In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [2]:
```
import pandas as pd
df=pd.read_csv(r"https://raw.githubusercontent.com/shrikant-temburwar/Wine-Qualit
```

In [3]:
```
# top five dataset
df.head()
```

Out[3]:

	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 [4]:
```
# last five dtaset
df.tail()
```

Out[4]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11

In [5]:
```
# check sample data
df.sample()
```

Out[5]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
603	13.2	0.46	0.52	2.2	0.071	12.0	35.0	1.0006	3.1	0.56	9.0

In [6]:
```
# check shape
df.shape
```

Out[6]:
```
(1599, 12)
```

In [7]:
```
# check info
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 [8]:
```
# indicate the columns
df.columns
```

Out[8]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [9]:
```
# indicate the null values
df.isnull().sum()
```

Out[9]:
```
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 [10]:
```
# check duplicated values
df.duplicated().sum()
```

Out[10]:
```
240
```

In [11]:
```
# show the duplicated values
df[df.duplicated()]
```

Out[11]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
4	7.4	0.700	0.00	1.90	0.076	11.0	34.0	0.99780	3.51	0.56	9
11	7.5	0.500	0.36	6.10	0.071	17.0	102.0	0.99780	3.35	0.80	10
27	7.9	0.430	0.21	1.60	0.106	10.0	37.0	0.99660	3.17	0.91	9
40	7.3	0.450	0.36	5.90	0.074	12.0	87.0	0.99780	3.33	0.83	10
65	7.2	0.725	0.05	4.65	0.086	4.0	11.0	0.99620	3.41	0.39	10
...	...	...	...	...	...	...	...	...	...	...	
1563	7.2	0.695	0.13	2.00	0.076	12.0	20.0	0.99546	3.29	0.54	10
1564	7.2	0.695	0.13	2.00	0.076	12.0	20.0	0.99546	3.29	0.54	10
1567	7.2	0.695	0.13	2.00	0.076	12.0	20.0	0.99546	3.29	0.54	10
1581	6.2	0.560	0.09	1.70	0.053	24.0	32.0	0.99402	3.54	0.60	11
1596	6.3	0.510	0.13	2.30	0.076	29.0	40.0	0.99574	3.42	0.75	11

240 rows × 12 columns

In [12]:
```
df=df.drop_duplicates() # drop the duplicates values
```

In [13]:
```
df.shape  # check the shape
```

Out[13]:
```
(1359, 12)
```

In [14]:
```
# check the Unique values
df.nunique()
```

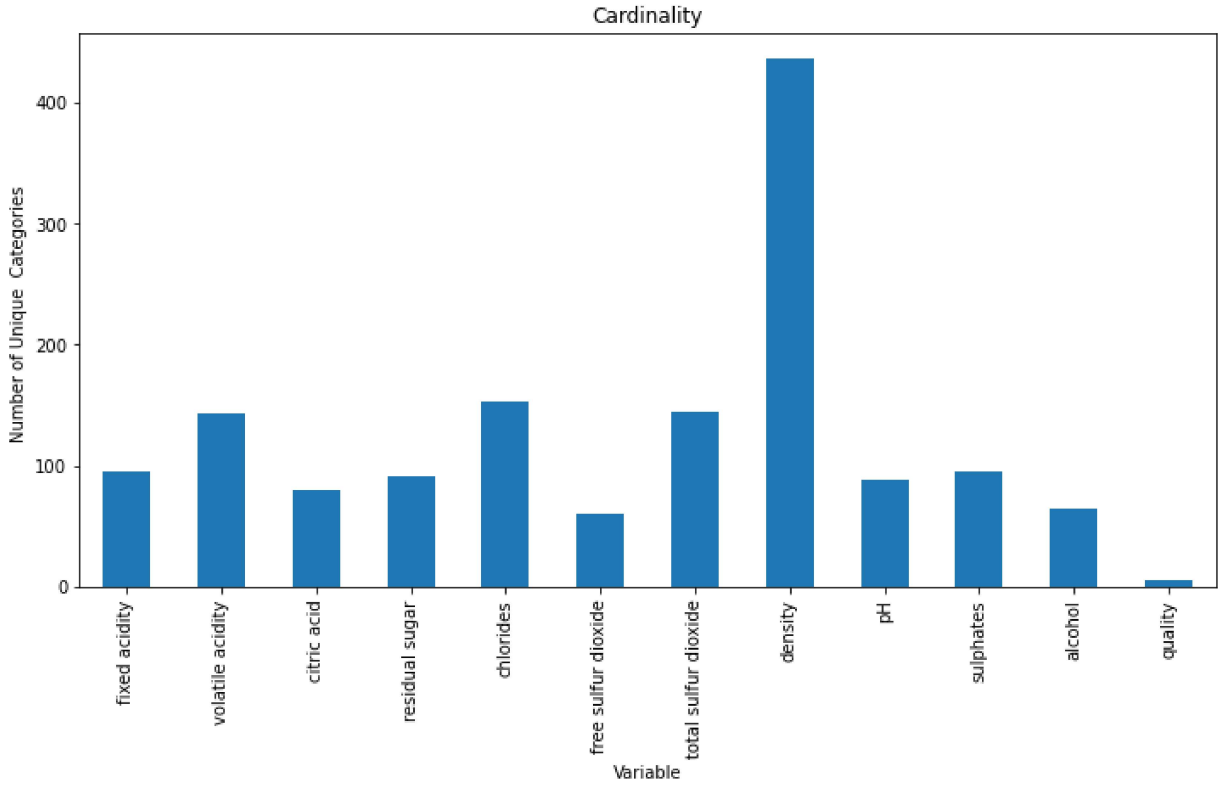
Out[14]:
```
fixed acidity           96
volatile acidity       143
citric acid             80
residual sugar          91
chlorides              153
free sulfur dioxide     60
total sulfur dioxide   144
density                436
pH                      89
sulphates               96
alcohol                 65
quality                  6
dtype: int64
```

In [15]:
```
df.nunique().sort_values()
```

Out[15]:
```
quality                  6
free sulfur dioxide     60
alcohol                 65
citric acid             80
pH                      89
residual sugar          91
fixed acidity           96
sulphates               96
volatile acidity       143
total sulfur dioxide   144
chlorides              153
density                436
dtype: int64
```

In [16]:
```
df.nunique().plot.bar(figsize=(12,6))
plt.ylabel('Number of Unique  Categories ')
plt.xlabel('Variable')
plt.title("Cardinality")
```

Out[16]: Text(0.5, 1.0, 'Cardinality')



In [17]:
```
df['fixed acidity'].value_counts()
```

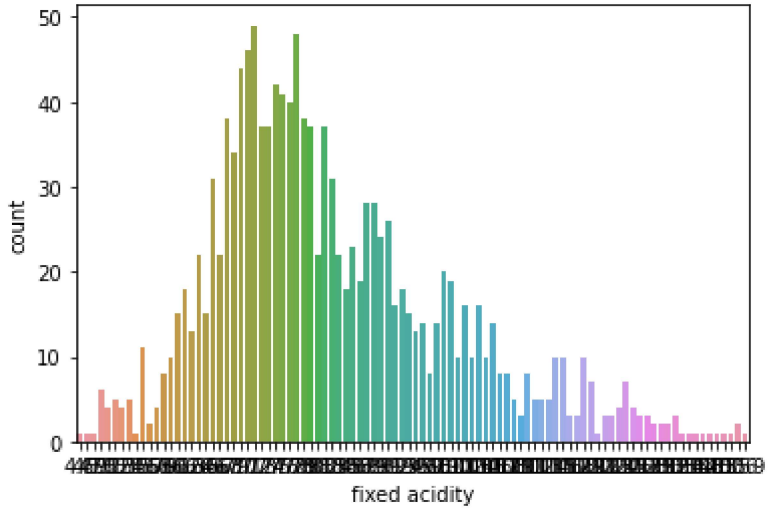
Out[17]:
```
7.2     49
7.8     48
7.1     46
7.0     44
7.5     42
        ..
13.8     1
13.4     1
4.7      1
15.0     1
5.5      1
Name: fixed acidity, Length: 96, dtype: int64
```

In [18]:
```
df['fixed acidity'].unique()
```

Out[18]:
```
array([ 7.4,  7.8, 11.2,  7.9,  7.3,  7.5,  6.7,  5.6,  8.9,  8.5,  8.1,
        7.6,  6.9,  6.3,  7.1,  8.3,  5.2,  5.7,  8.8,  6.8,  4.6,  7.7,
        8.7,  6.4,  6.6,  8.6, 10.2,  7. ,  7.2,  9.3,  8. ,  9.7,  6.2,
        5. ,  4.7,  8.4, 10.1,  9.4,  9. ,  8.2,  6.1,  5.8,  9.2, 11.5,
        5.4,  9.6, 12.8, 11. , 11.6, 12. , 15. , 10.8, 11.1, 10. , 12.5,
       11.8, 10.9, 10.3, 11.4,  9.9, 10.4, 13.3, 10.6,  9.8, 13.4, 10.7,
       11.9, 12.4, 12.2, 13.8,  9.1, 13.5, 10.5, 12.6, 14. , 13.7,  9.5,
       12.7, 12.3, 15.6,  5.3, 11.3, 13. ,  6.5, 12.9, 14.3, 15.5, 11.7,
       13.2, 15.9, 12.1,  5.1,  4.9,  5.9,  6. ,  5.5])
```

In [19]:
```
sns.countplot(df['fixed acidity'])
```

Out[19]:
```
<AxesSubplot:xlabel='fixed acidity', ylabel='count'>
```



In [20]:
```
df['citric acid'].unique()
```

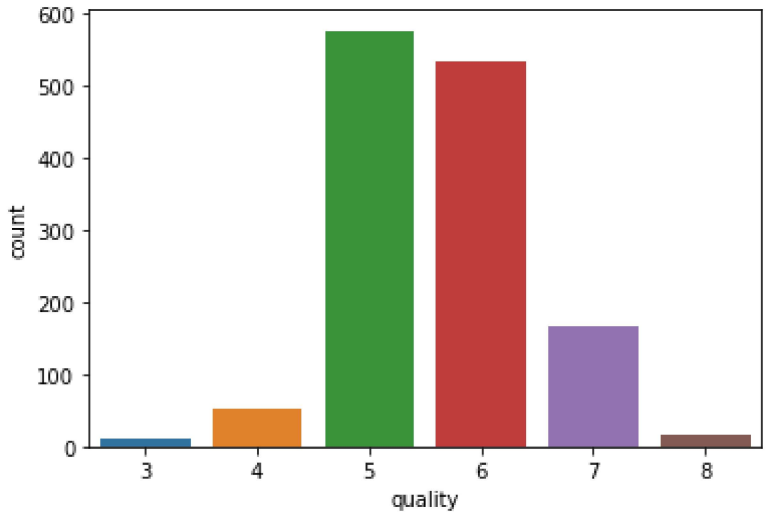
Out[20]:
```
array([0.  , 0.04, 0.56, 0.06, 0.02, 0.36, 0.08, 0.29, 0.18, 0.19, 0.28,
       0.51, 0.48, 0.31, 0.21, 0.11, 0.14, 0.16, 0.24, 0.07, 0.12, 0.25,
       0.09, 0.3 , 0.2 , 0.22, 0.15, 0.43, 0.52, 0.23, 0.37, 0.26, 0.57,
       0.4 , 0.49, 0.05, 0.54, 0.64, 0.7 , 0.47, 0.44, 0.17, 0.68, 0.53,
       0.1 , 0.01, 0.55, 1.  , 0.03, 0.42, 0.33, 0.32, 0.35, 0.6 , 0.74,
       0.58, 0.5 , 0.76, 0.46, 0.45, 0.38, 0.39, 0.66, 0.62, 0.67, 0.79,
       0.63, 0.61, 0.71, 0.65, 0.59, 0.34, 0.69, 0.73, 0.72, 0.41, 0.27,
       0.75, 0.13, 0.78])
```

In [21]:
```
df['quality'].unique()
```

Out[21]:
```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

In [22]:
```
sns.countplot(df['quality'])
```

Out[22]:
```
<AxesSubplot:xlabel='quality', ylabel='count'>
```

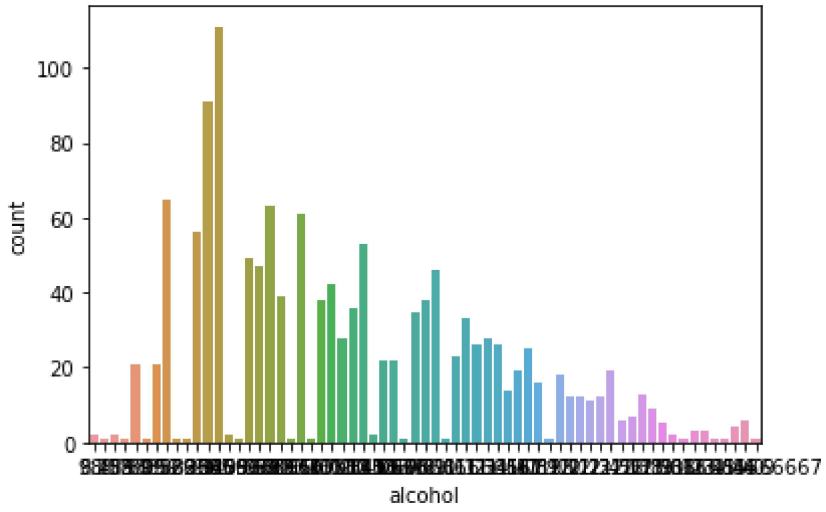


In [23]:
```
df['alcohol'].unique()
```

Out[23]:
```
array([ 9.4       ,  9.8       , 10.        ,  9.5       , 10.5       ,
        9.2       ,  9.9       ,  9.1       ,  9.3       ,  9.        ,
        9.7       , 10.1       , 10.6       ,  9.6       , 10.8       ,
       10.3       , 13.1       , 10.2       , 10.9       , 10.7       ,
       12.9       , 10.4       , 13.        , 14.        , 11.5       ,
       11.4       , 12.4       , 11.        , 12.2       , 12.8       ,
       12.6       , 12.5       , 11.7       , 11.3       , 12.3       ,
       12.        , 11.9       , 11.8       ,  8.7       , 13.3       ,
       11.2       , 11.6       , 11.1       , 13.4       , 12.1       ,
        8.4       , 12.7       , 14.9       , 13.2       , 13.6       ,
       13.5       , 10.03333333,  9.55       ,  8.5       , 11.06666667,
        9.56666667, 10.55       ,  8.8       , 13.56666667, 11.95       ,
        9.95       ,  9.23333333,  9.25       ,  9.05      , 10.75       ])
```

In [24]:
```
sns.countplot(df['alcohol'])
```

Out[24]:
```
<AxesSubplot:xlabel='alcohol', ylabel='count'>
```



In [25]:
```
# check memory_usag
df.memory_usage()
```

Out[25]:
```
Index                 10872
fixed acidity         10872
volatile acidity      10872
citric acid           10872
residual sugar        10872
chlorides             10872
free sulfur dioxide   10872
total sulfur dioxide  10872
density               10872
pH                    10872
sulphates             10872
alcohol               10872
quality               10872
dtype: int64
```

Categorical and Numerical Feature

In [26]:
```
[feature for feature in df.columns]
```

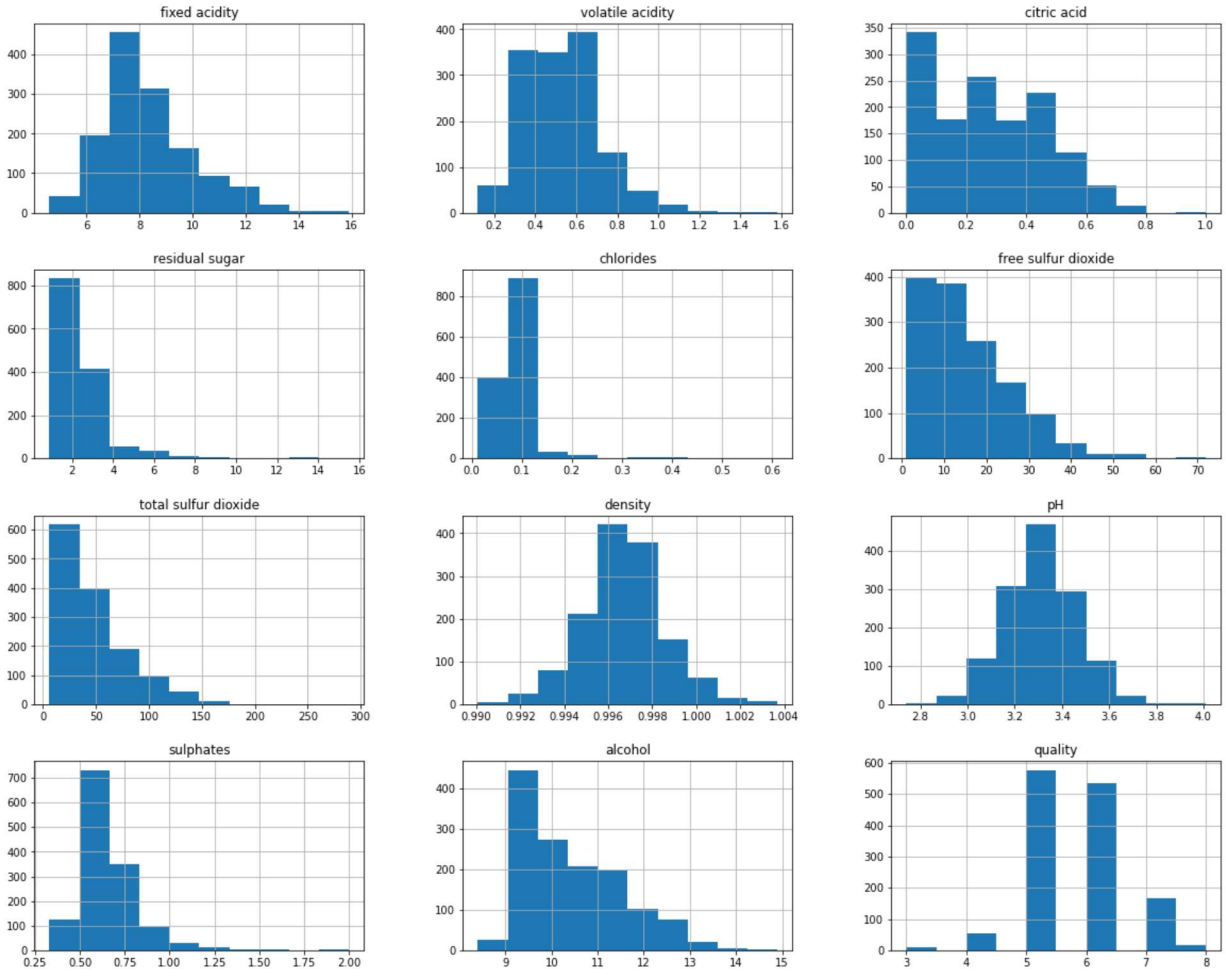
Out[26]:
```
['fixed acidity',
 'volatile acidity',
 'citric acid',
 'residual sugar',
 'chlorides',
 'free sulfur dioxide',
 'total sulfur dioxide',
 'density',
 'pH',
 'sulphates',
 'alcohol',
 'quality']
```

In [27]:
```
num_columns=[feature for feature in df.columns if df[feature].dtypes!='object']
num_columns
```

Out[27]:
```
['fixed acidity',
 'volatile acidity',
 'citric acid',
 'residual sugar',
 'chlorides',
 'free sulfur dioxide',
 'total sulfur dioxide',
 'density',
 'pH',
 'sulphates',
 'alcohol',
 'quality']
```

In [28]:
```
df.hist(figsize=(20,16))
```

Out[28]:
```
array([[<AxesSubplot:title={'center':'fixed acidity'}>,
        <AxesSubplot:title={'center':'volatile acidity'}>,
        <AxesSubplot:title={'center':'citric acid'}>],
       [<AxesSubplot:title={'center':'residual sugar'}>,
        <AxesSubplot:title={'center':'chlorides'}>,
        <AxesSubplot:title={'center':'free sulfur dioxide'}>],
       [<AxesSubplot:title={'center':'total sulfur dioxide'}>,
        <AxesSubplot:title={'center':'density'}>,
        <AxesSubplot:title={'center':'pH'}>],
       [<AxesSubplot:title={'center':'sulphates'}>,
        <AxesSubplot:title={'center':'alcohol'}>,
        <AxesSubplot:title={'center':'quality'}>]], dtype=object)
```



In [29]:
```
# discrete columns
num_columns_discrete=[feature for feature in df.columns if df[feature].dtype=='in
num_columns_discrete
```

Out[29]:
```
['quality']
```

In [30]:
```
# continuous columns
num_columns_contin=[feature for feature in df.columns if df[feature].dtype=='floa
num_columns_contin
```

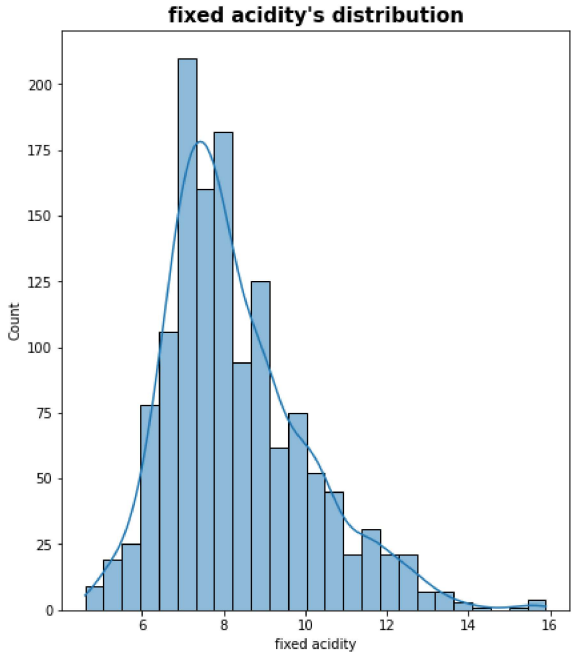
Out[30]:
```
['fixed acidity',
 'volatile acidity',
 'citric acid',
 'residual sugar',
 'chlorides',
 'free sulfur dioxide',
 'total sulfur dioxide',
 'density',
 'pH',
 'sulphates',
 'alcohol']
```

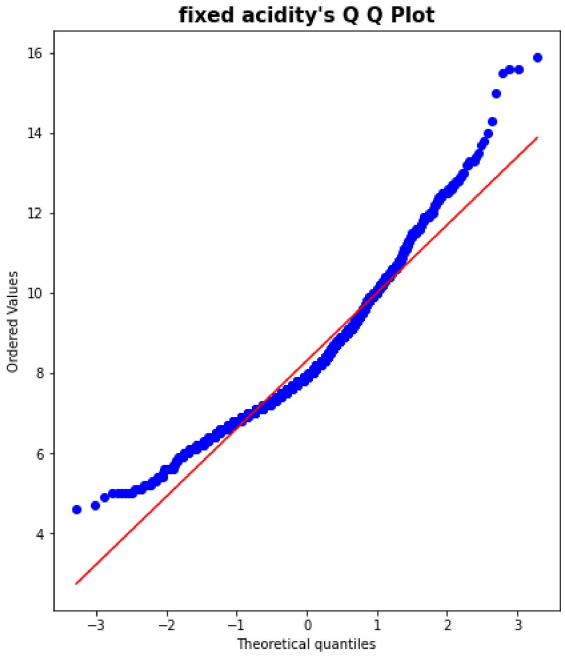
In [31]:
```
# indicate the Categorical columns

cat_columns=[feature for feature in df.columns if df[feature].dtype=='object']
cat_columns
```

Out[31]:
```
[]
```

In [32]:
```
import scipy.stats as stats
```

In [33]:
```
# Checking the distribution of Continous Feature
for feature in num_columns_contin:
    plt.figure(figsize=(15,8))
    plt.subplot(121)
    sns.histplot(data=df,x=feature ,kde=True , bins=25)
    plt.title("{}'s distribution".format(feature),fontweight='bold',fontsize=15)

    plt.subplot(122)
    stats.probplot(df[feature],dist='norm',plot=plt)
    plt.title("{}'s Q Q Plot".format(feature),fontweight='bold',fontsize=15)
    plt.show();
```





In [ ]:
```
```

In [34]:
```
#Statical Based Analysis
df.describe()
```

Out[34]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide
count	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000
mean	8.310596	0.529478	0.272333	2.523400	0.088124	15.893304	46.825975
std	1.736990	0.183031	0.195537	1.352314	0.049377	10.447270	33.408946
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000
75%	9.200000	0.640000	0.430000	2.600000	0.091000	21.000000	63.000000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000

In [35]:
```
df.describe().T
```

Out[35]:

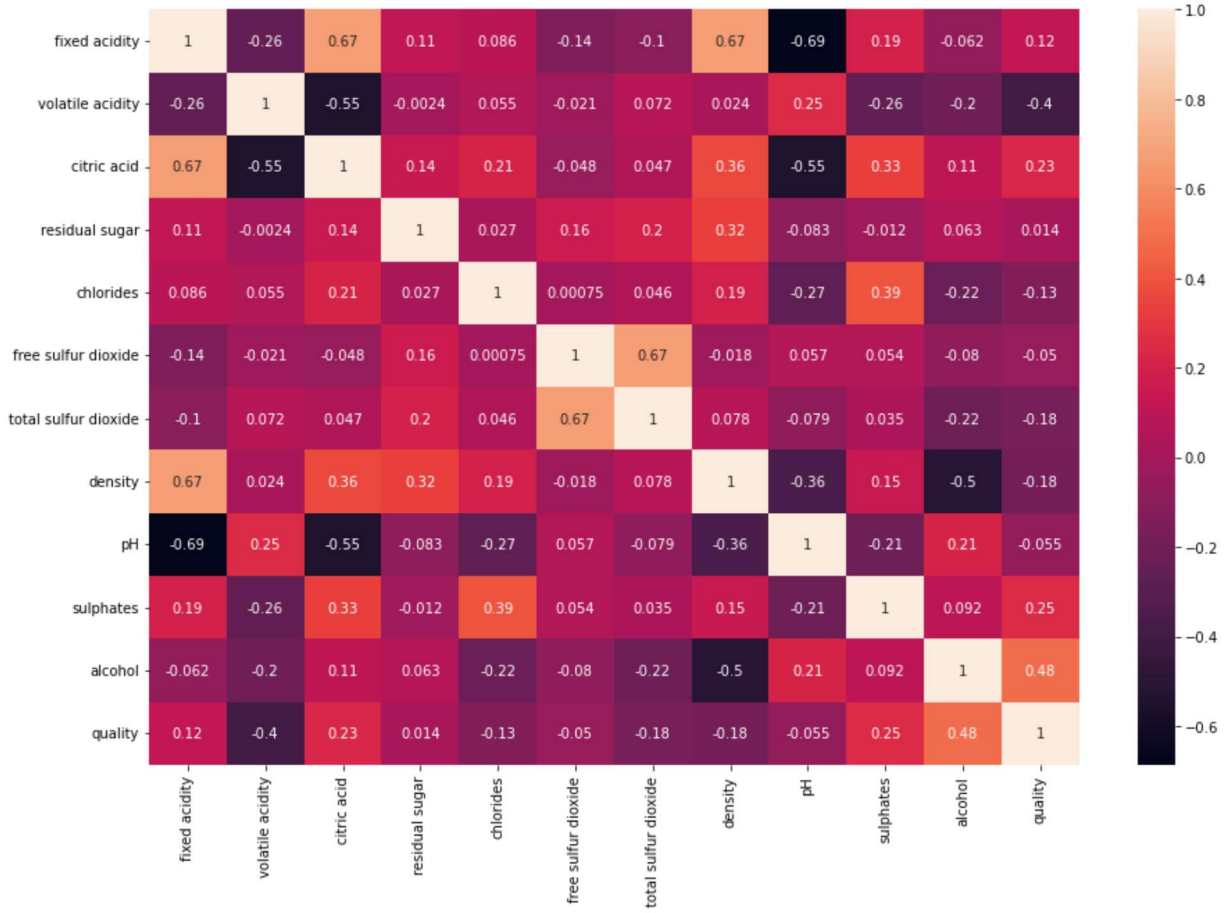
	count	mean	std	min	25%	50%	75%	max
fixed acidity	1359.0	8.310596	1.736990	4.60000	7.1000	7.9000	9.20000	15.90000
volatile acidity	1359.0	0.529478	0.183031	0.12000	0.3900	0.5200	0.64000	1.58000
citric acid	1359.0	0.272333	0.195537	0.00000	0.0900	0.2600	0.43000	1.00000
residual sugar	1359.0	2.523400	1.352314	0.90000	1.9000	2.2000	2.60000	15.50000
chlorides	1359.0	0.088124	0.049377	0.01200	0.0700	0.0790	0.09100	0.61100
free sulfur dioxide	1359.0	15.893304	10.447270	1.00000	7.0000	14.0000	21.00000	72.00000
total sulfur dioxide	1359.0	46.825975	33.408946	6.00000	22.0000	38.0000	63.00000	289.00000
density	1359.0	0.996709	0.001869	0.99007	0.9956	0.9967	0.99782	1.00369
pH	1359.0	3.309787	0.155036	2.74000	3.2100	3.3100	3.40000	4.01000
sulphates	1359.0	0.658705	0.170667	0.33000	0.5500	0.6200	0.73000	2.00000
alcohol	1359.0	10.432315	1.082065	8.40000	9.5000	10.2000	11.10000	14.90000
quality	1359.0	5.623252	0.823578	3.00000	5.0000	6.0000	6.00000	8.00000

In [36]:
```
# check correlartion
df.corr()
```

Out[36]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	
fixed acidity	1.000000	-0.255124	0.667437	0.111025	0.085886	-0.140580	-0.103777	0.670195	-(
volatile acidity	-0.255124	1.000000	-0.551248	-0.002449	0.055154	-0.020945	0.071701	0.023943	(
citric acid	0.667437	-0.551248	1.000000	0.143892	0.210195	-0.048004	0.047358	0.357962	-(
residual sugar	0.111025	-0.002449	0.143892	1.000000	0.026656	0.160527	0.201038	0.324522	-(
chlorides	0.085886	0.055154	0.210195	0.026656	1.000000	0.000749	0.045773	0.193592	-(
free sulfur dioxide	-0.140580	-0.020945	-0.048004	0.160527	0.000749	1.000000	0.667246	-0.018071	(
total sulfur dioxide	-0.103777	0.071701	0.047358	0.201038	0.045773	0.667246	1.000000	0.078141	-(
density	0.670195	0.023943	0.357962	0.324522	0.193592	-0.018071	0.078141	1.000000	-(
pH	-0.686685	0.247111	-0.550310	-0.083143	-0.270893	0.056631	-0.079257	-0.355617	1
sulphates	0.190269	-0.256948	0.326062	-0.011837	0.394557	0.054126	0.035291	0.146036	-(
alcohol	-0.061596	-0.197812	0.105108	0.063281	-0.223824	-0.080125	-0.217829	-0.504995	(
quality	0.119024	-0.395214	0.228057	0.013640	-0.130988	-0.050463	-0.177855	-0.184252	-(

In [37]:
```
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(),annot=True)
plt.show()
```



In [38]:
```
# check varianec
df.var()
```

Out[38]:
```
fixed acidity             3.017134
volatile acidity          0.033500
citric acid               0.038235
residual sugar            1.828752
chlorides                 0.002438
free sulfur dioxide     109.145456
total sulfur dioxide   1116.157653
density                   0.000003
pH                        0.024036
sulphates                 0.029127
alcohol                   1.170866
quality                   0.678281
dtype: float64
```

In [39]:
```
#indicate  standard deviation
df.std()
```

Out[39]:
```
fixed acidity            1.736990
volatile acidity         0.183031
citric acid              0.195537
residual sugar           1.352314
chlorides                0.049377
free sulfur dioxide     10.447270
total sulfur dioxide    33.408946
density                  0.001869
pH                       0.155036
sulphates                0.170667
alcohol                  1.082065
quality                  0.823578
dtype: float64
```

In [40]:
```
# check quantile()
df.quantile()
```

Out[40]:
```
fixed acidity            7.9000
volatile acidity         0.5200
citric acid              0.2600
residual sugar           2.2000
chlorides                0.0790
free sulfur dioxide     14.0000
total sulfur dioxide    38.0000
density                  0.9967
pH                       3.3100
sulphates                0.6200
alcohol                 10.2000
quality                  6.0000
Name: 0.5, dtype: float64
```

In [41]:
```
# check the minimum values
df.min()
```

Out[41]:
```
fixed acidity           4.60000
volatile acidity        0.12000
citric acid             0.00000
residual sugar          0.90000
chlorides               0.01200
free sulfur dioxide     1.00000
total sulfur dioxide    6.00000
density                 0.99007
pH                      2.74000
sulphates               0.33000
alcohol                 8.40000
quality                 3.00000
dtype: float64
```

In [42]:
```
# check the maximum values
df.max()
```

Out[42]:
```
fixed acidity           15.90000
volatile acidity         1.58000
citric acid              1.00000
residual sugar          15.50000
chlorides                0.61100
free sulfur dioxide     72.00000
total sulfur dioxide   289.00000
density                  1.00369
pH                       4.01000
sulphates                2.00000
alcohol                 14.90000
quality                  8.00000
dtype: float64
```

In [43]:
```
# check mean()
df.mean()
```

Out[43]:
```
fixed acidity           8.310596
volatile acidity        0.529478
citric acid             0.272333
residual sugar          2.523400
chlorides               0.088124
free sulfur dioxide    15.893304
total sulfur dioxide   46.825975
density                 0.996709
pH                      3.309787
sulphates               0.658705
alcohol                10.432315
quality                 5.623252
dtype: float64
```

In [44]:
```
df.median()
```

Out[44]:
```
fixed acidity           7.9000
volatile acidity        0.5200
citric acid             0.2600
residual sugar          2.2000
chlorides               0.0790
free sulfur dioxide    14.0000
total sulfur dioxide   38.0000
density                 0.9967
pH                      3.3100
sulphates               0.6200
alcohol                10.2000
quality                 6.0000
dtype: float64
```

In [45]:
```
df.mode()
```

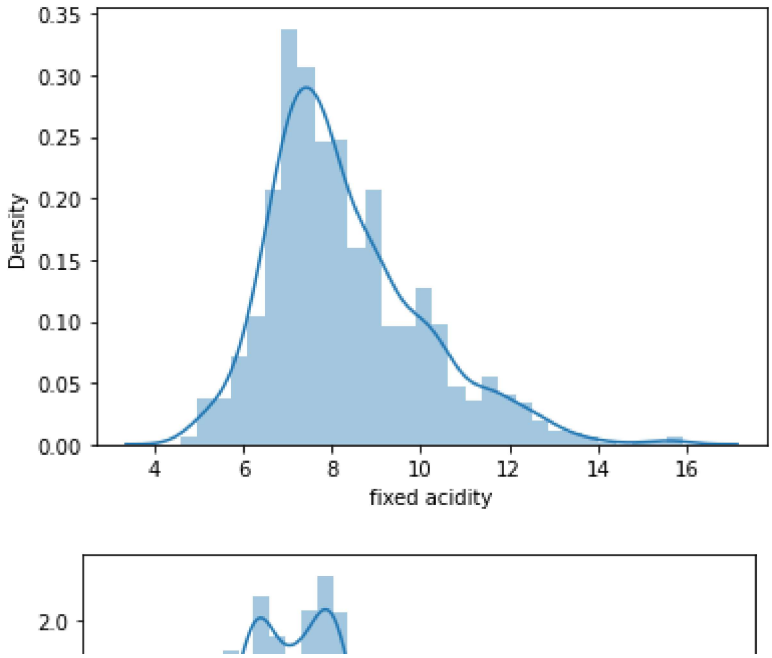
Out[45]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	q
0	7.2	0.5	0.0	2.0	0.08	6.0	28.0	0.9968	3.3	0.54	9.5	

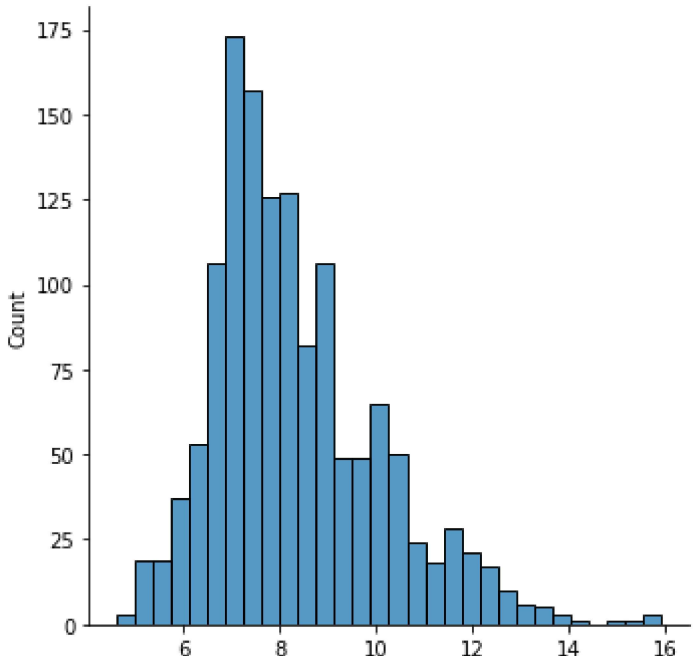
In [46]:
```
# check skewness
df.skew()
```

Out[46]:
```
fixed acidity           0.941041
volatile acidity        0.729279
citric acid             0.312726
residual sugar          4.548153
chlorides               5.502487
free sulfur dioxide     1.226579
total sulfur dioxide    1.540368
density                 0.044778
pH                      0.232032
sulphates               2.406505
alcohol                 0.859841
quality                 0.192407
dtype: float64
```

In [47]:
```
for i in num_columns:
    sns.distplot(df[i])
    plt.show()
```



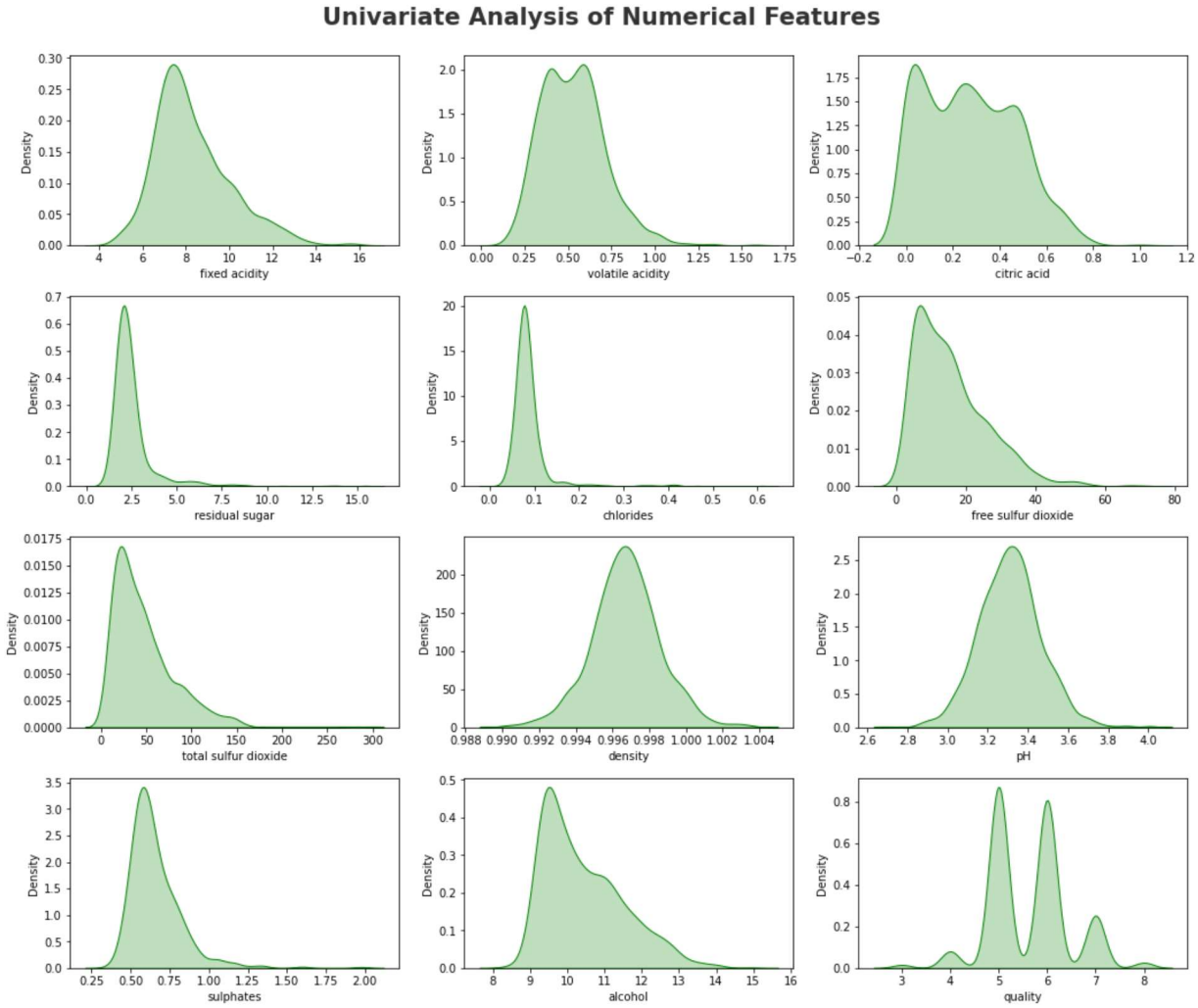
In [48]:
```
for i in num_columns:
    sns.displot(df[i])

    plt.show()
```



Univariate Analysis of Numerical feature

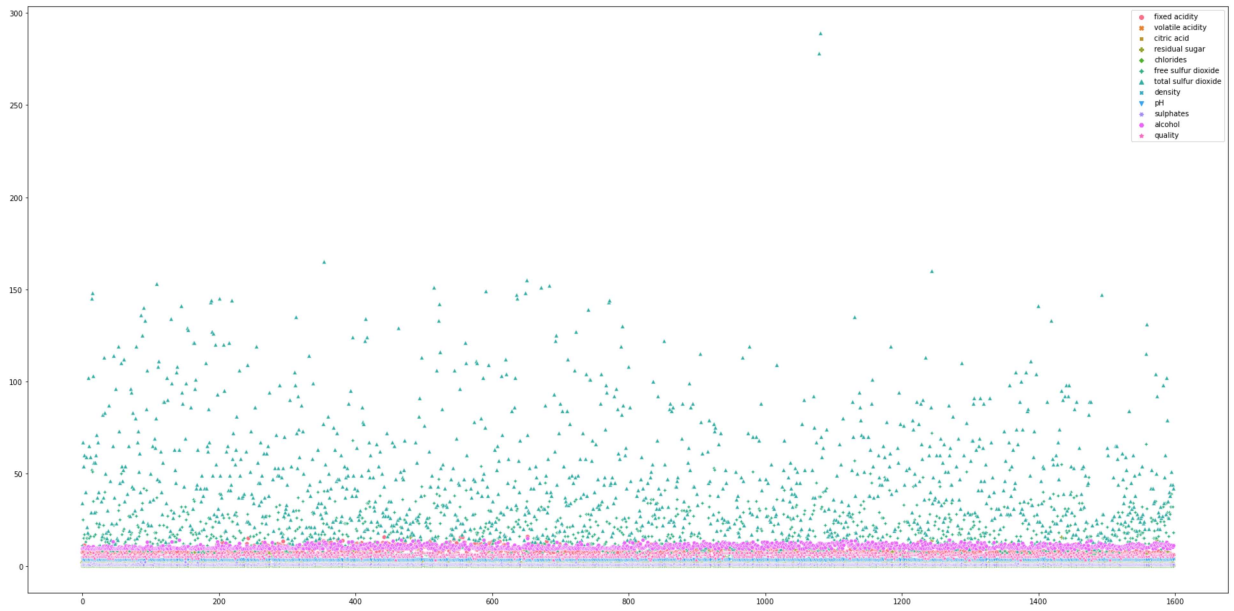
In [49]:

```
plt.figure(figsize=(15,15))
plt.suptitle('Univariate Analysis of Numerical Features', fontsize =21 ,fontweigh
for i in range(0, len(num_columns)):
    plt.subplot(5, 3, i+1)
    sns.kdeplot(x=df[num_columns[i]],shade=True, color='g')
    plt.xlabel(num_columns[i])
    plt.tight_layout()
```



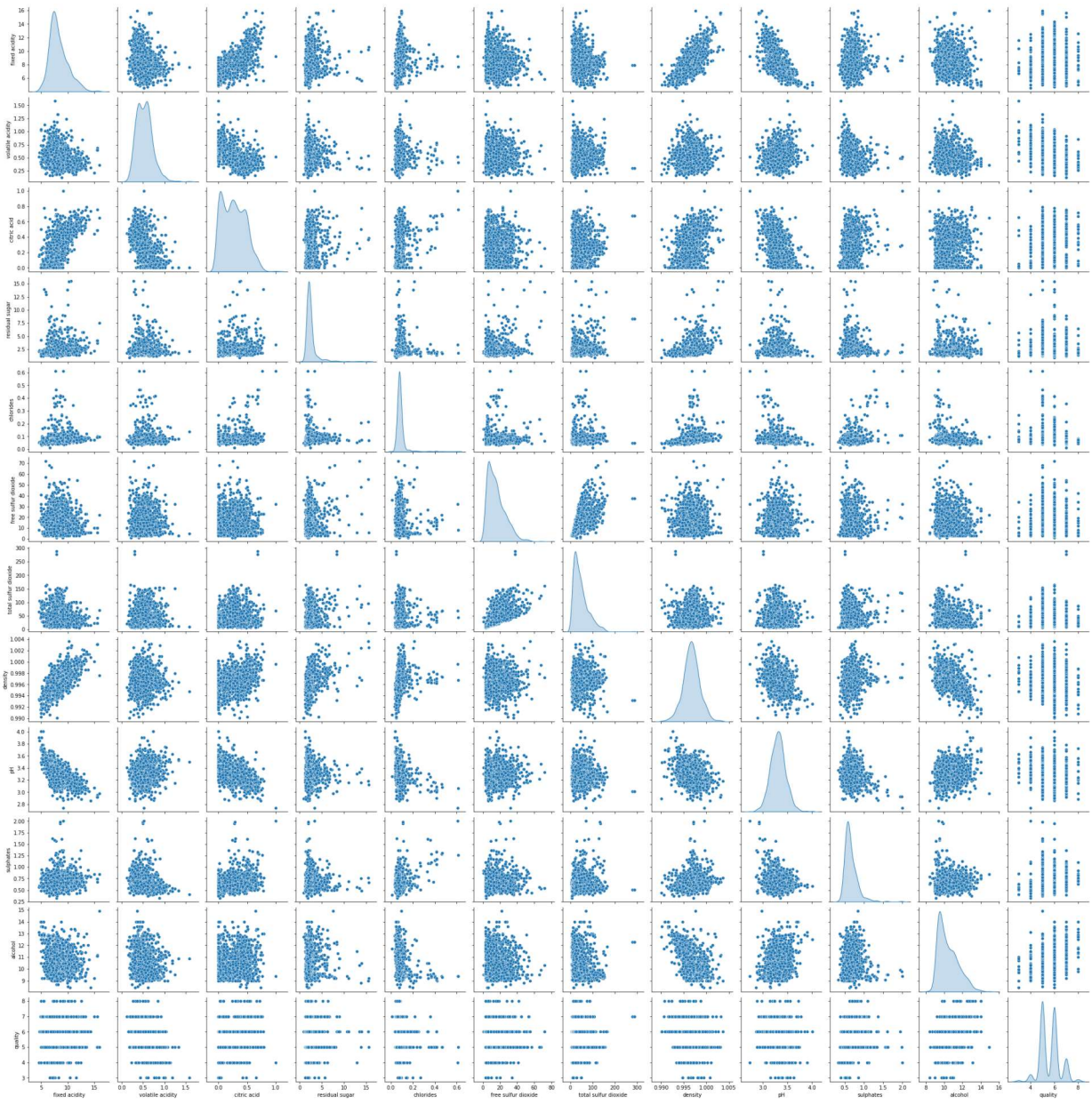
In [50]:
```
# check the scatter plot
plt.figure(figsize=(30,15))
sns.scatterplot(data=df)
```

Out[50]: <AxesSubplot:>



In [51]:
```
# indiacte the pairplot
sns.pairplot(df,diag_kind='kde')
```

Out[51]: <seaborn.axisgrid.PairGrid at 0x22c7acee400>

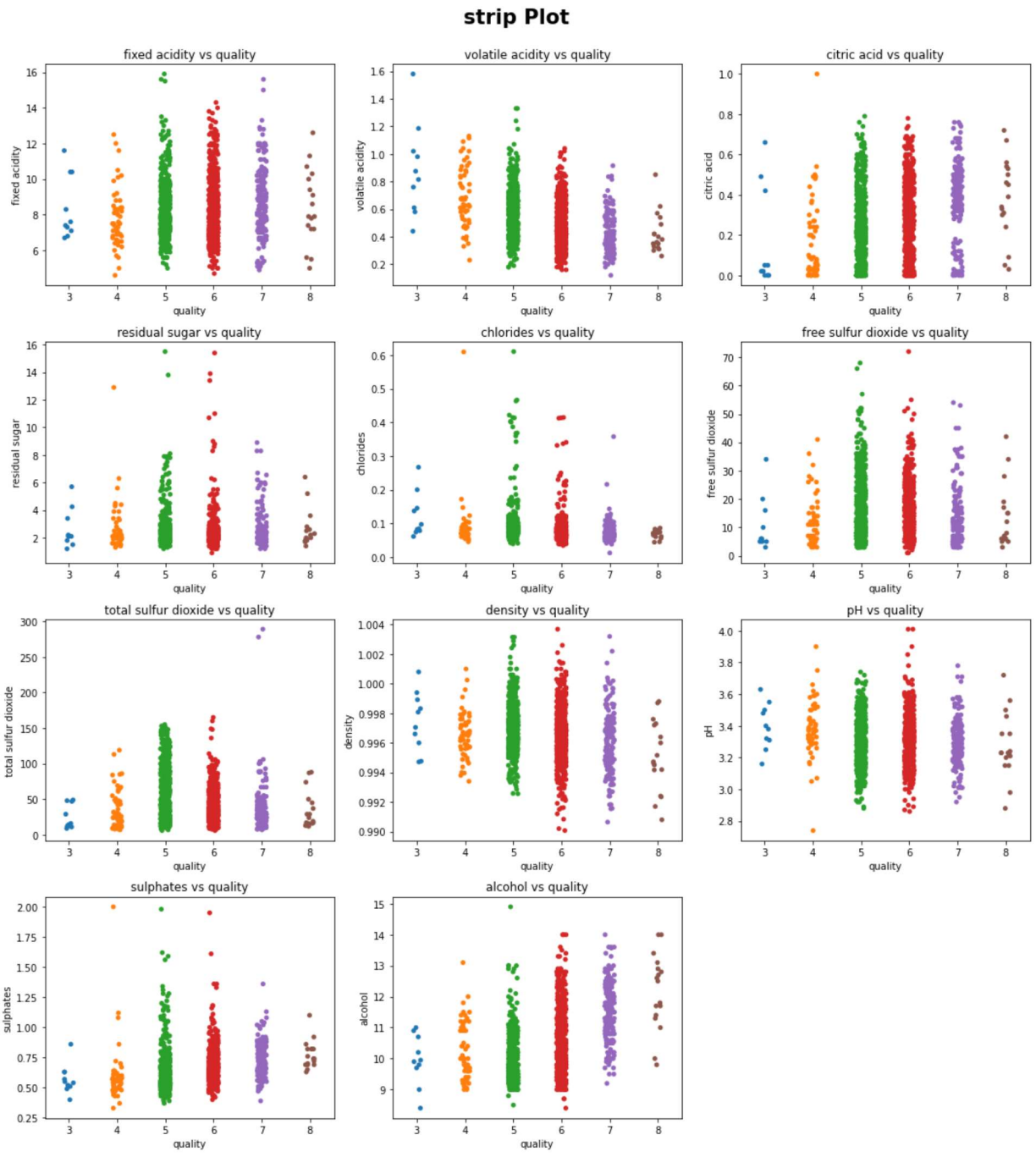


which alcohol has largest number of quality

In [52]:
```
df.groupby(['alcohol'])['quality'].sum().sort_values(ascending=False)
```

Out[52]:
```
alcohol
9.50     592
9.40     472
10.00    339
9.20     338
9.80     334
         ...
8.50       5
8.80       5
14.90      5
9.05       4
9.95       3
Name: quality, Length: 65, dtype: int64
```

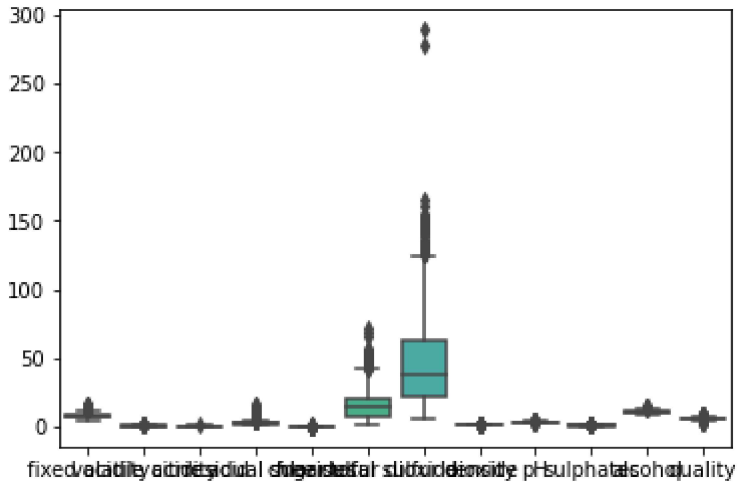
In [53]:
```
#Visualising data scatter in each continous feature with respect to quality
plt.figure(figsize=(15,20))
plt.suptitle('strip Plot', fontsize =21 , fontweight ='bold' , alpha = 1 , y= 1)
for i in range (0 , len(num_columns_contin)):
    plt.subplot(5 , 3 ,i+1)
    sns.stripplot(y= num_columns_contin[i], x='quality' , data=df)
    plt.title('{} vs quality'.format(num_columns_contin[i],fontsize=15))
    plt.tight_layout()
```



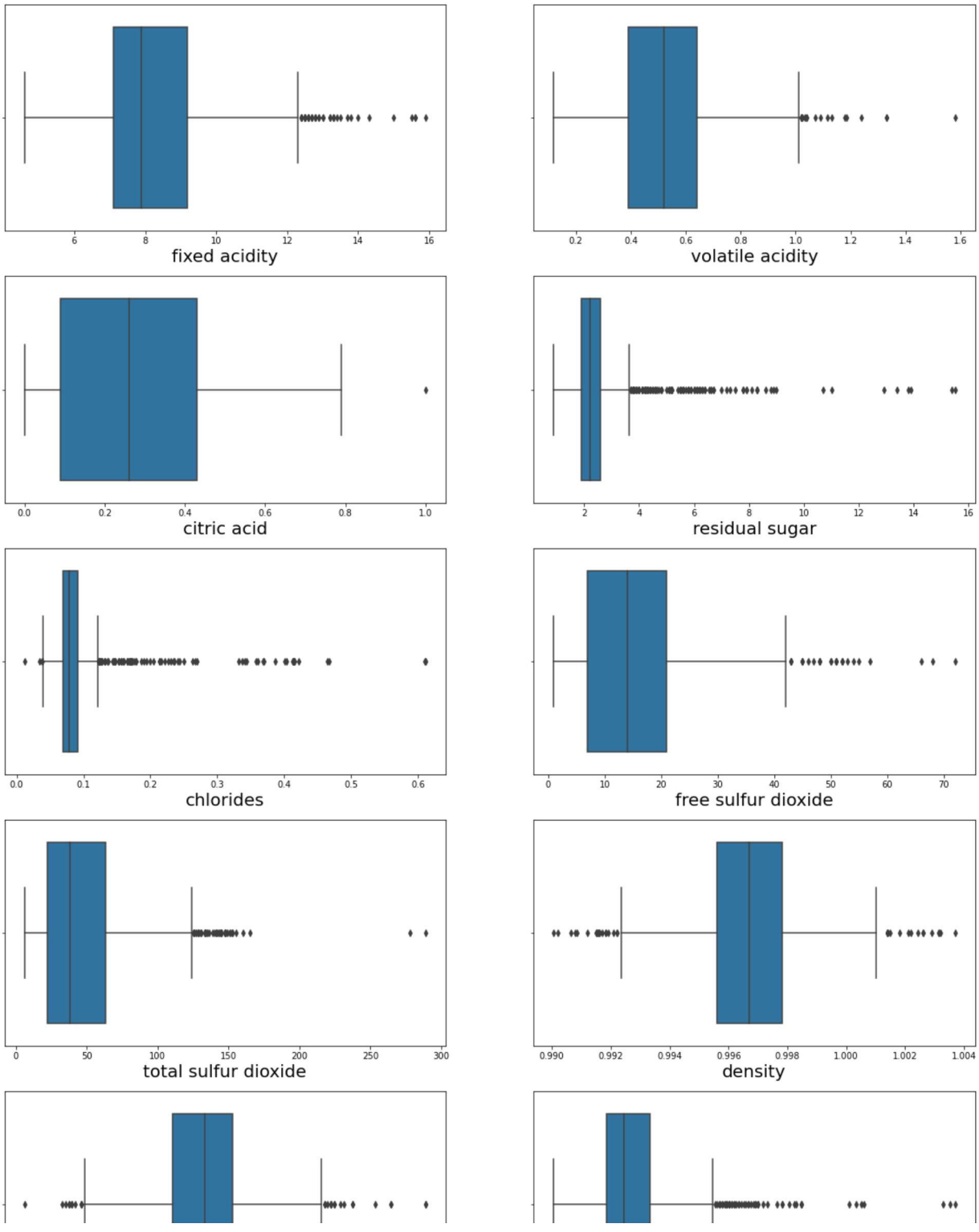
Box plot

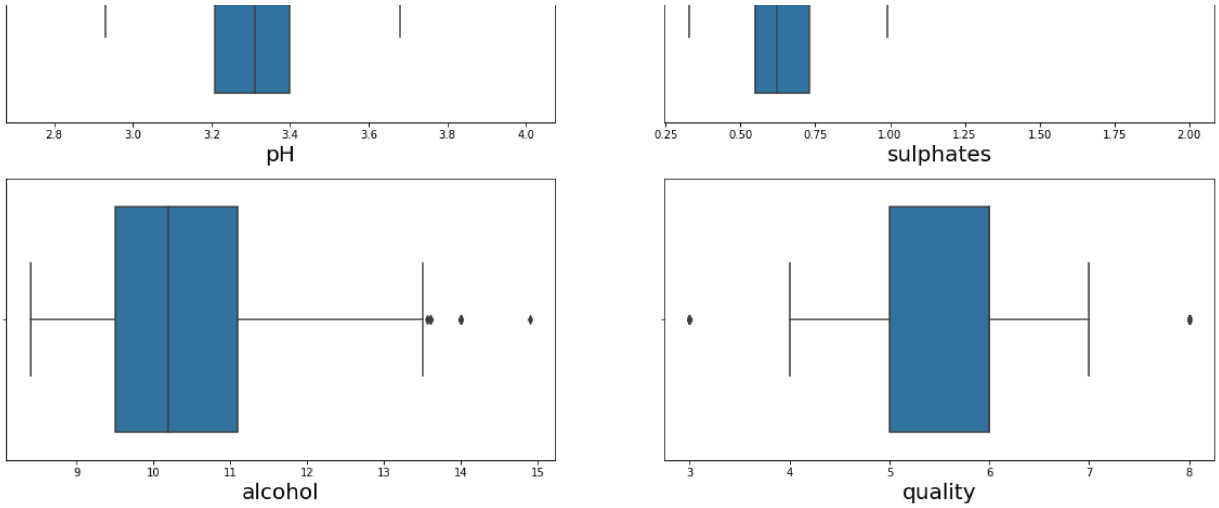
In [54]:
```
sns.boxplot(data=df)
```

Out[54]: <AxesSubplot:>



In [55]:
```
plt.figure(figsize=(20,45))
plotnumber=1

for i in num_columns:
    if plotnumber<=12:
        ax=plt.subplot(8,2,plotnumber)
        sns.boxplot(df[i])
        plt.xlabel(i,fontsize=20)

    plotnumber+=1
plt.show()
```





Droping the Outliers

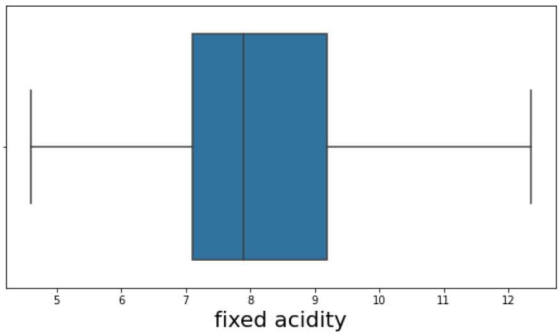
In [56]:
```
def outliers_imputation_mild(data,column):
    IQR=data[column].quantile(0.75)-data[column].quantile(0.25)
    lower_fence=data[column].quantile(0.25)-(IQR*1.5)
    upper_fence=data[column].quantile(0.75)+(IQR*1.5)
    print("IQR:",IQR)
    print(f"Lower Fence{column}:",lower_fence)
    print(f"Upper Fence{column}:",upper_fence)
    print('_____________________________________________')
    data.loc[data[column]<=lower_fence,column]=lower_fence
    data.loc[data[column]>=upper_fence,column]=upper_fence
```

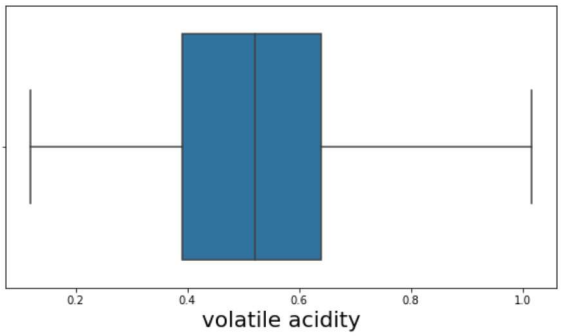
In [57]:
```
columns=df.columns
```

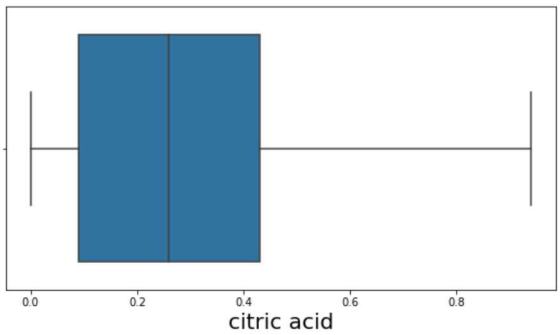
In [58]:
```
for i in columns:
    outliers_imputation_mild(df,i)
```

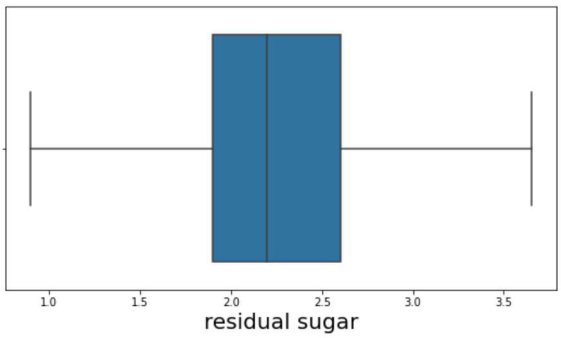
```
IQR: 2.0999999999999996
Lower Fencefixed acidity: 3.95
Upper Fencefixed acidity: 12.349999999999998

--------------------------------------------------
IQR: 0.25
Lower Fencevolatile acidity: 0.015000000000000013
Upper Fencevolatile acidity: 1.0150000000000001

--------------------------------------------------
IQR: 0.33999999999999997
Lower Fencecitric acid: -0.42000000000000004
Upper Fencecitric acid: 0.94

--------------------------------------------------
IQR: 0.7000000000000002
Lower Fenceresidual sugar: 0.8499999999999996
Upper Fenceresidual sugar: 3.6500000000000004

--------------------------------------------------
IQR: 0.02099999999999999
Lower Fencechlorides: 0.03850000000000002
Upper Fencechlorides: 0.12249999999999998

--------------------------------------------------
IQR: 14.0
Lower Fencefree sulfur dioxide: -14.0
Upper Fencefree sulfur dioxide: 42.0

--------------------------------------------------
IQR: 41.0
Lower Fencetotal sulfur dioxide: -39.5
Upper Fencetotal sulfur dioxide: 124.5

--------------------------------------------------
IQR: 0.0022199999999999998
Lower Fencedensity: 0.99227
Upper Fencedensity: 1.00115

--------------------------------------------------
IQR: 0.18999999999999995
Lower FencepH: 2.925
Upper FencepH: 3.6849999999999996

--------------------------------------------------
IQR: 0.17999999999999994
Lower Fencesulphates: 0.28000000000000014
Upper Fencesulphates: 0.9999999999999999

--------------------------------------------------
IQR: 1.5999999999999996
Lower Fencealcohol: 7.1000000000000005
Upper Fencealcohol: 13.5

--------------------------------------------------
IQR: 1.0
Lower Fencequality: 3.5
Upper Fencequality: 7.5

--------------------------------------------------
```

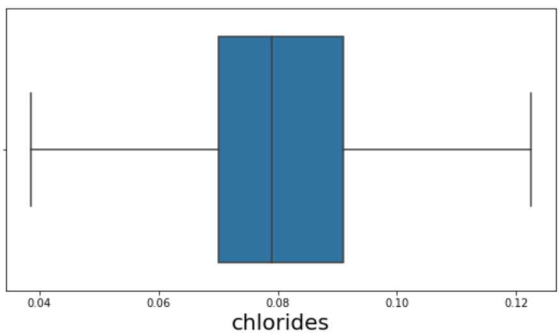
In [59]:
```
plt.figure(figsize=(20,45))
plotnumber=1

for i in num_columns:
    if plotnumber<=12:
        ax=plt.subplot(8,2,plotnumber)
        sns.boxplot(df[i])
        plt.xlabel(i,fontsize=20)

    plotnumber+=1
plt.show()
```

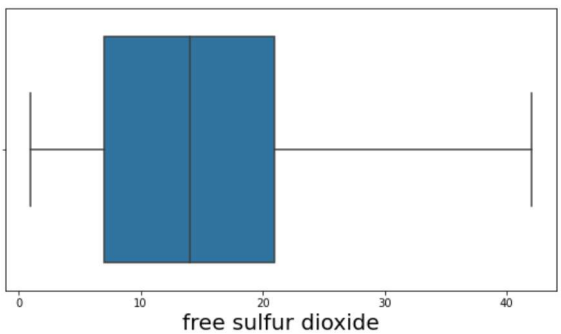


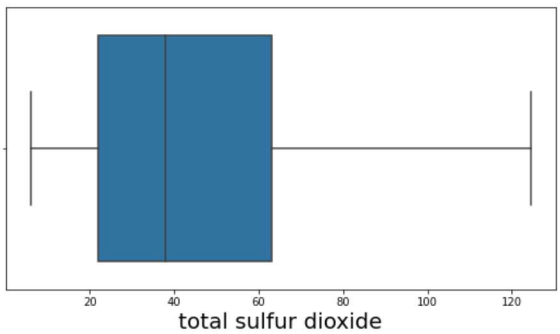


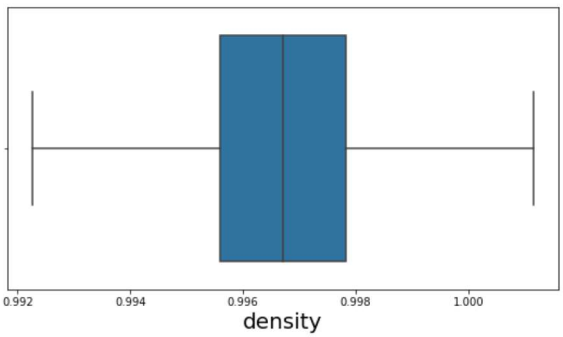


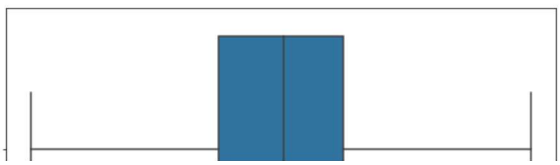


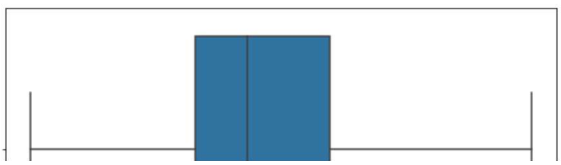


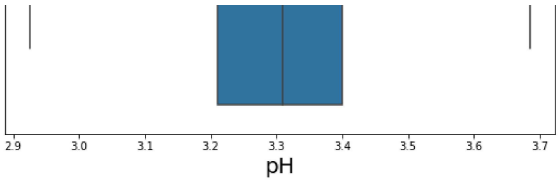


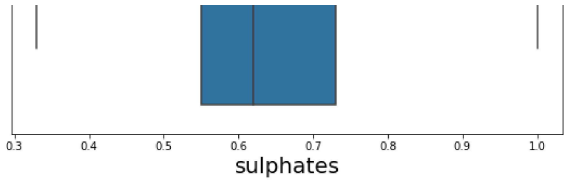


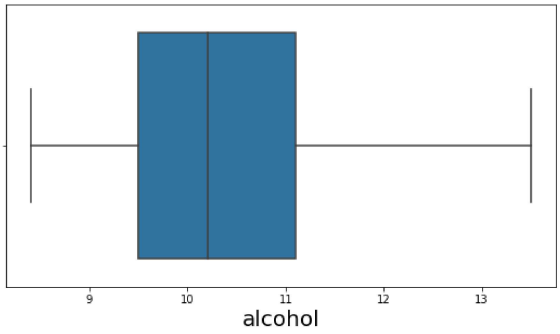


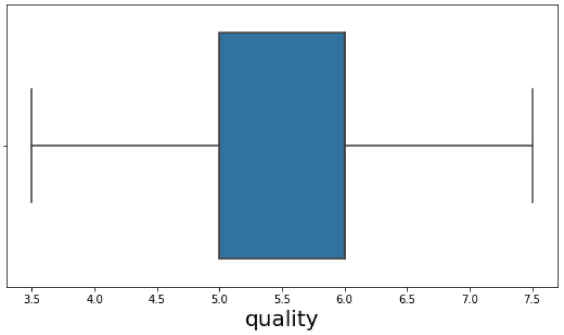





2.9
3.0
3.1
3.2
3.3
3.4
3.5
3.6
3.7
pH


0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
sulphates

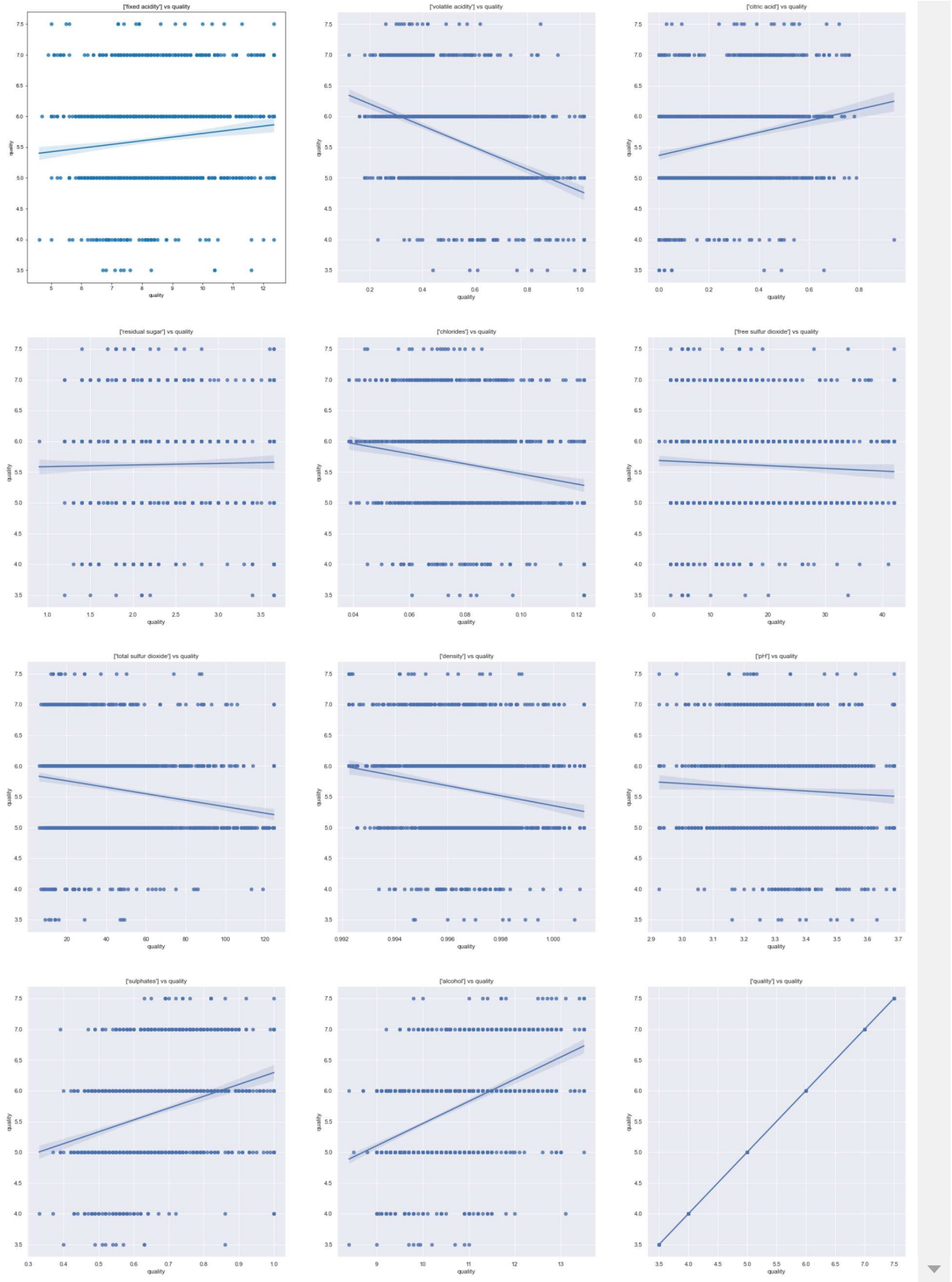

9
10
11
12
13
alcohol


3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
quality

Reg plot

In [60]:

```
plt.figure(figsize=(30,55))
for i in enumerate(num_columns):
    plt.subplot(5,3,i[0]+1)
    sns.set(rc={'figure.figsize':(8,9)})
    sns.regplot(data=df,x=i[1],y='quality')
    plt.xlabel('quality')
    plt.title("{} vs quality".format([i[1]]))
```



In [61]:
```
X=df.drop("quality",axis=1)
```

In [62]:
```
y=df['quality']
```

Train_Test_Split

In [100]:
```
from sklearn.model_selection import train_test_split,GridSearchCV
```

In [ ]:
```

```

In [101]:
```
y.head()
```

Out[101]:
```
0    5
1    5
2    5
3    6
5    5
Name: quality, dtype: int32
```

In [102]:
```
y.dtype
```

Out[102]:
```
dtype('int32')
```

In [103]:
```
y.astype(int)
```

Out[103]:
```
0       5
1       5
2       5
3       6
5       5
       ..
1593    6
1594    5
1595    6
1597    5
1598    6
Name: quality, Length: 1359, dtype: int32
```

In [104]:
```
y=y.astype(int)
```

In [105]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.33,random_state=10
```

In [106]:
```
X_train.shape , y_train.shape
```

Out[106]:
```
((910, 11), (910,))
```

In [107]:
```
X_test.shape , y_test.shape
```

Out[107]:
```
((449, 11), (449,))
```

Loggistic Regression

In [169]:
```
from sklearn.linear_model import LogisticRegression
```

In [172]:
```
classifier=LogisticRegression()
classifier
```

Out[172]:
```
LogisticRegression
LogisticRegression()
```

In [175]:
```
classifier.fit(X_train,y_train)
```

Out[175]:
```
LogisticRegression
LogisticRegression()
```

In [177]:
```
classifier.score(X_train,y_train)
```

Out[177]:
```
0.5967032967032967
```

In [179]:
```
y_predict=model.predict(X_test)
```

In [180]:
```
y_predict
```

Out[180]:
```
array([7, 6, 6, 5, 6, 7, 6, 7, 5, 7, 5, 5, 6, 5, 5, 5, 5, 5, 6, 5, 6, 6,
       7, 5, 5, 5, 5, 6, 5, 5, 6, 6, 6, 5, 6, 5, 5, 4, 7, 4, 5, 6, 6, 6,
       5, 6, 5, 5, 5, 6, 6, 5, 5, 6, 5, 7, 6, 7, 5, 5, 5, 6, 7, 5, 6, 6,
       5, 6, 6, 5, 5, 5, 5, 6, 4, 3, 7, 5, 5, 6, 6, 5, 5, 5, 5, 5, 6, 7,
       5, 6, 5, 5, 5, 6, 6, 5, 5, 4, 5, 6, 6, 5, 4, 5, 5, 5, 6, 6, 5, 5,
       5, 5, 7, 5, 5, 6, 7, 6, 5, 6, 4, 7, 5, 6, 6, 6, 7, 5, 5, 5, 6, 5,
       5, 5, 7, 5, 5, 6, 4, 7, 5, 6, 7, 6, 5, 5, 5, 6, 5, 7, 5, 6, 6, 5,
       5, 6, 4, 5, 6, 6, 5, 6, 5, 5, 5, 5, 6, 5, 5, 6, 7, 6, 6, 5, 5, 5,
       7, 6, 6, 5, 6, 6, 6, 5, 7, 5, 5, 7, 5, 5, 7, 5, 6, 5, 5, 5, 6, 5,
       7, 7, 5, 6, 5, 6, 6, 5, 6, 4, 6, 6, 6, 5, 7, 7, 6, 5, 5, 5, 7, 7,
       5, 6, 5, 5, 5, 6, 7, 5, 5, 6, 5, 5, 5, 7, 6, 5, 5, 6, 5, 4, 5, 5,
       4, 6, 7, 7, 5, 6, 6, 5, 6, 4, 7, 6, 6, 7, 7, 6, 6, 7, 6, 7, 5, 6,
       6, 6, 7, 7, 5, 6, 6, 6, 7, 5, 6, 6, 7, 7, 7, 6, 7, 5, 5, 6, 5, 6,
       4, 6, 5, 6, 5, 6, 5, 7, 6, 6, 5, 6, 7, 5, 6, 6, 5, 6, 6, 7, 5, 6,
       6, 5, 5, 6, 6, 5, 7, 6, 5, 4, 6, 6, 4, 7, 7, 6, 6, 5, 5, 5, 7, 5,
       6, 5, 7, 6, 6, 6, 5, 5, 5, 5, 7, 5, 6, 7, 5, 5, 6, 7, 7, 7, 6, 7,
       6, 6, 6, 5, 5, 5, 6, 6, 5, 5, 6, 6, 6, 6, 6, 7, 6, 5, 6, 7, 6, 5,
       6, 7, 5, 6, 5, 5, 6, 6, 5, 5, 5, 6, 5, 6, 5, 5, 7, 6, 5, 5, 5, 5,
       6, 5, 7, 6, 5, 5, 6, 4, 6, 6, 5, 4, 6, 6, 5, 6, 5, 5, 6, 6, 6, 5,
       5, 6, 5, 5, 6, 5, 5, 7, 6, 5, 6, 6, 5, 5, 6, 7, 5, 6, 6, 6, 6, 6,
       5, 5, 6, 6, 6, 7, 5, 5, 6])
```

In [181]:
```
from sklearn.metrics import accuracy_score
```

In [182]:
```
accuracy_score(y_test,y_predict)
```

Out[182]:
```
0.5278396436525612
```

Support Vector Classifier

In [184]:
```
from sklearn.svm import SVC
```

In [186]:
```
svc=SVC()
svc
```

Out[186]:
```
SVC()
```

In [187]:
```
svc.fit(X_train,y_train)
```

Out[187]:
```
SVC()
```

In [188]:
```
svc.score(X_train,y_train)
```

Out[188]:
```
0.5098901098901099
```

In [189]:
```
svc_predict=svc.predict(X_test)
```

In [190]:
```
svc_predict
```

Out[190]:
```
array([6, 6, 6, 5, 6, 6, 6, 6, 6, 5, 5, 5, 6, 5, 6, 6, 6, 6, 5, 6, 6, 6,
       6, 6, 6, 5, 6, 5, 5, 6, 6, 6, 6, 5, 6, 6, 6, 5, 6, 6, 6, 6, 6, 6,
       6, 6, 6, 6, 5, 6, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
       5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 5, 6, 6, 5, 6, 6, 6, 6, 6, 6,
       6, 6, 6, 5, 5, 5, 6, 6, 5, 5, 5, 6, 6, 5, 5, 6, 6, 6, 6, 6, 5, 6,
       6, 6, 6, 5, 5, 6, 6, 6, 5, 6, 6, 6, 5, 6, 6, 6, 6, 6, 5, 6, 6, 6,
       5, 5, 6, 5, 6, 6, 6, 6, 5, 6, 6, 6, 6, 6, 5, 6, 6, 6, 6, 6, 6, 6,
       6, 6, 5, 5, 6, 6, 5, 6, 5, 5, 5, 5, 5, 6, 5, 5, 6, 6, 6, 6, 5, 6,
       6, 6, 6, 6, 6, 5, 6, 5, 6, 6, 5, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6, 6,
       6, 6, 6, 5, 6, 6, 6, 6, 6, 5, 6, 5, 6, 6, 6, 6, 5, 6, 5, 6, 6, 6,
       5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6, 6, 5, 6, 6, 6, 5, 6,
       6, 6, 6, 6, 5, 6, 6, 6, 6, 5, 6, 6, 6, 6, 6, 5, 6, 6, 6, 6, 6, 6,
       6, 5, 6, 6, 5, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6, 6, 6, 6, 6, 5, 5, 6,
       6, 6, 5, 6, 5, 5, 5, 6, 6, 6, 5, 6, 6, 5, 6, 6, 6, 6, 6, 6, 5, 5,
       6, 5, 5, 6, 6, 6, 5, 5, 6, 5, 6, 6, 6, 5, 5, 6, 6, 5, 6, 5, 6, 6,
       6, 5, 6, 6, 6, 6, 6, 5, 6, 6, 5, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6,
       6, 6, 6, 6, 6, 5, 5, 6, 6, 6, 6, 6, 6, 5, 5, 6, 6, 6, 5, 5, 6, 6,
       5, 6, 6, 6, 6, 6, 6, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 5,
       6, 5, 6, 6, 6, 5, 6, 5, 6, 6, 6, 5, 6, 5, 6, 6, 6, 6, 5, 5, 6, 6,
       5, 6, 6, 5, 6, 6, 6, 6, 6, 6, 5, 6, 6, 5, 6, 5, 6, 6, 5, 6, 6, 6,
       6, 6, 5, 6, 6, 6, 6, 6, 5])
```

In [191]:
```
from sklearn.metrics import accuracy_score
```

In [193]:
```
accuracy_score(y_test,svc_predict)
```

Out[193]:
```
0.4766146993318486
```

In [ ]:
```

```

Decision Tree Classifier

In [108]:
```
from sklearn.tree import DecisionTreeClassifier
model=DecisionTreeClassifier()
```

In [109]:
```
model
```

Out[109]:
```
DecisionTreeClassifier
DecisionTreeClassifier()
```

In [110]:
```
model.fit(X_train,y_train)
```

Out[110]:
```
DecisionTreeClassifier
DecisionTreeClassifier()
```

In [111]:
```
model.score(X_train,y_train)
```

Out[111]:
```
1.0
```

In [112]:
```
from sklearn import tree
import matplotlib.pyplot as plt
fig=plt.figure(figsize=(25,15))
tree.plot_tree(model,filled=True)
```

Out[112]:
```
[Text(0.4792114752024291, 0.9705882352941176, 'X[10] <= 10.15\ngini = 0.642\n
samples = 910\nvalue = [7, 31, 389, 359, 124]'),
 Text(0.21492282388663966, 0.9117647058823529, 'X[6] <= 62.5\ngini = 0.495\ns
amples = 459\nvalue = [3, 15, 298, 131, 12]'),
 Text(0.11266447368421052, 0.8529411764705882, 'X[1] <= 0.548\ngini = 0.571\n
samples = 282\nvalue = [3, 11, 151, 105, 12]'),
 Text(0.05788208502024292, 0.7941176470588235, 'X[9] <= 0.585\ngini = 0.596\n
samples = 136\nvalue = [1, 4, 52, 68, 11]'),
 Text(0.020242914979757085, 0.7352941176470589, 'X[7] <= 0.996\ngini = 0.512
\nsamples = 44\nvalue = [0, 4, 28, 12, 0]'),
 Text(0.008097165991902834, 0.6764705882352942, 'X[10] <= 9.125\ngini = 0.245
\nsamples = 14\nvalue = [0, 2, 12, 0, 0]'),
 Text(0.004048582995951417, 0.6176470588235294, 'gini = 0.0\nsamples = 1\nval
ue = [0, 1, 0, 0, 0]'),
 Text(0.012145748987854251, 0.6176470588235294, 'X[8] <= 3.425\ngini = 0.142
\nsamples = 13\nvalue = [0, 1, 12, 0, 0]'),
 Text(0.008097165991902834, 0.5588235294117647, 'gini = 0.0\nsamples = 11\nva
lue = [0, 0, 11, 0, 0]'),
 Text(0.016194331983805668, 0.5588235294117647, 'X[9] <= 0.56\ngini = 0.5\nsa
```

In [113]:
```
fig.savefig("decistion_tree_classifier.png")
```

In [114]:
```
y_predict=model.predict(X_test)
```

In [115]:
```
y_predict
```

Out[115]:
```
array([7, 6, 6, 5, 6, 7, 6, 7, 5, 7, 5, 5, 6, 5, 5, 5, 5, 5, 6, 5, 6, 6,
       7, 5, 5, 5, 5, 6, 5, 5, 6, 6, 6, 5, 6, 5, 5, 4, 7, 4, 5, 6, 6, 6,
       5, 6, 5, 5, 5, 6, 6, 5, 5, 6, 5, 7, 6, 7, 5, 5, 5, 6, 7, 5, 6, 6,
       5, 6, 6, 5, 5, 5, 5, 6, 4, 3, 7, 5, 5, 6, 6, 5, 5, 5, 5, 5, 6, 7,
       5, 6, 5, 5, 5, 6, 6, 5, 5, 4, 5, 6, 6, 5, 4, 5, 5, 5, 6, 6, 5, 5,
       5, 5, 7, 5, 5, 6, 7, 6, 5, 6, 4, 7, 5, 6, 6, 6, 7, 5, 5, 5, 6, 5,
       5, 5, 7, 5, 5, 6, 4, 7, 5, 6, 7, 6, 5, 5, 5, 6, 5, 7, 5, 6, 6, 5,
       5, 6, 4, 5, 6, 6, 5, 6, 5, 5, 5, 5, 6, 5, 5, 6, 7, 6, 6, 5, 5, 5,
       7, 6, 6, 5, 6, 6, 6, 5, 7, 5, 5, 7, 5, 5, 7, 5, 6, 5, 5, 5, 6, 5,
       7, 7, 5, 6, 5, 6, 6, 5, 6, 4, 6, 6, 6, 5, 7, 7, 6, 5, 5, 5, 7, 7,
       5, 6, 5, 5, 5, 6, 7, 5, 5, 6, 5, 5, 5, 7, 6, 5, 5, 6, 5, 4, 5, 5,
       4, 6, 7, 7, 5, 6, 6, 5, 6, 4, 7, 6, 6, 7, 7, 6, 6, 7, 6, 7, 5, 6,
       6, 6, 7, 7, 5, 6, 6, 6, 7, 5, 6, 6, 7, 7, 7, 6, 7, 5, 5, 6, 5, 6,
       4, 6, 5, 6, 5, 6, 5, 7, 6, 6, 5, 6, 7, 5, 6, 6, 5, 6, 6, 7, 5, 6,
       6, 5, 5, 6, 6, 5, 7, 6, 5, 4, 6, 6, 4, 7, 7, 6, 6, 5, 5, 5, 7, 5,
       6, 5, 7, 6, 6, 6, 5, 5, 5, 5, 7, 5, 6, 7, 5, 5, 6, 7, 7, 7, 6, 7,
       6, 6, 6, 5, 5, 5, 6, 6, 5, 5, 6, 6, 6, 6, 6, 7, 6, 5, 6, 7, 6, 5,
       6, 7, 5, 6, 5, 5, 6, 6, 5, 5, 5, 6, 5, 6, 5, 5, 7, 6, 5, 5, 5, 5,
       6, 5, 7, 6, 5, 5, 6, 4, 6, 6, 5, 4, 6, 6, 5, 6, 5, 5, 6, 6, 6, 5,
       5, 6, 5, 5, 6, 5, 5, 7, 6, 5, 6, 6, 5, 5, 6, 7, 5, 6, 6, 6, 6, 6,
       5, 5, 6, 6, 6, 7, 5, 5, 6])
```

In [116]:
```
from sklearn.metrics import accuracy_score
```

In [117]:
```
accuracy_score(y_test,y_predict)
```

Out[117]:
```
0.5278396436525612
```

Hyper Parameter Tuning DecisionTree Classifier Model

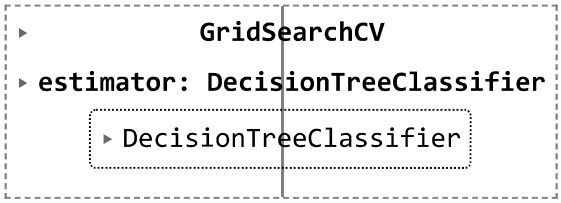
In [118]:
```
grid_param={
    'criterion':['gini','entropy'],
    'max_depth':range(2,15,1),
    'min_samples_leaf':range(2,12,1),
    'min_samples_split':range(2,12,1),
    'splitter' : ['best' , 'random']

}
```

In [119]:
```
from sklearn.model_selection import GridSearchCV
grid_search=GridSearchCV(estimator=model,param_grid=grid_param,cv=4,verbose=1)
```

In [120]:
```
grid_search.fit(X_train,y_train)
```

```
Fitting 4 folds for each of 5200 candidates, totalling 20800 fits
```

Out[120]:



In [121]:
```
grid_search.best_params_
```

Out[121]:
```
{'criterion': 'gini',
 'max_depth': 9,
 'min_samples_leaf': 9,
 'min_samples_split': 6,
 'splitter': 'random'}
```

In [124]:
```
n_best_params=DecisionTreeClassifier(criterion= 'gini',max_depth= 6,min_samples_le
```

In [125]:
```
model_with_best_params.fit(X_train,y_train)
```

Out[125]:
```
DecisionTreeClassifier(max_depth=6, min_samples_leaf=4, min_samples_split=6,
                       splitter='random')
```

In [128]:
```
from sklearn import tree
import matplotlib.pyplot as plt
fig=plt.figure(figsize=(25,15))
tree.plot_tree(model_with_best_params,filled=True,fontsize=10)
```

```
 Text(0.7380952380952381, 0.21428571428571427, 'gini = 0.375\nsamples = 4\nva
lue = [0, 0, 0, 1, 3]'),
 Text(0.7619047619047619, 0.21428571428571427, 'gini = 0.0\nsamples = 5\nvalu
e = [0, 0, 0, 0, 5]'),
 Text(0.7738095238095238, 0.35714285714285715, 'gini = 0.531\nsamples = 8\nva
lue = [0, 1, 0, 2, 5]'),
 Text(0.8958333333333334, 0.6428571428571429, 'X[6] <= 33.019\ngini = 0.58\ns
amples = 61\nvalue = [0, 0, 9, 34, 18]'),
 Text(0.8273809523809523, 0.5, 'X[6] <= 18.443\ngini = 0.555\nsamples = 33\nv
alue = [0, 0, 6, 20, 7]'),
 Text(0.7976190476190477, 0.35714285714285715, 'X[2] <= 0.166\ngini = 0.663\n
samples = 13\nvalue = [0, 0, 4, 5, 4]'),
 Text(0.7857142857142857, 0.21428571428571427, 'X[9] <= 0.503\ngini = 0.642\n
samples = 9\nvalue = [0, 0, 4, 2, 3]'),
 Text(0.7738095238095238, 0.07142857142857142, 'gini = 0.5\nsamples = 4\nvalu
e = [0, 0, 2, 2, 0]'),
 Text(0.7976190476190477, 0.07142857142857142, 'gini = 0.48\nsamples = 5\nval
ue = [0, 0, 2, 0, 3]'),
 Text(0.8095238095238095, 0.21428571428571427, 'gini = 0.375\nsamples = 4\nva
```

In [131]:
```
y_predict2=model_with_best_params.predict(X_test)
```

In [132]:
```
y_predict2
```

Out[132]:
```
array([7, 5, 5, 5, 5, 5, 5, 6, 5, 7, 5, 5, 6, 5, 5, 6, 5, 6, 7, 5, 5, 5,
       6, 5, 7, 5, 5, 6, 5, 6, 6, 5, 6, 5, 5, 5, 5, 5, 6, 5, 5, 6, 6, 5,
       6, 5, 5, 6, 5, 6, 5, 5, 5, 5, 5, 6, 5, 7, 5, 6, 5, 6, 7, 5, 5, 5,
       5, 5, 6, 5, 5, 5, 5, 6, 6, 5, 6, 5, 6, 5, 6, 5, 5, 5, 5, 5, 5, 7,
       5, 5, 5, 5, 5, 5, 7, 5, 5, 5, 5, 5, 6, 5, 5, 5, 5, 6, 6, 6, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 5, 5, 5, 6, 6, 6, 6, 6, 5, 5, 7, 5,
       5, 5, 7, 5, 5, 5, 5, 6, 7, 5, 6, 6, 5, 5, 5, 5, 5, 6, 5, 7, 6, 5,
       5, 7, 5, 5, 5, 5, 5, 6, 5, 5, 5, 5, 5, 5, 5, 5, 6, 5, 6, 6, 5, 5, 6,
       6, 7, 6, 5, 7, 5, 6, 5, 5, 5, 5, 6, 6, 5, 6, 5, 5, 5, 7, 7, 5, 5,
       6, 6, 6, 7, 5, 5, 6, 6, 6, 5, 7, 6, 6, 5, 5, 7, 5, 5, 5, 5, 6, 7,
       5, 6, 5, 5, 5, 6, 6, 5, 6, 6, 5, 6, 5, 6, 6, 6, 5, 5, 5, 6, 5, 5,
       5, 6, 6, 7, 5, 6, 5, 5, 5, 5, 6, 6, 6, 7, 6, 7, 6, 7, 7, 7, 6, 5,
       5, 5, 5, 7, 5, 7, 5, 6, 6, 6, 6, 6, 7, 6, 6, 5, 5, 6, 5, 6, 5, 6,
       5, 6, 5, 5, 5, 5, 5, 6, 6, 5, 5, 6, 7, 5, 6, 6, 5, 7, 6, 6, 5, 6,
       6, 5, 5, 7, 6, 5, 7, 5, 5, 5, 5, 5, 5, 7, 4, 6, 6, 7, 5, 5, 7, 5,
       6, 6, 6, 6, 5, 5, 5, 5, 5, 5, 7, 5, 6, 6, 5, 7, 5, 6, 6, 6, 5, 6,
       5, 6, 5, 5, 5, 6, 7, 6, 5, 7, 6, 6, 6, 5, 6, 6, 7, 5, 5, 7, 6, 5,
       5, 6, 7, 6, 5, 5, 5, 5, 5, 5, 5, 6, 5, 6, 6, 5, 7, 7, 6, 5, 6, 5,
       6, 5, 5, 5, 5, 5, 5, 6, 6, 5, 5, 5, 6, 6, 6, 5, 5, 7, 5, 5, 6, 5,
       5, 5, 6, 5, 6, 5, 6, 6, 6, 5, 6, 5, 5, 5, 6, 5, 6, 5, 7, 6, 6, 5,
       7, 5, 5, 5, 6, 7, 6, 6, 6])
```

In [133]:
```
accuracy_score(y_test,y_predict2)
```

Out[133]:
```
0.5256124721603563
```

Random Forest Classifier

In [135]:
```
from sklearn.ensemble import RandomForestClassifier
Rf_model=RandomForestClassifier()
```

In [136]:
```
Rf_model
```

Out[136]:
```
RandomForestClassifier
RandomForestClassifier()
```

In [137]:
```
Rf_model.fit(X_train,y_train)
```

Out[137]:
```
RandomForestClassifier
RandomForestClassifier()
```

In [138]:
```
y_pred_rf=Rf_model.predict(X_test)
```

In [139]:
```
y_pred_rf
```

Out[139]:
```
array([6, 6, 6, 5, 5, 6, 5, 6, 5, 7, 5, 5, 6, 5, 5, 6, 5, 5, 6, 6, 6, 6,
       7, 5, 7, 5, 6, 6, 5, 6, 7, 6, 6, 5, 6, 5, 5, 5, 6, 5, 5, 6, 6, 5,
       5, 5, 6, 6, 5, 6, 5, 5, 5, 5, 6, 5, 5, 7, 5, 5, 5, 6, 7, 5, 5, 6,
       5, 5, 6, 5, 5, 5, 5, 6, 6, 5, 6, 6, 5, 6, 7, 5, 5, 5, 5, 5, 5, 7,
       5, 5, 5, 5, 5, 5, 6, 5, 5, 5, 5, 6, 6, 5, 5, 5, 5, 6, 6, 6, 5, 5,
       5, 5, 5, 5, 5, 5, 6, 6, 5, 6, 5, 3, 5, 6, 6, 6, 5, 6, 5, 5, 6, 5,
       5, 5, 7, 5, 6, 5, 6, 7, 6, 5, 7, 6, 5, 5, 5, 6, 5, 6, 5, 6, 6, 5,
       6, 6, 5, 5, 6, 5, 5, 6, 5, 5, 5, 5, 6, 5, 5, 5, 5, 6, 6, 5, 5, 6,
       7, 6, 6, 5, 5, 6, 6, 5, 5, 5, 5, 6, 6, 5, 6, 5, 5, 5, 6, 7, 3, 5,
       6, 6, 6, 6, 5, 5, 6, 6, 6, 5, 6, 6, 6, 5, 7, 7, 5, 5, 5, 6, 6, 7,
       5, 6, 5, 5, 5, 6, 7, 6, 6, 6, 5, 6, 6, 6, 6, 6, 5, 5, 5, 4, 5, 5,
       5, 6, 7, 7, 5, 6, 6, 5, 6, 5, 6, 6, 6, 7, 6, 7, 6, 7, 5, 6, 5, 5,
       6, 6, 6, 6, 5, 7, 6, 7, 7, 6, 6, 6, 7, 6, 6, 5, 6, 7, 5, 6, 5, 6,
       5, 6, 5, 6, 5, 5, 5, 7, 6, 6, 5, 6, 7, 5, 6, 6, 5, 7, 6, 6, 5, 5,
       7, 5, 5, 6, 6, 5, 5, 6, 6, 5, 5, 6, 5, 7, 6, 6, 6, 6, 6, 5, 6, 5,
       6, 7, 7, 6, 6, 5, 6, 5, 5, 5, 7, 5, 6, 6, 5, 7, 6, 6, 7, 7, 5, 6,
       5, 6, 6, 5, 5, 6, 7, 6, 5, 6, 6, 6, 6, 6, 6, 6, 7, 5, 6, 7, 6, 5,
       5, 7, 7, 6, 5, 5, 6, 5, 6, 5, 5, 6, 6, 6, 5, 5, 7, 7, 6, 5, 6, 5,
       6, 5, 6, 6, 5, 5, 6, 5, 6, 5, 5, 5, 6, 6, 6, 6, 5, 7, 6, 6, 6, 5,
       5, 5, 5, 5, 6, 5, 5, 6, 6, 5, 5, 6, 5, 5, 6, 6, 6, 6, 6, 7, 6, 6,
       5, 6, 6, 5, 6, 7, 5, 6, 6])
```

In [142]:
```
accuracy_score(y_test,y_pred_rf)
```

Out[142]:
```
0.5946547884187082
```

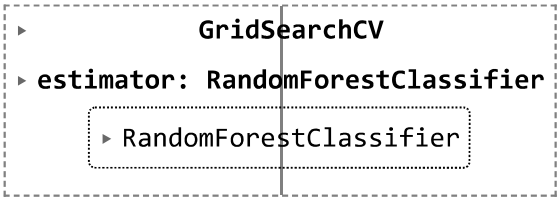
In [143]:
```
# we are tuning three hyperparameters right now, we are passing the different val
grid_param = {
    "n_estimators" : [90,100,115,130],
    'criterion': ['gini', 'entropy'],
    'max_depth' : range(2,20,1),
    'min_samples_leaf' : range(1,10,1),
    'min_samples_split': range(2,10,1),
    'max_features' : ['auto','log2']
}
```

In [147]:
```
from sklearn.model_selection import GridSearchCV
grid_search=GridSearchCV(estimator=Rf_model,param_grid=grid_param,cv=2,verbose=1,
```

In [149]:
```
grid_search.fit(X_train,y_train)
```

```
Fitting 2 folds for each of 20736 candidates, totalling 41472 fits
```

Out[149]:



In [150]:
```
grid_search.best_params_
```

Out[150]:
```
{'criterion': 'gini',
 'max_depth': 4,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 6,
 'n_estimators': 100}
```

In [157]:
```
ForestClassifier(criterion='gini',max_depth=4,max_features='auto',min_samples_lea
```

In [158]:
```
Rf_model_with_best_params.fit(X_train,y_train)
```

Out[158]:
```
RandomForestClassifier

RandomForestClassifier(max_depth=4, max_features='auto', min_samples_split=6)
```

In [159]:
```
y_predict_rf_best_par=Rf_model_with_best_params.predict(X_test)
```

In [160]:
```
y_predict_rf_best_par
```

Out[160]:
```
array([6, 5, 6, 5, 5, 6, 5, 6, 6, 7, 5, 5, 6, 5, 5, 6, 6, 6, 6, 6, 5, 6,
       6, 5, 6, 5, 5, 6, 5, 6, 7, 6, 6, 5, 5, 5, 6, 5, 6, 5, 5, 6, 6, 5,
       5, 5, 5, 6, 5, 6, 5, 5, 5, 5, 6, 6, 5, 7, 5, 6, 5, 6, 7, 6, 6, 6,
       5, 6, 6, 5, 5, 5, 6, 6, 6, 5, 6, 6, 5, 5, 6, 5, 5, 5, 5, 5, 5, 6,
       5, 5, 5, 5, 5, 5, 6, 5, 5, 5, 5, 5, 6, 5, 5, 5, 5, 6, 6, 6, 5, 6,
       5, 5, 5, 5, 5, 5, 6, 6, 5, 6, 5, 6, 5, 6, 6, 6, 5, 5, 5, 5, 6, 5,
       5, 5, 6, 5, 5, 5, 5, 6, 6, 5, 7, 6, 5, 5, 5, 6, 5, 6, 5, 6, 6, 5,
       5, 6, 5, 5, 5, 5, 5, 6, 5, 5, 5, 5, 6, 5, 5, 5, 5, 6, 6, 5, 5, 6,
       6, 6, 6, 5, 6, 5, 6, 5, 5, 5, 5, 6, 6, 5, 6, 5, 5, 5, 6, 6, 5, 5,
       6, 6, 6, 6, 5, 5, 6, 6, 6, 5, 6, 6, 6, 5, 6, 6, 5, 5, 5, 6, 6, 6,
       5, 6, 5, 5, 5, 6, 6, 6, 6, 6, 5, 6, 5, 6, 6, 5, 5, 5, 5, 6, 5, 5,
       5, 6, 6, 6, 5, 6, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5,
       5, 6, 6, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 7, 5, 6, 5, 6,
       5, 6, 5, 6, 5, 5, 5, 6, 6, 6, 5, 6, 6, 5, 6, 6, 5, 6, 6, 6, 5, 6,
       6, 5, 5, 6, 6, 5, 6, 5, 6, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 5, 6, 5,
       6, 6, 6, 6, 6, 5, 6, 5, 5, 5, 6, 5, 6, 6, 6, 6, 5, 6, 6, 6, 5, 6,
       5, 6, 6, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 5, 6, 6, 5,
       5, 6, 6, 6, 5, 5, 6, 5, 5, 5, 6, 6, 5, 6, 6, 5, 7, 6, 6, 5, 6, 5,
       6, 5, 6, 6, 5, 5, 6, 5, 6, 5, 5, 5, 6, 6, 6, 5, 5, 6, 5, 5, 6, 5,
       5, 5, 6, 5, 6, 5, 6, 6, 6, 5, 6, 6, 5, 5, 6, 6, 6, 6, 6, 7, 6, 5,
       6, 6, 6, 5, 6, 6, 6, 6, 6])
```

In [161]:
```
accuracy_score(y_test,y_predict_rf_best_par)
```

Out[161]:
```
0.5746102449888641
```

In [164]:
```
from sklearn.svm import SVC
from  sklearn.ensemble import BaggingClassifier
```

In [165]:
```
model_bagging_svc = BaggingClassifier(base_estimator=SVC(),n_estimators=50, randc
```

In [166]:
```
y_predict_bagging=model_bagging_svc.predict(X_test)
```

In [167]:
```
accuracy_score(y_test,y_predict_bagging)
```

Out[167]:
```
0.47438752783964366
```

In [ ]: