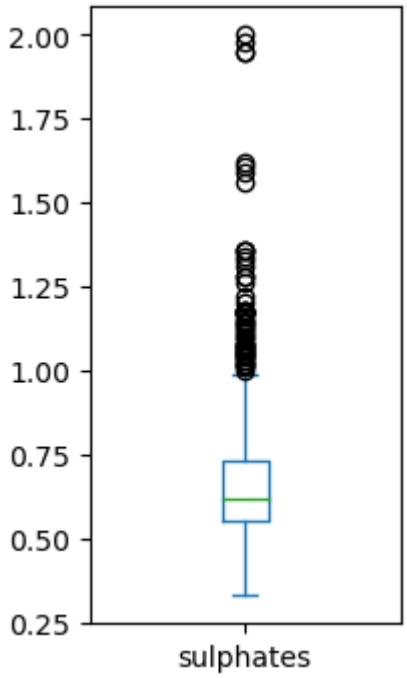
In [113]:
```
plt.figure(figsize=(2,4))
df['pH'].plot.box()
```

Out[113]: <AxesSubplot:>



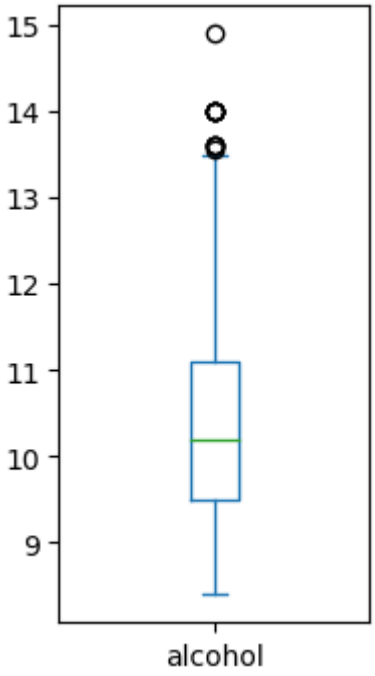
In [114]:
```
plt.figure(figsize=(2,4))
df['sulphates'].plot.box()
```

Out[114]: <AxesSubplot:>



In [115]:
```
plt.figure(figsize=(2,4))
df['alcohol'].plot.box()
```

Out[115]: <AxesSubplot:>



In [ ]:
```
# As from the above "boxplot" analysis we can see that in most of the columns (
```

In [ ]:
```
# REMOVING OUTLIERS -->
```

In [116]:
```
# As from above you have seen that we didnt identify outliers from TARGET/QUALI
#  because we no no need to remove OUTLIERS from the TARGET/QUALITY column.
```

In [117]:
```
#  Ideally we can call the OUTLIERS whos ZSCORE VALUE is LESS THEN 3 AND MORE T
# so we have to remove all the data whose ZSCORE >3 & <3
# For this first we need to identify the ZSCORE VALUES, for which we have to in
```

In [118]:
```
from scipy.stats import zscore
```

In [119]:
```
z = np.abs(zscore(df))
z
```

Out[119]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	0.528360	0.961877	1.391472	0.453218	0.243707	0.466193	0.379133	0.558274	1.288643
1	0.298547	1.967442	1.391472	0.043416	0.223875	0.872638	0.624363	0.028261	0.719933
2	0.298547	1.297065	1.186070	0.169427	0.096353	0.083669	0.229047	0.134264	0.331177
3	1.654856	1.384443	1.484154	0.453218	0.264960	0.107592	0.411500	0.664277	0.979104
4	0.528360	0.961877	1.391472	0.453218	0.243707	0.466193	0.379133	0.558274	1.288643
...	...	...	...	...	...	...	...	...	...
1594	1.217796	0.403229	0.980669	0.382271	0.053845	1.542054	0.075043	0.978765	0.899886
1595	1.390155	0.123905	0.877968	0.240375	0.541259	2.211469	0.137820	0.862162	1.353436
1596	1.160343	0.099554	0.723916	0.169427	0.243707	1.255161	0.196679	0.533554	0.705508
1597	1.390155	0.654620	0.775267	0.382271	0.264960	1.542054	0.075043	0.676657	1.677400
1598	1.332702	1.216849	1.021999	0.752894	0.434990	0.203223	0.135861	0.666057	0.511130

1599 rows × 12 columns

In [120]:
```
#  above here we "abs" i.e absolute method it returns us the all zscore values
#  so we just need to remove lesserr then 3 zscore values.
```

In [121]:
```
threshold = 3
print(np.where(z>3))
```

```
(array([  13,   14,   15,   15,   17,   17,   19,   33,   38,   42,   43,
         45,   57,   81,   81,   83,   86,   88,   91,   92,   95,  106,
        106,  109,  120,  126,  127,  142,  144,  147,  151,  151,  151,
        151,  163,  164,  169,  169,  181,  199,  226,  226,  240,  243,
        244,  258,  258,  274,  281,  291,  324,  325,  339,  340,  347,
        354,  374,  381,  391,  396,  396,  400,  400,  442,  442,  451,
        459,  467,  480,  480,  494,  515,  517,  544,  554,  554,  555,
        555,  557,  557,  568,  584,  588,  591,  595,  608,  614,  636,
        639,  649,  649,  651,  652,  652,  652,  672,  672,  684,  690,
        690,  692,  692,  695,  723,  724,  730,  754,  776,  777,  795,
        821,  832,  836,  837,  889,  899,  911,  917,  923,  925,  926,
        982, 1017, 1018, 1043, 1051, 1051, 1071, 1074, 1079, 1079, 1081,
       1081, 1111, 1114, 1131, 1154, 1165, 1175, 1186, 1231, 1235, 1244,
       1244, 1244, 1260, 1269, 1269, 1270, 1270, 1288, 1289, 1295, 1296,
       1299, 1299, 1300, 1312, 1316, 1319, 1319, 1321, 1358, 1367, 1370,
       1370, 1372, 1372, 1374, 1374, 1434, 1434, 1434, 1435, 1435, 1435,
       1469, 1474, 1474, 1474, 1476, 1476, 1476, 1478, 1493, 1496, 1505,
       1558, 1558, 1570, 1574, 1589], dtype=int64), array([ 9,  5,  5,  6,
4,  9,  4,  3,  1,  4,  9,  8,  5,  4,  9,  4,  9,
        9,  9,  9,  8,  4,  9,  6,  1,  1,  1, 10, 10,  4,  2,  4,  8,  9,
        3,  3,  4,  9,  4,  1,  4,  9,  4,  0,  0,  4,  9,  3,  4,  4,  3,
        3,  9,  9,  0,  6,  0,  0,  0,  3,  5,  3,  5,  0,  7,  4, 11, 10,
        3,  7,  3,  6, 11,  0,  0,  7,  0,  7,  0,  7,  4,  5, 10,  6,  3,
        7,  9,  6,  9,  3,  6,  6,  0,  3, 10,  1,  6,  6,  1, 11,  4,  9,
        8,  9,  1,  4,  4,  4,  4,  9, 10, 11,  7,  7,  7, 11,  3,  3,  3,
        5,  5,  5,  7,  7,  3,  4,  9,  3,  3,  3,  6,  3,  6,  8,  7,  5,
        5,  4,  5,  3,  5,  3,  3,  5,  6,  4,  7, 10,  7, 10,  9,  9,  5,
        5,  1, 11,  8,  1,  8,  4,  9,  8,  5,  9,  4,  9,  4,  9,  4, 11,
        3,  5,  7,  3,  5,  7, 11,  3,  5,  7,  3,  5,  7, 11,  6,  6, 11,
        4,  5,  4,  3,  3], dtype=int64))
```

In [ ]:
```
#  here above we can get the all data points whose z score value is > 3
```

In [132]:
```
df1 = df[(z<3).all(axis=1)]
df1.shape
df1
```

Out[132]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alco
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	
...	...	...	...	...	...	...	...	...	...	...	
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	1
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	1
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	1
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	1
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	1

1451 rows × 12 columns

In [133]:
```
df1.shape
```

Out[133]:
```
(1451, 12)
```

In [ ]:
```
# now above we can see that in our new dataset df1 the number of rows are decre
# it means we are succefully removes the OUTLIERS from the dataset
```

In [134]:
```
# here again we are checking the zscore of our new dataset df1 ()
z1 = np.abs(zscore(df1))
z1
```

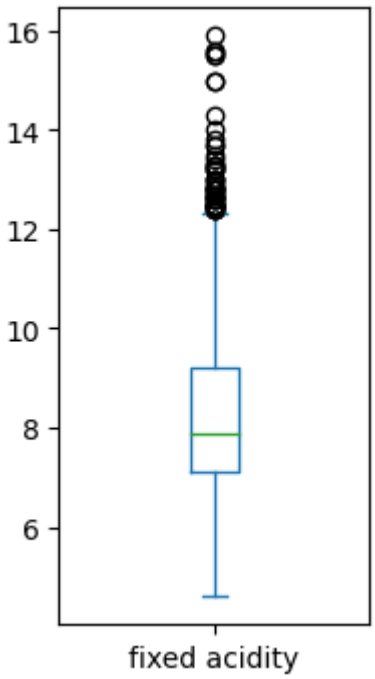
Out[134]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	0.552930	1.050914	1.390400	0.565439	0.258851	0.441060	0.330784	0.635485	1.375896
1	0.309900	2.119336	1.390400	0.246832	0.790825	1.063255	0.790477	0.052390	0.821951
2	0.309900	1.407054	1.180831	0.101284	0.504550	0.011256	0.348768	0.169009	0.396561
3	1.755851	1.442071	1.543569	0.565439	0.306564	0.203646	0.552634	0.752103	1.105544
4	0.552930	1.050914	1.390400	0.565439	0.258851	0.441060	0.330784	0.635485	1.375896
...	...	...	...	...	...	...	...	...	...
1594	1.282019	0.457346	0.971261	0.449401	0.409125	1.815413	0.008992	1.055489	0.950506
1595	1.464291	0.160562	0.866477	0.217323	0.926828	2.567570	0.246835	0.927209	1.446794
1596	1.221261	0.076865	0.709300	0.101284	0.258851	1.493059	0.126918	0.565690	0.737811
1597	1.464291	0.724451	0.761692	0.449401	0.306564	1.815413	0.008992	0.723125	1.801286
1598	1.403533	1.264001	1.072038	1.407221	0.688265	0.311097	0.058963	0.711464	0.525116

1451 rows × 12 columns

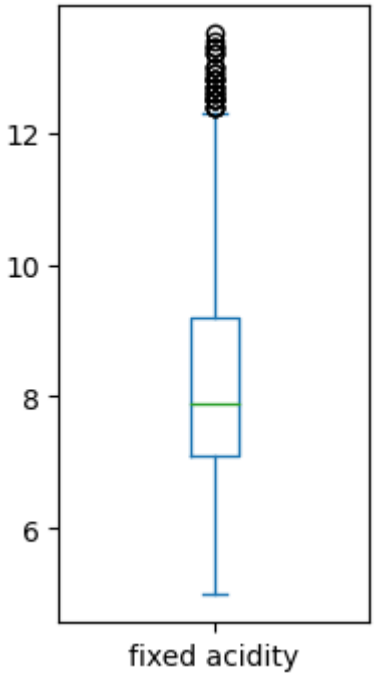
In [136]:
```
plt.figure(figsize=(2,4))
df['fixed acidity'].plot.box()
```

Out[136]: <AxesSubplot:>



In [137]:
```
plt.figure(figsize=(2,4))
df1['fixed acidity'].plot.box()
```

Out[137]: <AxesSubplot:>



In [ ]:
```
#  here above you can clearly seen the diffrence between the df & df1 for colu
#  so we are succesfully removed the outliers from the dataset and save it to
```

In [138]:
```
df1
```

Out[138]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alco
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	
...	...	...	...	...	...	...	...	...	...	...	
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	1
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	1
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	1
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	1
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	1

1451 rows × 12 columns

In [ ]:
```
#  CHECKING & REMOVING OF SKEWNESS FROM THE DATASET
```

In [139]:
```
df1.skew()
```

Out[139]:
```
fixed acidity           0.823934
volatile acidity        0.380659
citric acid             0.291297
residual sugar          2.456107
chlorides               2.275162
free sulfur dioxide     0.869250
total sulfur dioxide    1.183161
density                 0.055738
pH                      0.114705
sulphates               0.891492
alcohol                 0.758958
quality                 0.407865
dtype: float64
```

In [140]:
```
#  here above by checking skewness we can see that the skewness of most of the
#  "residual sugar", "chlorides" and "total sulfur dioxide" columns.
#  because skewness shows the distribution of data, if the distribution is wide
#  the ideal values of skewness is (-0.5 to +0.5) (bellshaped curve)
# we can remove the skewness by many types like:- CUBEROOT METHOD, POWERTRANSFO
```

In [141]:
```
# so here we have to remove skewness from those columns by using CUBEROOT METHO
```

In [148]:
```
df['residual sugar'] = np.cbrt (df1 ['residual sugar'])
df1.skew()
```

Out[148]:
```
fixed acidity           0.823934
volatile acidity        0.380659
citric acid             0.291297
residual sugar          1.629524
chlorides               2.275162
free sulfur dioxide     0.869250
total sulfur dioxide    1.183161
density                 0.055738
pH                      0.114705
sulphates               0.891492
alcohol                 0.758958
quality                 0.407865
dtype: float64
```

In [ ]:
```
#  here we can see that the skewness of "residual sugar" column is decreased fr
```

In [168]:
```
df1['total sulfur dioxide'] = np.cbrt (df1 ['total sulfur dioxide'])
df1.skew()
```

Out[168]:
```
fixed acidity           0.823934
volatile acidity        0.380659
citric acid             0.291297
residual sugar          1.629524
chlorides               2.275162
free sulfur dioxide     0.869250
total sulfur dioxide    0.342621
density                 0.055738
pH                      0.114705
sulphates               0.891492
alcohol                 0.758958
quality                 0.407865
dtype: float64
```

In [180]:
```
df1['chlorides'] = np.cbrt (df1 ['chlorides'])
df1.skew()
```

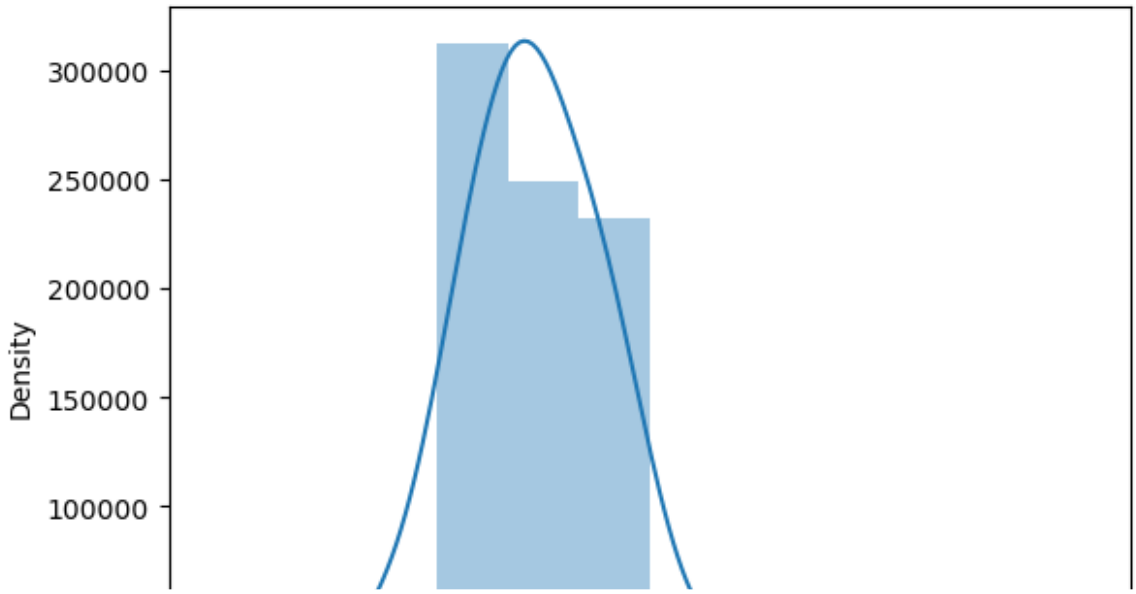
Out[180]:
```
fixed acidity           0.823934
volatile acidity        0.380659
citric acid             0.291297
residual sugar          1.227116
chlorides               0.664570
free sulfur dioxide     0.869250
total sulfur dioxide    0.342621
density                 0.055738
pH                      0.114705
sulphates               0.891492
alcohol                 0.758958
quality                 0.407865
dtype: float64
```

In [ ]:
```
#  here above we removed the skewness from "residual sugar", "chloride" & "tot
```

In [ ]:
```
# here we can see the distribution of data in "residual sugar" and "chlorides"
```

In [186]:
```
sns.distplot(df1['residual sugar'],bins=10)
```

Out[186]:
```
<AxesSubplot:xlabel='residual sugar', ylabel='Density'>
```



In [ ]:
```
# now upto here above we have removed the skewness from the data.
```

In [188]:
```
df1
```

Out[188]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	al
0	7.4	0.700	0.00	1.000004	0.908968	11.0	3.239612	0.99780	3.51	0.56	
1	7.8	0.880	0.00	1.000005	0.917567	25.0	4.061548	0.99680	3.20	0.68	
2	7.8	0.760	0.04	1.000005	0.915423	15.0	3.779763	0.99700	3.26	0.65	
3	11.2	0.280	0.56	1.000004	0.908522	17.0	3.914868	0.99800	3.16	0.58	
4	7.4	0.700	0.00	1.000004	0.908968	11.0	3.239612	0.99780	3.51	0.56	
...	...	...	...	...	...	...	...	...	...	...	
1594	6.2	0.600	0.08	1.000004	0.914678	32.0	3.530348	0.99490	3.45	0.58	
1595	5.9	0.550	0.10	1.000004	0.902140	39.0	3.708430	0.99512	3.52	0.76	
1596	6.3	0.510	0.13	1.000005	0.908968	29.0	3.419952	0.99574	3.42	0.75	
1597	5.9	0.645	0.12	1.000004	0.908522	32.0	3.530348	0.99547	3.57	0.71	
1598	6.0	0.310	0.47	1.000007	0.904735	18.0	3.476027	0.99549	3.39	0.66	

1451 rows × 12 columns

In [202]:
```
# Finding MULTICOLLINEARITY between the columns
```

In [ ]:
```
#  SPLITTING OF DATA
```

In [213]:
```
x1 = df1.iloc[:,:-1]
```

In [214]:
```
x1.shape
```

Out[214]: (1451, 11)

In [215]:
```
y1= df1.iloc[:,-1]
y1.shape
```

Out[215]: (1451,)

In [ ]:
```
# here above we are splitting the data in 'x' & 'y'
#  where 'x' is the fetures & 'y' is our target column.
```

In [ ]:
```
# to find the multicollinearity between the features  and remove it we can use
# we can not aplly VIF on the TARGET COLUMN
# for apllyin VIF we have to import some libraries as follows
```

In [206]:
```
import statsmodels.api as sm
from scipy import stats
from statsmodels .stats.outliers_influence import variance_inflation_factor
```

In [227]:
```
# here we are making "def function" for calculating VIF
def calc_vif(x1):
    vif = pd.DataFrame()
    vif["FETURES"] = x1.columns
    vif["VIF FACTOR"] = [variance_inflation_factor(x1.values,i) for i in range
    return (vif)
```

In [228]:
```
x1.shape
```

Out[228]: (1451, 11)

In [229]:
```
calc_vif(x1)
```

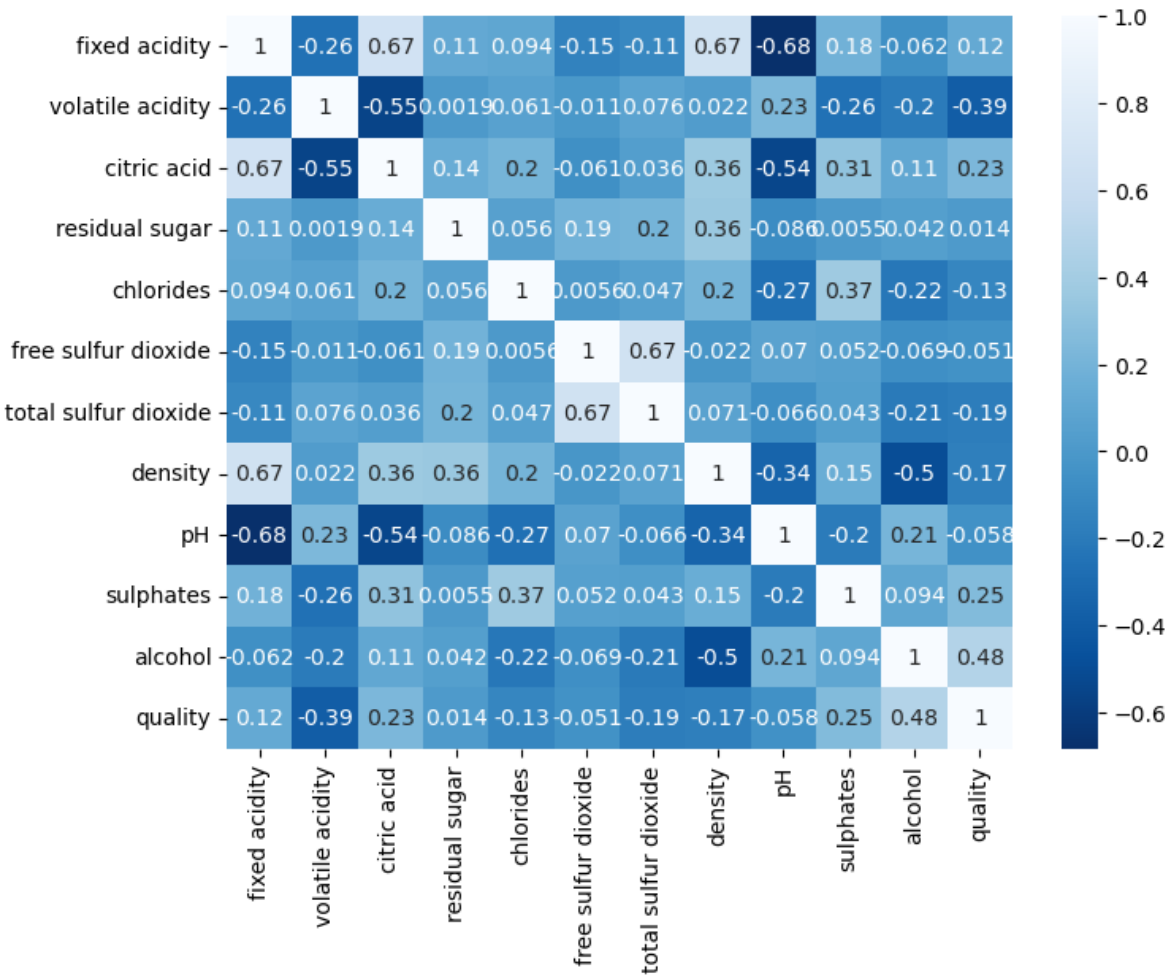
Out[229]:

	FETURES	VIF FACTOR
0	fixed acidity	1.816938e+02
1	volatile acidity	1.957835e+01
2	citric acid	9.152273e+00
3	residual sugar	1.435171e+06
4	chlorides	1.698077e+04
5	free sulfur dioxide	8.075290e+00
6	total sulfur dioxide	5.141863e+01
7	density	1.563874e+06
8	pH	1.645206e+03
9	sulphates	3.293070e+01
10	alcohol	2.432639e+02

In [ ]:
```
# here above we can see that the VIF calue of "free sulfur dioxide" "total sulf
# so we have to drop the column with highest VIF but befor we have to check the
#  by using heatmap
```

In [230]:
```
plt.figure(figsize=(8,6))
sns.heatmap(dfcor,annot=True,cmap="Blues_r")
```

Out[230]: <AxesSubplot:>



In [ ]:
```
# here above in the heatmap we can find that the correlation of "free sulfurdi
#  as compared to other HIGLY VIF value columnsit is very low, so we can drop
```

In [ ]:
```
x1 = df1.drop(["free sulfur dioxide"], axis=1, inplace = True)
# here above we droped the column "free sulfur dioxide"
```

In [236]:
```
df1.columns
```

Out[236]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'total sulfur dioxide', 'density', 'pH', 'sulphates',
       'alcohol', 'quality'],
      dtype='object')
```

In [ ]:
```
# here we can see that the "free sulfur dioxide" column is removed fromt he lis
```

In [246]:
```
x2 = df1.iloc[:,:-1]
x2.shape
```

Out[246]: (1451, 9)

In [239]:
```
calc_vif(x2)
```

Out[239]:

	FETURES	VIF FACTOR
0	fixed acidity	1.792890e+02
1	volatile acidity	1.930398e+01
2	citric acid	8.992555e+00
3	residual sugar	1.414287e+06
4	chlorides	1.697509e+04
5	total sulfur dioxide	2.420939e+01
6	density	1.541220e+06
7	pH	1.626710e+03
8	sulphates	3.272154e+01
9	alcohol	2.431894e+02

In [242]:
```
x1 = df1.drop(["citric acid"], axis=1, inplace = True)
```

In [244]:
```
df1.columns
```

Out[244]:
```
Index(['fixed acidity', 'volatile acidity', 'residual sugar', 'chlorides',
       'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol',
       'quality'],
      dtype='object')
```

In [245]:
```
df1.shape
```

Out[245]:
```
(1451, 10)
```

In [ ]:
```
# here above we dropped two columns with high VIF VALUES
```

In [247]:
```
calc_vif(x2)
```

Out[247]:

	FETURES	VIF FACTOR
0	fixed acidity	1.619552e+02
1	volatile acidity	1.406346e+01
2	residual sugar	1.403532e+06
3	chlorides	1.696158e+04
4	total sulfur dioxide	2.345388e+01
5	density	1.531234e+06
6	pH	1.614316e+03
7	sulphates	3.270769e+01
8	alcohol	2.329306e+02

In [ ]:
```
#  now we can see that the VIF VALUES of all columns are in similar form
```

In [248]:
```
df1
```

Out[248]:

	fixed acidity	volatile acidity	residual sugar	chlorides	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	1.000004	0.908968	3.239612	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	1.000005	0.917567	4.061548	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	1.000005	0.915423	3.779763	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	1.000004	0.908522	3.914868	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	1.000004	0.908968	3.239612	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	1.000004	0.914678	3.530348	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	1.000004	0.902140	3.708430	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	1.000005	0.908968	3.419952	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	1.000004	0.908522	3.530348	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	1.000007	0.904735	3.476027	0.99549	3.39	0.66	11.0	6

1451 rows × 10 columns

In [249]:
```
x2.shape
```

Out[249]:
```
(1451, 9)
```

In [250]:
```
y2 = df1.iloc[:,-1]
y2.shape
```

Out[250]:
```
(1451,)
```

In [ ]:
```
# here x2 and y2 are our FEATURE AND TARGET COLUMNS
```

In [ ]:
```

```

In [ ]:
```
#  SCALING TECHNIQUES :-
#  now above we can see that our dataset is not STANDARDISE, so by using SCALI
#  to get better results.
```

In [191]:
```
from sklearn.preprocessing import StandardScaler
```

In [192]:
```
st = StandardScaler()
```

In [251]:
```
x2= st.fit_transform(x2)
x2
```

Out[251]:
```
array([[-0.55292997,  1.05091371, -0.62231267, ...,  1.37589608,
        -0.63645843, -0.99985592],
       [-0.3099004 ,  2.11933582,  0.45871385, ..., -0.8219512 ,
         0.2883529 , -0.6081737 ],
       [-0.3099004 ,  1.40705441,  0.03616248, ..., -0.3965614 ,
         0.05715007, -0.6081737 ],
       ...,
       [-1.22126131, -0.07686519,  0.03616248, ...,  0.73781138,
         0.82782618,  0.56687296],
       [-1.46429088,  0.72445139, -0.44552952, ...,  1.80128587,
         0.51955574, -0.21649148],
       [-1.40353349, -1.26400088,  1.58029014, ...,  0.52511649,
         0.13421768,  0.56687296]])
```

In [ ]:
```
#  here above we can see that after standardization of our 'x'
# now we have to make it into DataFrame
```

In [252]:
```
xf = pd.DataFrame(data=x2)
print(xf)
```

```
             0         1         2         3         4         5         6  \
0    -0.552930  1.050914 -0.622313 -0.182053 -0.154751  0.635485  1.375896
1    -0.309900  2.119336  0.458714  0.922610  0.931518  0.052390 -0.821951
2    -0.309900  1.407054  0.036162  0.647121  0.559112  0.169009 -0.396561
3     1.755851 -1.442071 -0.622313 -0.239320  0.737666  0.752103 -1.105544
4    -0.552930  1.050914 -0.622313 -0.182053 -0.154751  0.635485  1.375896
...        ...       ...       ...       ...       ...       ...       ...
1446 -1.282019  0.457346 -0.445530  0.551435  0.229486 -1.055489  0.950506
1447 -1.464291  0.160562 -0.117041 -1.059233  0.464838 -0.927209  1.446794
1448 -1.221261 -0.076865  0.036162 -0.182053  0.083586 -0.565690  0.737811
1449 -1.464291  0.724451 -0.445530 -0.239320  0.229486 -0.723125  1.801286
1450 -1.403533 -1.264001  1.580290 -0.725863  0.157694 -0.711464  0.525116

             7         8
0    -0.636458 -0.999856
1     0.288353 -0.608174
2     0.057150 -0.608174
3    -0.482323 -0.608174
4    -0.636458 -0.999856
...        ...       ...
1446 -0.482323  0.077270
1447  0.904894  0.762714
1448  0.827826  0.566873
1449  0.519556 -0.216491
1450  0.134218  0.566873

[1451 rows x 9 columns]
```

In [ ]:
```
# here above our whole dataframe 'df1' is now tranform and we can see it above
# here above we not applied SCALING TECHNIQUE on the TARGET COLUMN.(Y)
# BEFORE APPLYING MACHINE LEARNING MODEL WE HAVE TO APPLY SCALING & RESAMPLING
```

In [ ]:
```
# RESAMPLING TECHNIQUES
```

In [255]:
```
y2.value_counts()
```

Out[255]:
```
5    617
6    586
7    185
4     47
8     16
Name: quality, dtype: int64
```

In [ ]:
```
#  here above we can see that the distribution of values with the unique numbe
#  equal by using RESAMPLING TECHNIQUE.
```

In [ ]:
```
#  here for RESAMPLING we are using SMOTE METHOD i.e SYNTHETIC MINORITY OVERSA
# here with the help of SMOTE we can solve the CLASS IMMBALANCE PROBLEM.
```

In [ ]:
```
# for this first we need to install and import some libraries :-
```

In [256]:
```
pip install -U imbalanced-learn
```

```
Collecting imbalanced-learnNote: you may need to restart the kernel to use up
dated packages.

  Downloading imbalanced_learn-0.11.0-py3-none-any.whl (235 kB)
     ---------------------------------------- 235.6/235.6 kB 1.8 MB/s eta 0:00:
00
Collecting joblib>=1.1.1
  Downloading joblib-1.3.1-py3-none-any.whl (301 kB)
     ---------------------------------------- 302.0/302.0 kB 2.1 MB/s eta 0:00:
00
Requirement already satisfied: numpy>=1.17.3 in c:\users\admin\anaconda3\lib
\site-packages (from imbalanced-learn) (1.21.5)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\admin\anaconda
3\lib\site-packages (from imbalanced-learn) (1.0.2)
Requirement already satisfied: scipy>=1.5.0 in c:\users\admin\anaconda3\lib\s
ite-packages (from imbalanced-learn) (1.9.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\admin\anacond
a3\lib\site-packages (from imbalanced-learn) (2.2.0)
Installing collected packages: joblib, imbalanced-learn
  Attempting uninstall: joblib
    Found existing installation: joblib 1.1.0
    Uninstalling joblib-1.1.0:
      Successfully uninstalled joblib-1.1.0
Successfully installed imbalanced-learn-0.11.0 joblib-1.3.1
```

In [257]:
```
from imblearn.over_sampling import SMOTE
```

In [258]:
```
smt = SMOTE()
```

In [259]:
```
trainx, trainy = smt.fit_resample(x2,y2)
```

In [260]:
```
trainy.value_counts()
```

Out[260]:
```
5    617
6    617
7    617
4    617
8    617
Name: quality, dtype: int64
```

In [262]:
```
# here above we can see that, we appied SMOTE SUCCEFULLY ON THE DATASET,
# and BALANCE the dataset
```

In [266]:
```
trainy.shape
```

Out[266]:
```
(3085,)
```

In [ ]:
```
# here we can see that the no. of rows are increased from 1451 to 3085, that me
```

In [269]:
```
trainx.shape
```

Out[269]:
```
(3085, 9)
```

In [ ]:
```

```

In [ ]:
```
# FINDING THE BEST RANDOM STATE FOR THE MODEL
```

In [271]:
```
from sklearn.model_selection import train_test_split
```

In [273]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_r
```

In [274]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [275]:
```
dtc = DecisionTreeClassifier
```

In [280]:
```
maxaccu = 0
maxrs = 0

for i in range(1,200):
    x_train,x_test,y_train,y_test = train_test_split(trainx,trainy,test_size=0
    dtc = DecisionTreeClassifier()
    dtc.fit(x_train,y_train)
    pred = dtc.predict(x_test)
    acc = accuracy_score(y_test,pred)

    if acc > maxaccu :
        maxaccu = acc
        maxrs = i

print ("Best accuracy is",maxaccu, "at random state", maxrs)
```

```
Best accuracy is 0.8233387358184765 at random state 198
```

In [ ]:
```
# here above we can find the MAXIMUM ACCURACY of 82% is occurs on random state=
```

In [ ]:
```
# now we are apllying the model
```

In [282]:
```
from sklearn.model_selection import GridSearchCV
```

In [287]:
```
grid_param = {'criterion':['gini','entropy']}
```

In [288]:
```
gd_sr = GridSearchCV (estimator=dtc, param_grid= grid_param, scoring="accuracy
```

In [289]:
```
gd_sr.fit(trainx,trainy)
```

Out[289]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy']}, scoring='accurac
y')
```

In [291]:
```
best_perameter = gd_sr.best_params_
print(best_perameter)
```

```
{'criterion': 'entropy'}
```

In [ ]:
```
# here we can find the best perameter for the model is "entropy"
```

In [294]:
```
best_result = gd_sr.best_score_
print(best_result)
```

```
0.7063209076175041
```

In [295]:
```
print(round(best_result,2))
```

```
0.71
```

In [ ]:
```
#  the best score is .71
```

In [ ]:
```
# now applying the model with "entropy" parameter and "198" randomstate
```

In [299]:
```
dtc1 = DecisionTreeClassifier (criterion="entropy")
```

In [300]:
```
x_train,x_test,y_train,y_test = train_test_split(trainx,trainy,test_size=0.20,
```

In [302]:
```
dtc1.fit(x_train,y_train)
dtc1.score(x_train,y_train)
dtc1pred = dtc1.predict(x_test)
print(accuracy_score(y_test,dtc1pred))
print(confusion_matrix(y_test,dtc1pred))
print(classification_report(y_test,dtc1pred))
```

```
0.8055105348460292
[[128  14   3   1   0]
 [  6  89  26   3   0]
 [ 11  28  66  11   2]
 [  0   1   5  95   3]
 [  0   0   2   4 119]]
              precision    recall  f1-score   support

           4       0.88      0.88      0.88       146
           5       0.67      0.72      0.70       124
           6       0.65      0.56      0.60       118
           7       0.83      0.91      0.87       104
           8       0.96      0.95      0.96       125

    accuracy                           0.81       617
   macro avg       0.80      0.80      0.80       617
weighted avg       0.80      0.81      0.80       617
```

In [ ]:
```
# here above we can see that the accuracy of our model is = 88%
```

In [303]:
```
def pred_func(q):
    q= q.reshape(1,9)
    qt = dtc1.predict(q)
    print(qt)

    if qt == 3:
        print("not good quality")
    elif (qt == 4):
        print ("not good quality")
    elif (qt == 5):
        print ("not good quality")
    elif (qt == 6):
        print ("not good quality")
    elif (qt == 7):
        print ("good quality")
    elif (qt == 8):
        print ("good quality")
    elif (qt == 9):
        print("good quality")
```

In [304]:
```
q= np.array([-0.552930,1.050914,-0.622313,-0.182053,-0.154751,0.635485,1.37589(
pred_func(q)
```

```
[5]
not good quality
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: