Project done by : Janhavi Pramod Yarguddi

Contact no. : 9511794188

Importing Libraries

In [1]:

```
import pandas as pd
import numpy as np
from scipy.stats import mode
from statsmodels.stats.outliers_influence import variance_inflation_factor

# For visualisation purpose
import seaborn as sns
import matplotlib.pyplot as plt

#For model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

#For evaluation metrics
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.metrics import confusion_matrix,accuracy_score,precision_score,recall_s

# For model_selcction and testing
from sklearn.model_selection import train_test_split,GridSearchCV,RandomizedSearchCV

import warnings
warnings.filterwarnings("ignore")
```

Problem Statement:

```
To predict whether the water is potable or not
```

Data Gathering and Data Validation

Exploratory Data Analysis :

In [2]:

```
df_water = pd.read_csv("water_potability.csv")
df_water.head()
```

Out[2]:

	Unnamed: 0	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	O
0	0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	
1	1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	
2	2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	
3	3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	
4	4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	

In [3]:

```
df_water.drop("Unnamed: 0",axis=1,inplace=True)
```

In [4]:

```
df_water.head()
```

Out[4]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carb
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.3797
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.1800
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.8686
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.4365
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.5582

Checking for no. of rows and columns

In [5]:

```
df_water.shape
```

Out[5]:

```
(3276, 10)
```

The given data set has 3276 rows and 10 features(columns). The feature names are : 'ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity','Organic_carbon', 'Trihalomethanes', 'Turbidity', 'Potability'.

In [6]:

```
df_water.columns
```

Out[6]:

```
Index(['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivit
y',
       'Organic_carbon', 'Trihalomethanes', 'Turbidity', 'Potability'],
      dtype='object')
```

Finding Missing Values

In [7]:

```
df_water.isna().sum()
```

Out[7]:

```
ph                 491
Hardness             0
Solids               0
Chloramines          0
Sulfate            781
Conductivity         0
Organic_carbon       0
Trihalomethanes    162
Turbidity            0
Potability           0
dtype: int64
```

In [8]:

```
df_water.isna().mean()*100
```

Out[8]:

```
ph                 14.987790
Hardness            0.000000
Solids              0.000000
Chloramines         0.000000
Sulfate            23.840049
Conductivity        0.000000
Organic_carbon      0.000000
Trihalomethanes     4.945055
Turbidity           0.000000
Potability          0.000000
dtype: float64
```

From this we find that around 15% data is missing in the "ph" feature. 24% data is missing in "Sulphate" feature and around 5% data is missing from the feature named "Trihalomethanes"

In [9]:

```
df_water.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               2785 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3114 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

Here we are building a model . And the model requires all numerical data to be passed. And from this result we understand that here all the features data is numerical and hence no encoding is required. Also we find that there are some columns that have missing values.

Potability

In [10]:

```
df_water["Potability"].value_counts()
```

Out[10]:

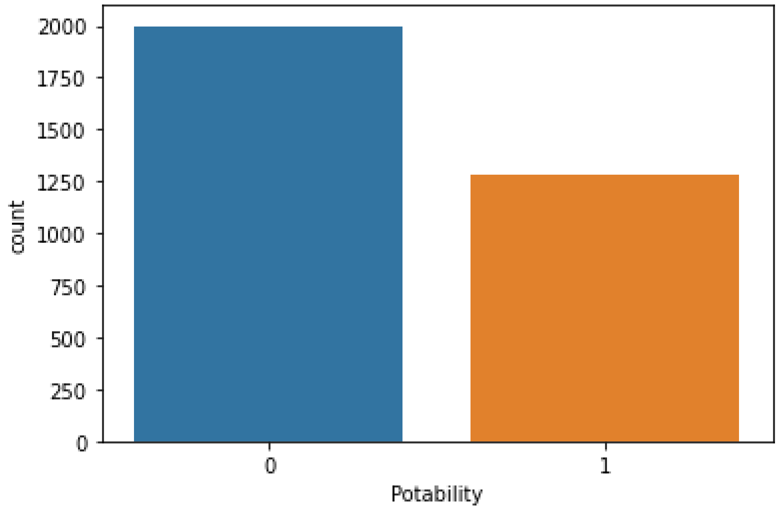
```
0    1998
1    1278
Name: Potability, dtype: int64
```

In [11]:

```
sns.countplot(df_water["Potability"])
```

Out[11]:

<AxesSubplot:xlabel='Potability', ylabel='count'>



In [12]:

```
df_water["Potability"].nunique()
```

Out[12]:

2

We have 2 categories here. That is this is binary classification problem. And also we get to know that the data that we have is not evenly distributed

Statistics

In [13]:

```
df_water.describe()
```

Out[13]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Org
count	2785.000000	3276.000000	3276.000000	3276.000000	2495.000000	3276.000000	
mean	7.080795	196.369496	22014.092526	7.122277	333.775777	426.205111	
std	1.594320	32.879761	8768.570828	1.583085	41.416840	80.824064	
min	0.000000	47.432000	320.942611	0.352000	129.000000	181.483754	
25%	6.093092	176.850538	15666.690297	6.127421	307.699498	365.734414	
50%	7.036752	196.967627	20927.833607	7.130299	333.073546	421.884968	
75%	8.062066	216.667456	27332.762127	8.114887	359.950170	481.792304	
max	14.000000	323.124000	61227.196008	13.127000	481.030642	753.342620	

From the statistics we find that the data in the given data set has a wide range and hence :

1) there is need for scaling either standardization or normalization

2) Also looking at the min value max value and the percentile values present in particular feature we can infer that outliers are present.

Detecting outlilers :

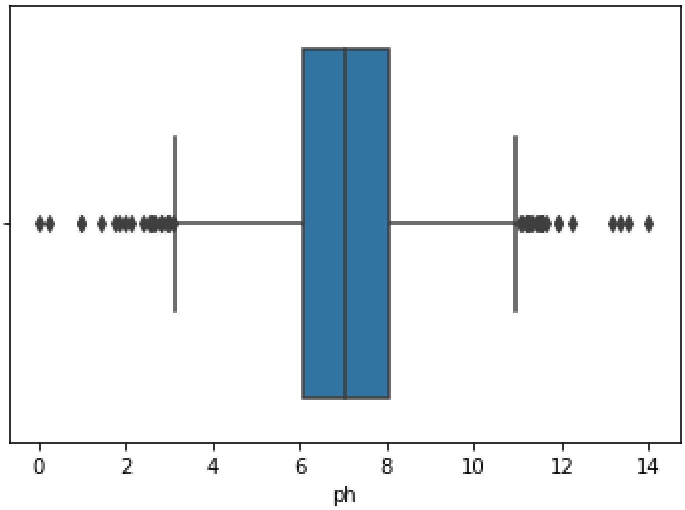
For feature "ph"

In [14]:

```
sns.boxplot(df_water["ph"])
```

Out[14]:

```
<AxesSubplot:xlabel='ph'>
```

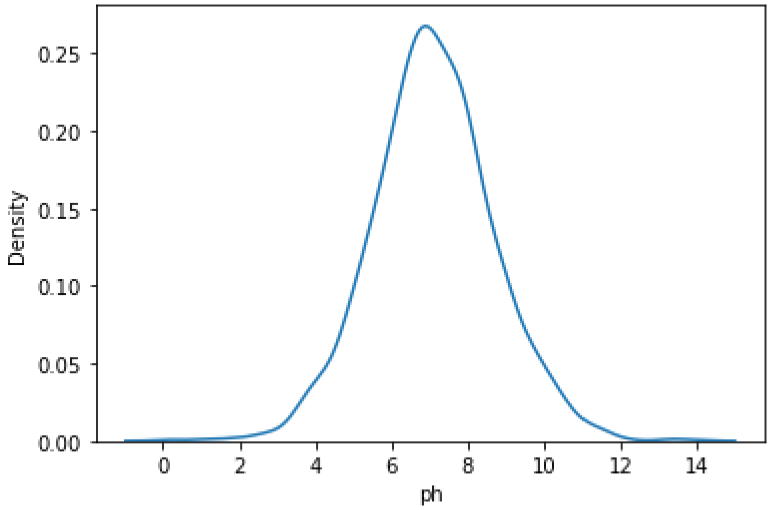


In [15]:

```
1 sns.kdeplot(df_water["ph"])
```

Out[15]:

<AxesSubplot:xlabel='ph', ylabel='Density'>



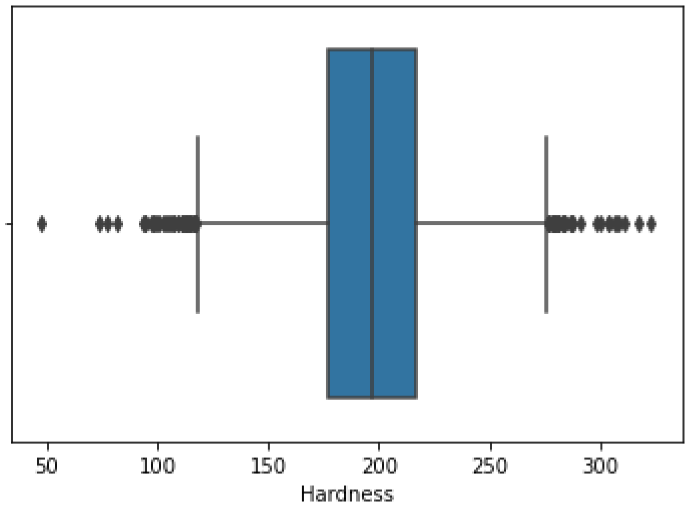
for feature "Hardness"

In [16]:

```
1 sns.boxplot(df_water["Hardness"])
```

Out[16]:

<AxesSubplot:xlabel='Hardness'>

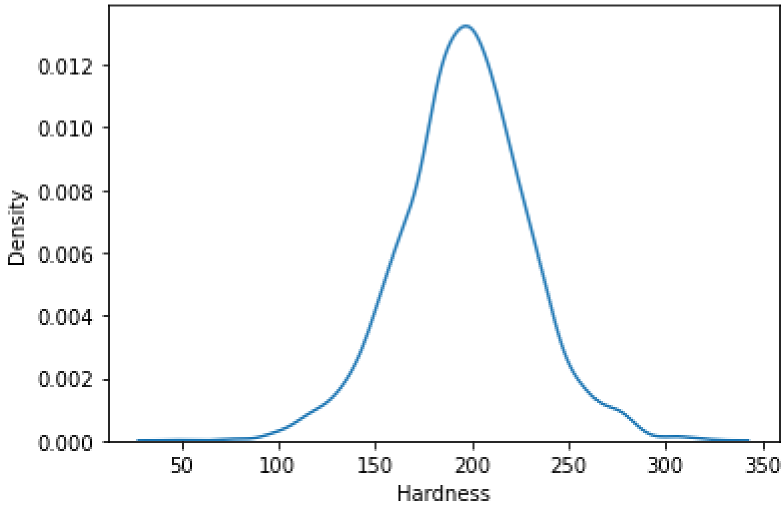


In [17]:

```
sns.kdeplot(df_water["Hardness"])
```

Out[17]:

```
<AxesSubplot:xlabel='Hardness', ylabel='Density'>
```



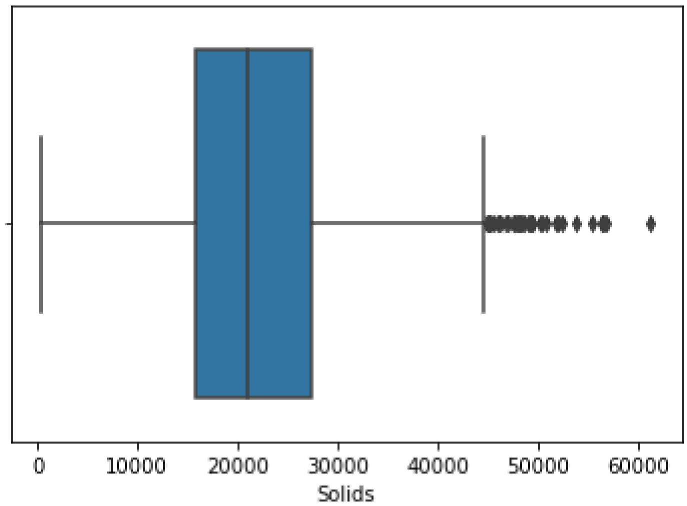
for feature "Solids"

In [18]:

```
sns.boxplot(df_water["Solids"])
```

Out[18]:

```
<AxesSubplot:xlabel='Solids'>
```

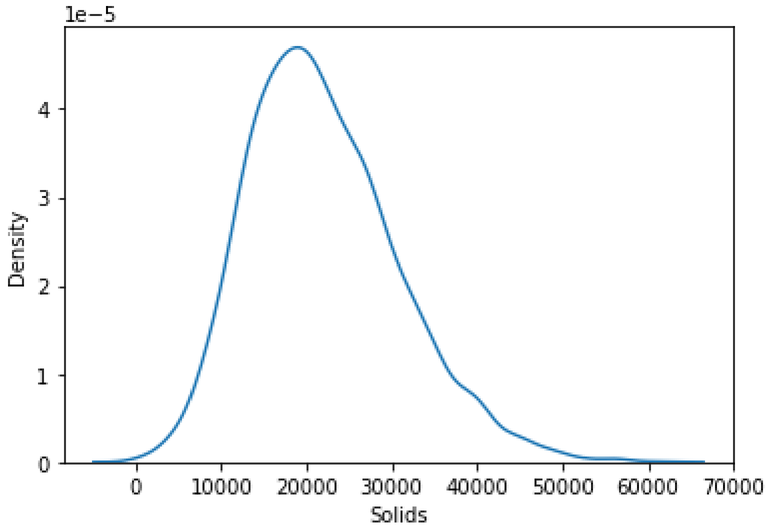


In [19]:

```
1 sns.kdeplot(df_water["Solids"])
```

Out[19]:

```
<AxesSubplot:xlabel='Solids', ylabel='Density'>
```



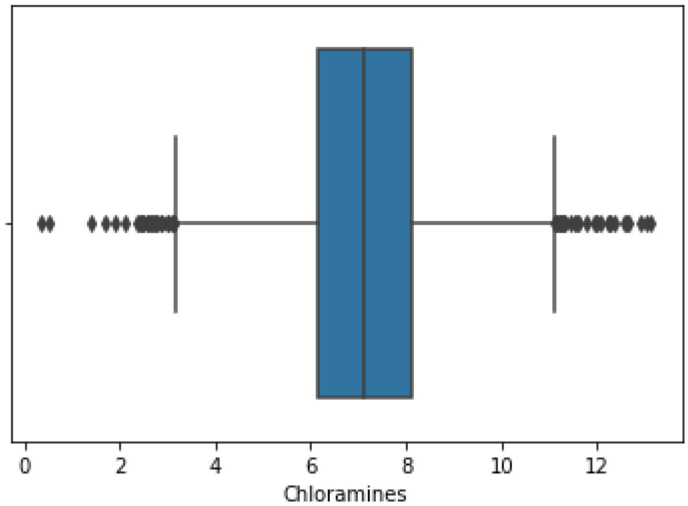
for feature "Chloramines"

In [20]:

```
1 sns.boxplot(df_water["Chloramines"])
```

Out[20]:

```
<AxesSubplot:xlabel='Chloramines'>
```

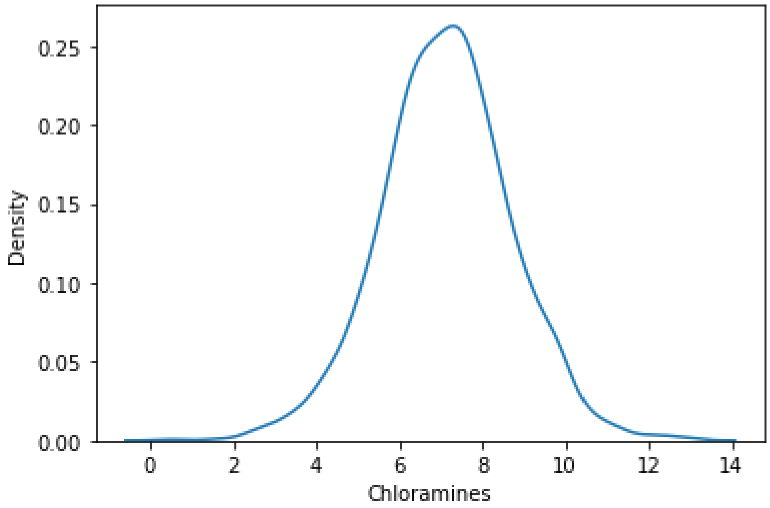


In [21]:

```
sns.kdeplot(df_water["Chloramines"])
```

Out[21]:

<AxesSubplot:xlabel='Chloramines', ylabel='Density'>



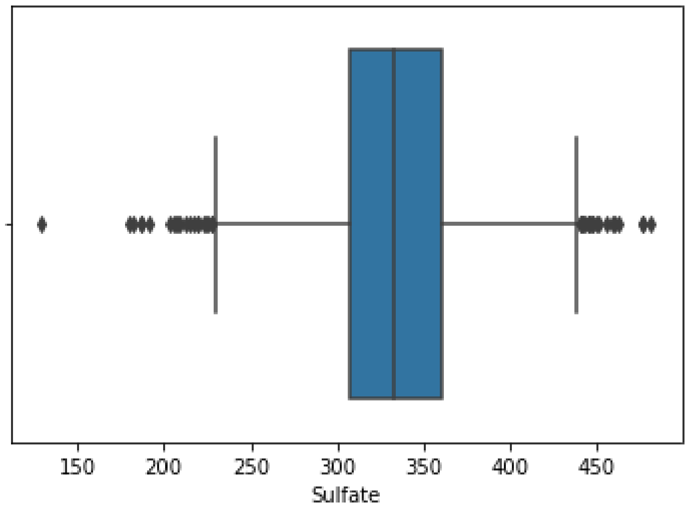
for feature "Sulfate"

In [22]:

```
sns.boxplot(df_water["Sulfate"])
```

Out[22]:

<AxesSubplot:xlabel='Sulfate'>

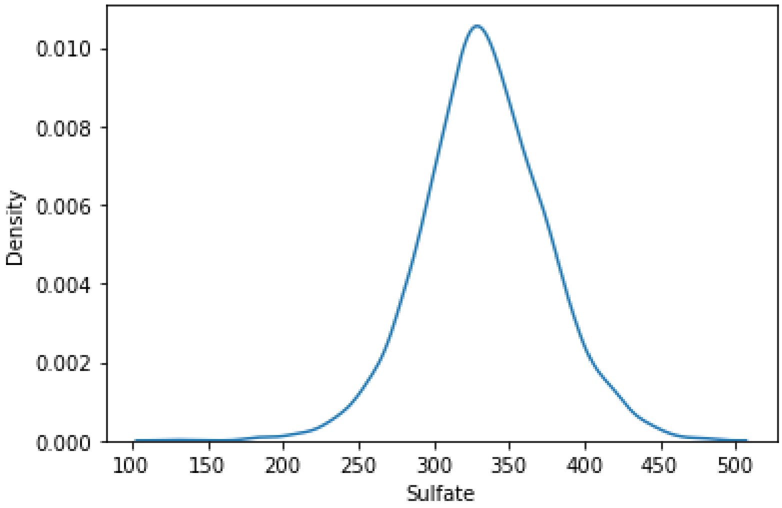


In [23]:

```
sns.kdeplot(df_water["Sulfate"])
```

Out[23]:

```
<AxesSubplot:xlabel='Sulfate', ylabel='Density'>
```



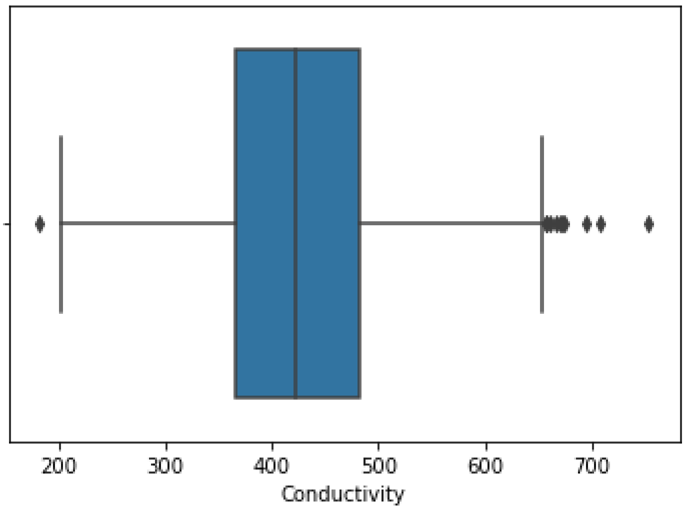
for feature "Conductivity"

In [24]:

```
sns.boxplot(df_water["Conductivity"])
```

Out[24]:

```
<AxesSubplot:xlabel='Conductivity'>
```

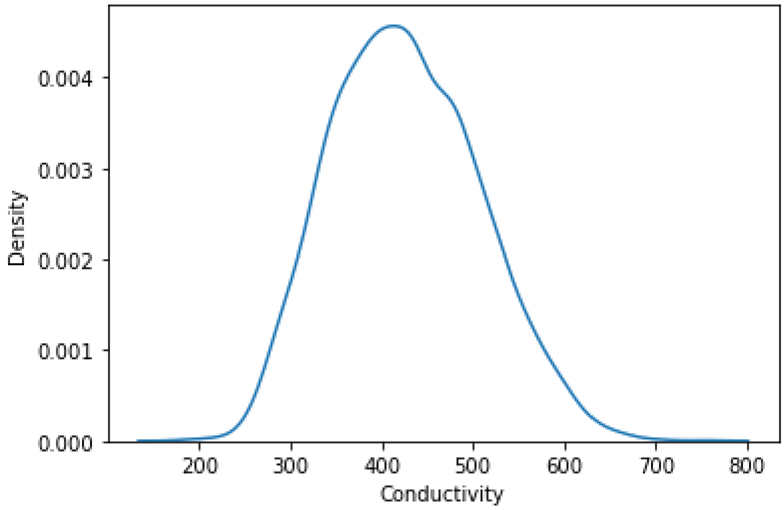


In [25]:

```
sns.kdeplot(df_water["Conductivity"])
```

Out[25]:

```
<AxesSubplot:xlabel='Conductivity', ylabel='Density'>
```



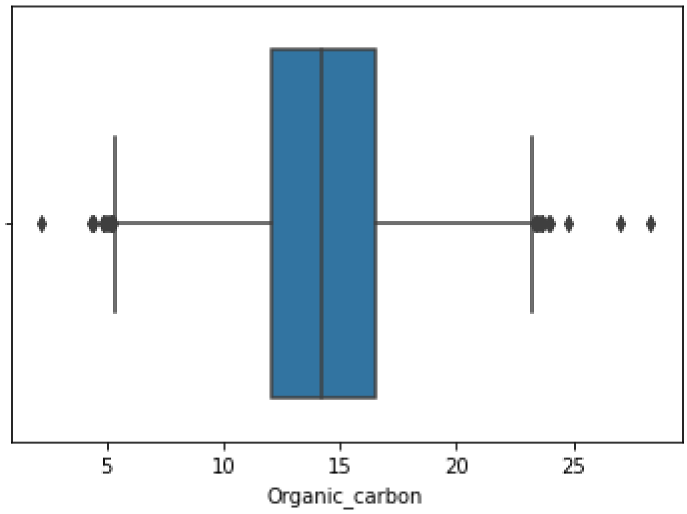
for feature "Organic_carbon"

In [26]:

```
sns.boxplot(df_water["Organic_carbon"])
```

Out[26]:

```
<AxesSubplot:xlabel='Organic_carbon'>
```

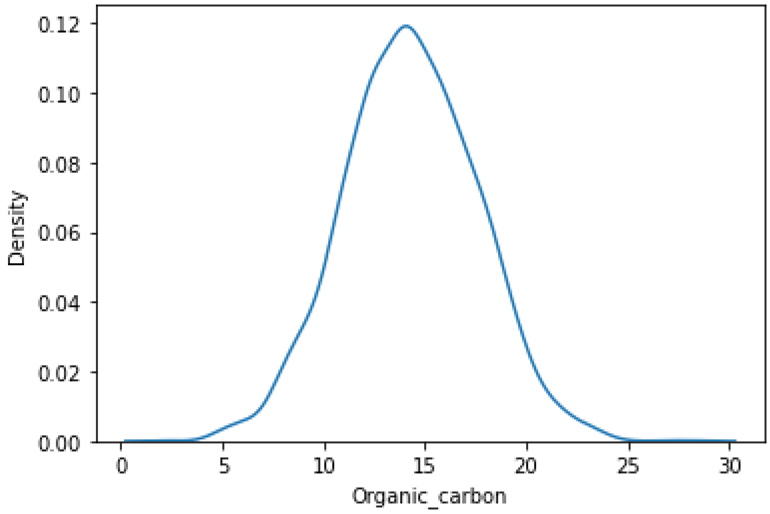


In [27]:

```
1 sns.kdeplot(df_water["Organic_carbon"])
```

Out[27]:

```
&lt;AxesSubplot:xlabel='Organic_carbon', ylabel='Density'&gt;
```



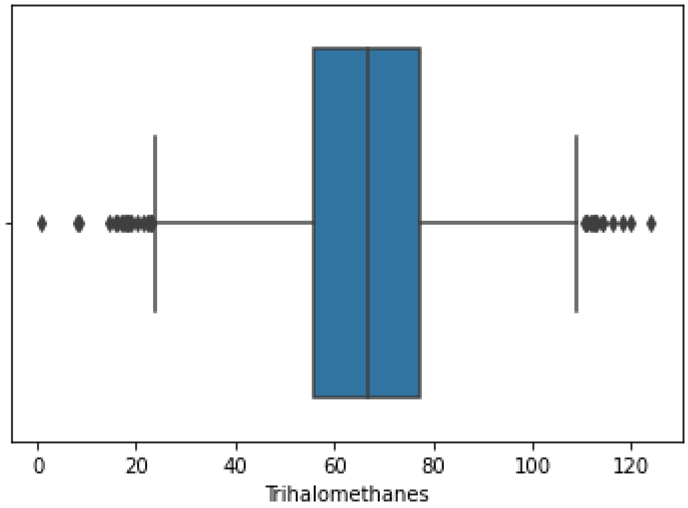
for feature "Trihalomethanes"

In [28]:

```
1 sns.boxplot(df_water["Trihalomethanes"])
```

Out[28]:

```
&lt;AxesSubplot:xlabel='Trihalomethanes'&gt;
```

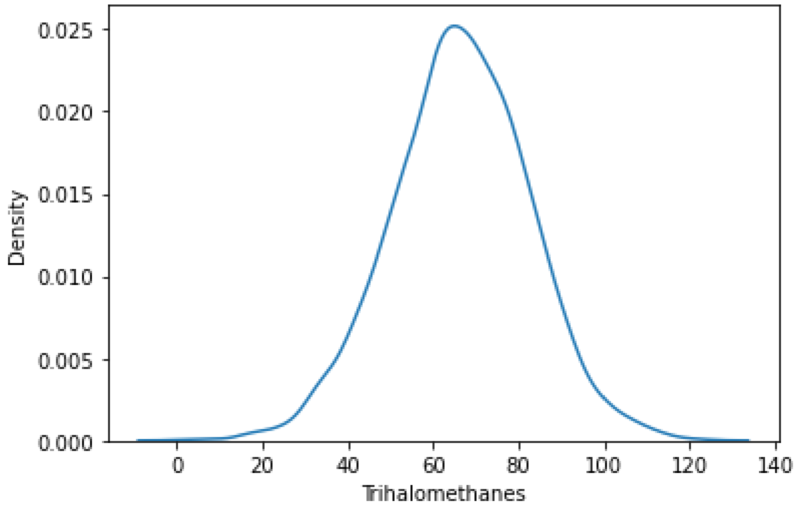


In [29]:

```
1 sns.kdeplot(df_water["Trihalomethanes"])
```

Out[29]:

<AxesSubplot:xlabel='Trihalomethanes', ylabel='Density'>



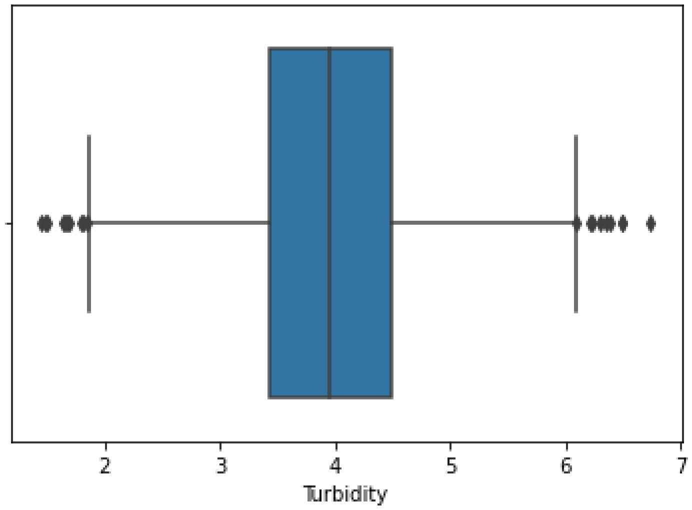
for feature "Turbidity"

In [30]:

```
1 sns.boxplot(df_water["Turbidity"])
```

Out[30]:

<AxesSubplot:xlabel='Turbidity'>

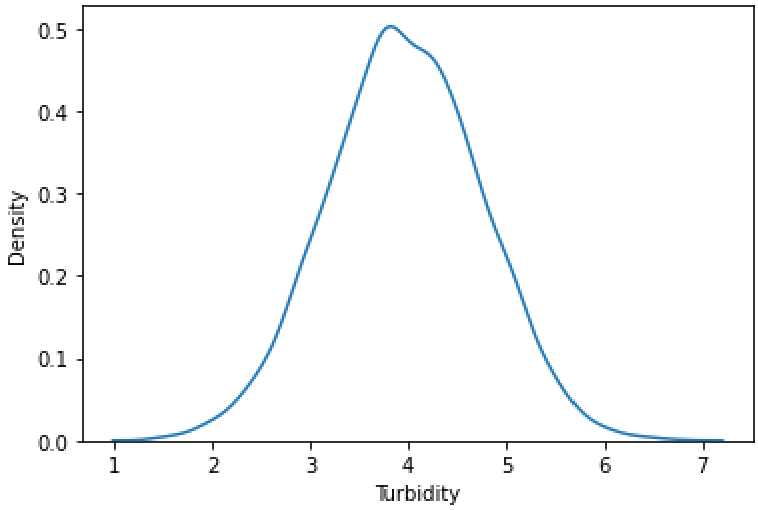


In [31]:

```
sns.kdeplot(df_water["Turbidity"])
```

Out[31]:

```
<AxesSubplot:xlabel='Turbidity', ylabel='Density'>
```



After going through all the features we find :

1) Presence of outliers in almost all the features

2) when plotted using kdeplot we come to know that the data is almost normally distributed, in some cases it is positively skewed(for eg - "solids") while in some cases it is negatively skewed(for eg : "Trihalomethanes","Sulfate").But since the range of the data is widely spread , data should necessarily undergo scaling (standardization) which to some extent will also reduce the impact of outliers on the data.

Data distribution and Correlation

In [32]:

```
df_water.corr()
```

Out[32]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Orga
ph	1.000000	0.082096	-0.089288	-0.034350	0.018203	0.018614	
Hardness	0.082096	1.000000	-0.046899	-0.030054	-0.106923	-0.023915	
Solids	-0.089288	-0.046899	1.000000	-0.070148	-0.171804	0.013831	
Chloramines	-0.034350	-0.030054	-0.070148	1.000000	0.027244	-0.020486	
Sulfate	0.018203	-0.106923	-0.171804	0.027244	1.000000	-0.016121	
Conductivity	0.018614	-0.023915	0.013831	-0.020486	-0.016121	1.000000	
Organic_carbon	0.043503	0.003610	0.010242	-0.012653	0.030831	0.020966	
Trihalomethanes	0.003354	-0.013013	-0.009143	0.017084	-0.030274	0.001285	
Turbidity	-0.039057	-0.014449	0.019546	0.002363	-0.011187	0.005798	
Potability	-0.003556	-0.013837	0.033743	0.023779	-0.023577	-0.008128	

The range of correlation coeficient is: -1 to 1
And if the correlation coefficient is from -0.7 to -1 then we say it is negatively correlated and if r = 0.7 to 1 then we say it is positively correlated.
Here from the correlation coefficient values we get to know that no two features are strongly correlated with each other.We can infer the same using the pairplot.

In [33]:

```
df_water.corr().tail(1)
```

Out[33]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carb
Potability	-0.003556	-0.013837	0.033743	0.023779	-0.023577	-0.008128	-0.030(

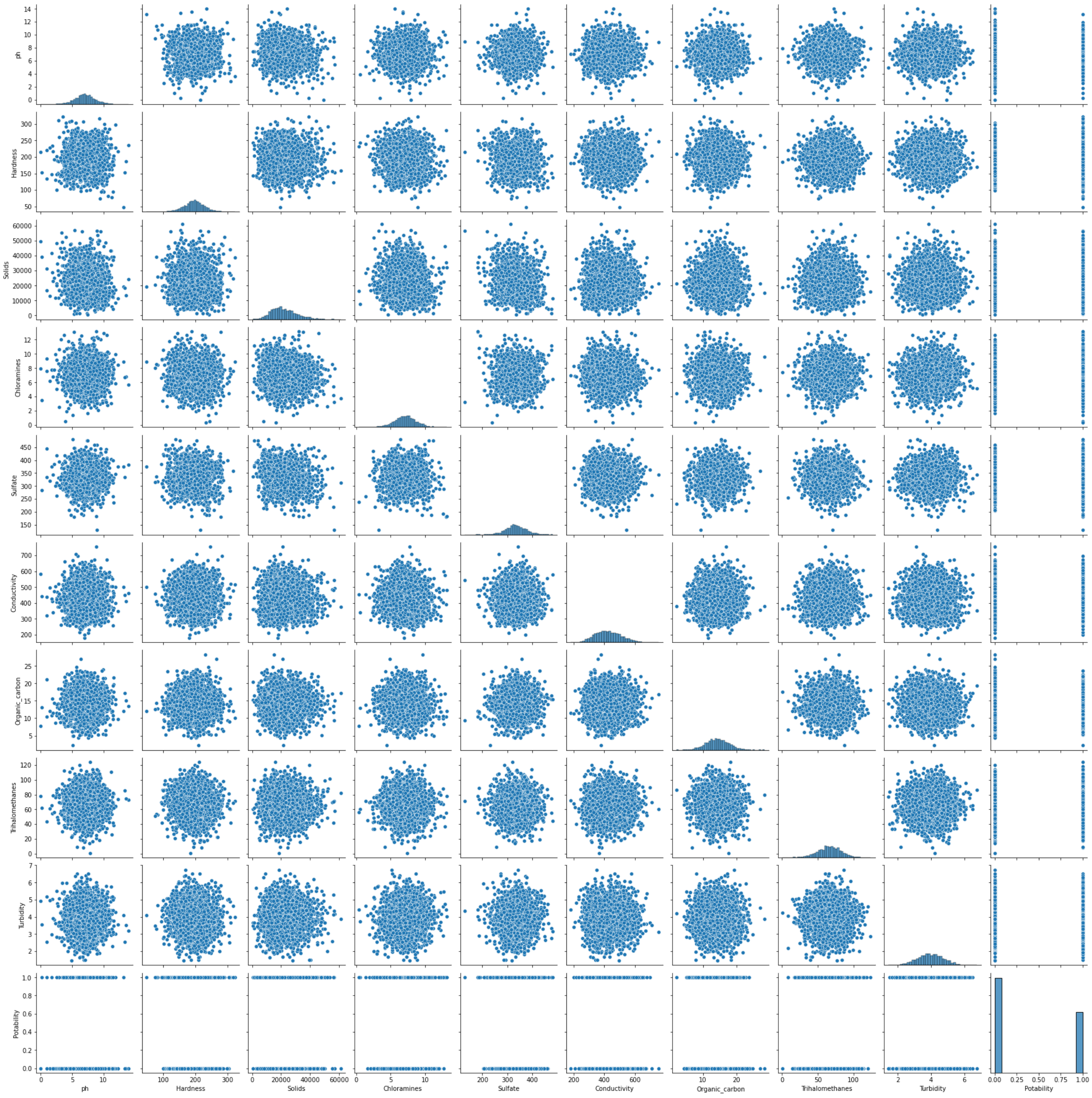
From the above we get the correlation between the dependent target variable and independent features.

In [34]:

```
sns.pairplot(df_water)
```

Out[34]:

```
<seaborn.axisgrid.PairGrid at 0x1c0b1810790>
```



In [ ]:

```
Water = pd.read_csv("water_potability.csv")
Water
```

In [ ]:

```
df_water1 = df_water.to_excel("water_potability1.xlsx")
df_water1
```

Feature Engineering

Handling missing values

In [35]:

```
df_water.isna().mean()*100
```

Out[35]:

```
ph                 14.987790
Hardness            0.000000
Solids              0.000000
Chloramines         0.000000
Sulfate            23.840049
Conductivity        0.000000
Organic_carbon      0.000000
Trihalomethanes     4.945055
Turbidity           0.000000
Potability          0.000000
dtype: float64
```

In [36]:

```
df_water.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               2785 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3114 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

From the above analysis we get to know that there are some values missing in the features "ph","Sulfates" and "Trihalomethanes"

In [37]:

```
df_water["ph"].mean()
```

Out[37]:

```
7.080794504276819
```

In [38]:

```
df_water["ph"].median()
```

Out[38]:

```
7.036752103833548
```

Since the distribution of "ph" seems to follow normal distribution , the missing values can be filled with **mean**

In [39]:

```
data = df_water.copy()
```

In [40]:

```
from scipy.stats import mode
```

In [41]:

```
df_water["Sulfate"].mean()
```

Out[41]:

```
333.7757766108134
```

In [42]:

```
df_water["Sulfate"].median()
```

Out[42]:

```
333.073545745888
```

In [43]:

```
df_water["Trihalomethanes"].mean()
```

Out[43]:

```
66.39629294676803
```

In [44]:

```
df_water["Trihalomethanes"].median()
```

Out[44]:

```
66.62248509808484
```

Filling missing values

In [45]:

```
data["ph"]              = data["ph"].fillna(data["ph"].median()).astype(float)
data['Sulfate'] = data['Sulfate'].fillna(data.groupby(['Potability'])['Sulfate'].tra
data['Trihalomethanes'] = data['Trihalomethanes'].fillna(data.groupby(['Potability']
```

In [46]:

```
data.isna().sum()
```

Out[46]:

```
ph                 0
Hardness           0
Solids             0
Chloramines        0
Sulfate            0
Conductivity       0
Organic_carbon     0
Trihalomethanes    0
Turbidity          0
Potability         0
dtype: int64
```

In [47]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               3276 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          3276 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3276 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

Handling outliers

In [48]:

```
data.describe()
```

Out[48]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Org
count	3276.000000	3276.000000	3276.000000	3276.000000	3276.000000	3276.000000	
mean	7.074194	196.369496	22014.092526	7.122277	333.785123	426.205111	
std	1.470040	32.879761	8768.570828	1.583085	36.145701	80.824064	
min	0.000000	47.432000	320.942611	0.352000	129.000000	181.483754	
25%	6.277673	176.850538	15666.690297	6.127421	317.094638	365.734414	
50%	7.036752	196.967627	20927.833607	7.130299	334.564290	421.884968	
75%	7.870050	216.667456	27332.762127	8.114887	350.385756	481.792304	
max	14.000000	323.124000	61227.196008	13.127000	481.030642	753.342620	

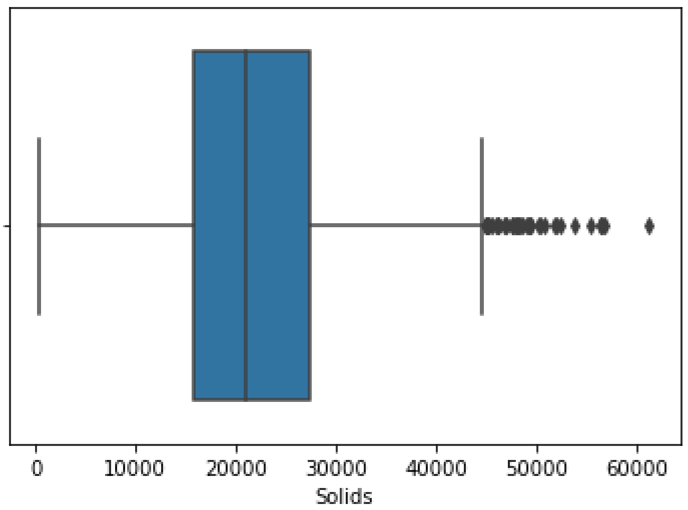
From the above analysis we get to know that extreme outliers are present in the feature named "solids" and "Conductivity"

In [49]:

```
sns.boxplot(data["Solids"])
```

Out[49]:

```
<AxesSubplot:xlabel='Solids'>
```



For detection of extreme outliers if any and then imputing those values with mean or median.

Solids

In [50]:

```
q1 = data["Solids"].quantile(0.25)
q2 = data["Solids"].quantile(0.50)
q3 = data["Solids"].quantile(0.75)
iqr = q3 - q1
upper_tail = q3 + 1.5*iqr
lower_tail = q1 - 1.5*iqr
```

In [51]:

```
median_solids =data.loc[(data["Solids"]<=upper_tail)&(data["Solids"]>=lower_tail),"S
data.loc[(data["Solids"]>upper_tail)|(data["Solids"]<lower_tail),"Solids"] = median_
```

Conductivity

In [52]:

```
q11 = data["Conductivity"].quantile(0.25)
q12 = data["Conductivity"].quantile(0.50)
q13 = data["Conductivity"].quantile(0.75)
iqr = q13 - q11
upper_tail_1 = q13 + 1.5*iqr
lower_tail_1 = q11 - 1.5*iqr
```

In [53]:

```
median_conductivity =data.loc[(data["Conductivity"]<=upper_tail_1)&(data["Conductivi
data.loc[(data["Conductivity"]>upper_tail_1)|(data["Conductivity"]<lower_tail_1),"Co
```

In [54]:

```
data.describe()
```

Out[54]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Org
count	3276.000000	3276.000000	3276.000000	3276.000000	3276.000000	3276.000000	
mean	7.074194	196.369496	21611.031084	7.122277	333.785123	425.484573	
std	1.470040	32.879761	8138.169195	1.583085	36.145701	79.449825	
min	0.000000	47.432000	320.942611	0.352000	129.000000	201.619737	
25%	6.277673	176.850538	15666.690297	6.127421	317.094638	365.811312	
50%	7.036752	196.967627	20709.279762	7.130299	334.564290	421.464253	
75%	7.870050	216.667456	26957.576932	8.114887	350.385756	480.855683	
max	14.000000	323.124000	44652.363872	13.127000	481.030642	652.537592	

- Since all the features here are of datatype float i.e. all data is numerical data , there is no need of Encoding (One hot encoding) or using the function pd.getdummies()

Feature Selection :

In [55]:

```
data.corr()
```

Out[55]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Orga
ph	1.000000	0.075760	-0.075988	-0.031741	0.014506	0.017844	
Hardness	0.075760	1.000000	-0.046151	-0.030054	-0.092718	-0.029564	
Solids	-0.075988	-0.046151	1.000000	-0.060990	-0.135737	0.006647	
Chloramines	-0.031741	-0.030054	-0.060990	1.000000	0.023490	-0.021607	
Sulfate	0.014506	-0.092718	-0.135737	0.023490	1.000000	-0.014027	
Conductivity	0.017844	-0.029564	0.006647	-0.021607	-0.014027	1.000000	
Organic_carbon	0.040240	0.003610	0.013017	-0.012653	0.027403	0.018193	
Trihalomethanes	0.003141	-0.012718	-0.017803	0.016615	-0.025797	-0.000587	
Turbidity	-0.036107	-0.014449	0.026211	0.002363	-0.009523	0.007564	
Potability	-0.003014	-0.013837	0.024972	0.023779	-0.026957	-0.008880	

In [56]:

```
data.corr().tail(1)
```

Out[56]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carb
Potability	-0.003014	-0.013837	0.024972	0.023779	-0.026957	-0.00888	-0.030

vif (Variance inflation factor)

In [57]:

```
vif = pd.DataFrame()
df = data.drop("Potability",axis=1)
vif["Feature"] = df.columns
vif["VIF_Values"] = [variance_inflation_factor(df.to_numpy(),i) for i in range(df.sh
vif
```

Out[57]:

	Feature	VIF_Values
0	ph	22.827399
1	Hardness	30.820793
2	Solids	7.654077
3	Chloramines	19.592880
4	Sulfate	56.421549
5	Conductivity	26.592780
6	Organic_carbon	18.699966
7	Trihalomethanes	17.344729
8	Turbidity	24.192158

If vif==1 then that will be ideal condition , which suggests that there is no multicollinearity between the features. But such condition is rarely found. And so usually the range threshold is 5 . If above 5 we drop the feature.

Model training :

In [58]:

```
x = data.drop("Potability",axis=1)
y = data["Potability"]
```

In [59]:

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=42,s
```

In [60]:

```
lg_regression = LogisticRegression()
lg_regression.fit(x_train,y_train)
```

Out[60]:

```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Model Evaluation :

In [61]:

```
# Training
y_pred_train = lg_regression.predict(x_train)

cnf_matrix = confusion_matrix(y_train,y_pred_train)
print("confusion matrix\n",cnf_matrix)

print("*"*50)
accuracy = accuracy_score(y_train,y_pred_train)
print("accuarcy\n",accuracy)
print("*"*50)
clf_report = classification_report(y_train,y_pred_train)
print("classificatio report\n",clf_report)
```

```
confusion matrix
 [[1598    0]
 [1022    0]]
**************************************************
accuarcy
 0.6099236641221374
**************************************************
classificatio report
               precision    recall  f1-score   support

           0       0.61      1.00      0.76      1598
           1       0.00      0.00      0.00      1022

    accuracy                           0.61      2620
   macro avg       0.30      0.50      0.38      2620
weighted avg       0.37      0.61      0.46      2620
```

In [62]:

```
# Testing
y_pred = lg_regression.predict(x_test)

cnf_matrix = confusion_matrix(y_test,y_pred)
print("confusion matrix\n",cnf_matrix)

print("*"*50)
accuracy = accuracy_score(y_test,y_pred)
print("accuarcy\n",accuracy)
print("*"*50)
clf_report = classification_report(y_test,y_pred)
print("classificatio report\n",clf_report)
```

```
confusion matrix
 [[400   0]
 [256   0]]
**************************************************
accuarcy
 0.6097560975609756
**************************************************
classificatio report
               precision    recall  f1-score   support

           0       0.61      1.00      0.76       400
           1       0.00      0.00      0.00       256

    accuracy                           0.61       656
   macro avg       0.30      0.50      0.38       656
weighted avg       0.37      0.61      0.46       656
```

Since here training and testing dataset both have almost similar accuracy , there is no overfitting issue. And hence no need of hyperparameter tuning.

But here we get almost just 60% accuracy and hence we have to also try fitting this dataset on some other classification algorithms as well, such as **KNN Classification** and **Decision Tree Classification** algorithm

KNN Classification algorithm

In [63]:

```
data.head()
```

Out[63]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbc
0	7.036752	204.890455	20791.318981	7.300212	368.516441	564.308654	10.3797
1	3.716080	129.422921	18630.057858	6.635246	334.564290	592.885359	15.1800
2	8.099124	224.236259	19909.541732	9.275884	334.564290	418.606213	16.8686
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.4365
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.5582

In [64]:

```
KNNClass = KNeighborsClassifier()
KNNClass.fit(x_train,y_train)
```

Out[64]:

```
KNeighborsClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Model Evaluation :

In [65]:

```
# Training
y_pred_train = KNNClass.predict(x_train)

cnf_matrix = confusion_matrix(y_train,y_pred_train)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuarcy = accuracy_score(y_train,y_pred_train)
print("Accuarcy",accuarcy)
print("*"*50)
clf_report = classification_report(y_train,y_pred_train)
print("Classification report\n",clf_report)
```

```
Confusion matrix
 [[1363  235]
 [ 526  496]]
**************************************************
Accuarcy 0.7095419847328245
**************************************************
Classification report
               precision    recall  f1-score   support

           0       0.72      0.85      0.78      1598
           1       0.68      0.49      0.57      1022

    accuracy                           0.71      2620
   macro avg       0.70      0.67      0.67      2620
weighted avg       0.70      0.71      0.70      2620
```

In [66]:

```
# Testing
y_pred = KNNClass.predict(x_test)

cnf_matrix = confusion_matrix(y_test,y_pred)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuarcy = accuracy_score(y_test,y_pred)
print("Accuarcy",accuarcy)
print("*"*50)
clf_report = classification_report(y_test,y_pred)
print("Classification report\n",clf_report)
```

```
Confusion matrix
 [[297 103]
 [181  75]]
**************************************************
Accuarcy 0.5670731707317073
**************************************************
Classification report
               precision    recall  f1-score   support

           0       0.62      0.74      0.68       400
           1       0.42      0.29      0.35       256

    accuracy                           0.57       656
   macro avg       0.52      0.52      0.51       656
weighted avg       0.54      0.57      0.55       656
```

Here there is a lot of difference in training and testing accuracy with training accuracy > testing accuracy --> overfitting issue

And hence it is necessary to do hyperparameter tuning using :

1) GridSearchCV

In [67]:

```
KNN_clf = KNeighborsClassifier()
hyperparameter = {"n_neighbors":np.arange(3,40),
                  "p":[1,2]}
gscv_knn_clf = GridSearchCV(KNN_clf,hyperparameter,cv = 10)
gscv_knn_clf.fit(x_train,y_train)
gscv_knn_clf.best_estimator_
```

Out[67]:

```
KNeighborsClassifier(n_neighbors=28)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [68]:

```
KNN_clf = KNeighborsClassifier(n_neighbors=28, p=2)
KNN_clf.fit(x_train,y_train)
```

Out[68]:

```
KNeighborsClassifier(n_neighbors=28)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Model Evaluation :

In [69]:

```
# Training
y_pred_train = KNN_clf.predict(x_train)

cnf_matrix = confusion_matrix(y_train,y_pred_train)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuarcy = accuracy_score(y_train,y_pred_train)
print("Accuarcy",accuarcy)
print("*"*50)
clf_report = classification_report(y_train,y_pred_train)
print("Classification report\n",clf_report)
```

```
Confusion matrix
 [[1515   83]
 [ 920  102]]
**************************************************
Accuarcy 0.617175572519084
**************************************************
Classification report
               precision    recall  f1-score   support

           0       0.62      0.95      0.75      1598
           1       0.55      0.10      0.17      1022

    accuracy                           0.62      2620
   macro avg       0.59      0.52      0.46      2620
weighted avg       0.59      0.62      0.52      2620
```

In [70]:

```
# Testing
y_pred = KNN_clf.predict(x_test)

cnf_matrix = confusion_matrix(y_test,y_pred)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuarcy = accuracy_score(y_test,y_pred)
print("Accuarcy",accuarcy)
print("*"*50)
clf_report = classification_report(y_test,y_pred)
print("Classification report\n",clf_report)
```

```
Confusion matrix
 [[385  15]
 [238  18]]
**************************************************
Accuarcy 0.614329268292683
**************************************************
Classification report
               precision    recall  f1-score   support

           0       0.62      0.96      0.75       400
           1       0.55      0.07      0.12       256

    accuracy                           0.61       656
   macro avg       0.58      0.52      0.44       656
weighted avg       0.59      0.61      0.51       656
```

In [ ]:

```

```

In [ ]:

```

```

2) RandomizedSearchCV

In [71]:

```
KNN_class = KNeighborsClassifier()
hyperparameter = {"n_neighbors":np.arange(3,30),
                  "p":[1,2]}
rscv_knn_clf = RandomizedSearchCV(KNN_class,hyperparameter,cv = 10)
rscv_knn_clf.fit(x_train,y_train)
rscv_knn_clf.best_estimator_
```

Out[71]:

```
KNeighborsClassifier(n_neighbors=24, p=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [72]:

```
KNN_class = KNeighborsClassifier(n_neighbors=24, p=1)
KNN_class.fit(x_train,y_train)
```

Out[72]:

```
KNeighborsClassifier(n_neighbors=24, p=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Model Evaluation:

In [73]:

```
# Training
y_pred_train = KNN_class.predict(x_train)

cnf_matrix = confusion_matrix(y_train,y_pred_train)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuarcy = accuracy_score(y_train,y_pred_train)
print("Accuarcy",accuarcy)
print("*"*50)
clf_report = classification_report(y_train,y_pred_train)
print("Classification report\n",clf_report)
```

```
Confusion matrix
 [[1526   72]
 [ 920  102]]
**************************************************
Accuarcy 0.6213740458015267
**************************************************
Classification report
               precision    recall  f1-score   support

           0       0.62      0.95      0.75      1598
           1       0.59      0.10      0.17      1022

    accuracy                           0.62      2620
   macro avg       0.61      0.53      0.46      2620
weighted avg       0.61      0.62      0.53      2620
```

In [74]:

```
# Testing
y_pred = KNN_class.predict(x_test)

cnf_matrix = confusion_matrix(y_test,y_pred)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuarcy = accuracy_score(y_test,y_pred)
print("Accuarcy",accuarcy)
print("*"*50)
clf_report = classification_report(y_test,y_pred)
print("Classification report\n",clf_report)
```

```
Confusion matrix
 [[380  20]
 [238  18]]
**************************************************
Accuarcy 0.6067073170731707
**************************************************
Classification report
               precision    recall  f1-score   support

           0       0.61      0.95      0.75       400
           1       0.47      0.07      0.12       256

    accuracy                           0.61       656
   macro avg       0.54      0.51      0.43       656
weighted avg       0.56      0.61      0.50       656
```

In [ ]:

```

```

In [ ]:

```

```

Here the **Solids**feature has a wide range of values and higher values when compared to values of other features. And since the KNN Classifier is a distance based algorithm, the report or the model prediction is affected due to this difference in the values. And hence it is necessary to convert the data in all the column in a similar range.

Here since almost all the features follow normal distribution as observed earlier , we cannot use normalisation instead we can use standardization which is actually feasible and is also not sensitive to outliers present if any.

Standardization:

In [75]:

```
std_scalar = StandardScaler()
array  = std_scalar.fit_transform(x)
x_std_df = pd.DataFrame(array,columns=x.columns)
x_std_df
```

Out[75]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon
0	-0.025474	0.259195	-0.100740	0.112415	0.961017	1.747584	-1.180651
1	-2.284717	-2.036414	-0.366351	-0.307694	0.021560	2.107322	0.270597
2	0.697319	0.847665	-0.209107	1.360594	0.021560	-0.086588	0.781117
3	0.845393	0.547651	0.050066	0.592008	0.639206	-0.783231	1.255134
4	1.372982	-0.464429	-0.446366	-0.363698	-0.654379	-0.340818	-0.824357
...	...	...	...	...	...	...	...
3271	-1.637002	-0.081758	-0.110822	0.028027	0.723943	1.270676	-0.118075
3272	0.499833	-0.085667	-0.526148	0.593290	-0.033706	-0.415860	1.698560
3273	1.595654	-0.626829	1.418785	0.144017	-0.033706	0.082583	-0.981329
3274	-1.324949	1.041355	-1.183145	-0.517373	-0.033706	-0.284518	-0.942064
3275	0.544611	-0.038546	-0.517008	0.244515	-0.033706	-1.233984	0.560940

3276 rows × 9 columns

Train test split

In [76]:

```
x_train,x_test,y_train,y_test = train_test_split(x_std_df,y,test_size=0.2,random_sta
```

Model Evaluation :

In [77]:

```
knn_clf = KNeighborsClassifier()
knn_clf.fit(x_train,y_train)
```

Out[77]:

```
KNeighborsClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [78]:

```
# Training
y_pred_train = knn_clf.predict(x_train)

cnf_matrix = confusion_matrix(y_train,y_pred_train)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuarcy = accuracy_score(y_train,y_pred_train)
print("Accuarcy",accuarcy)
print("*"*50)
clf_report = classification_report(y_train,y_pred_train)
print("Classification report\n",clf_report)
```

```
Confusion matrix
 [[1400  198]
 [ 424  598]]
**************************************************
Accuarcy 0.7625954198473283
**************************************************
Classification report
               precision    recall  f1-score   support

           0       0.77      0.88      0.82      1598
           1       0.75      0.59      0.66      1022

    accuracy                           0.76      2620
   macro avg       0.76      0.73      0.74      2620
weighted avg       0.76      0.76      0.76      2620
```

In [79]:

```
# Testing
y_pred = knn_clf.predict(x_test)

cnf_matrix = confusion_matrix(y_test,y_pred)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuarcy = accuracy_score(y_test,y_pred)
print("Accuarcy",accuarcy)
print("*"*50)
clf_report = classification_report(y_test,y_pred)
print("Classification report\n",clf_report)
```

```
Confusion matrix
 [[324  76]
 [173  83]]
**************************************************
Accuarcy 0.6204268292682927
**************************************************
Classification report
               precision    recall  f1-score   support

           0       0.65      0.81      0.72       400
           1       0.52      0.32      0.40       256

    accuracy                           0.62       656
   macro avg       0.59      0.57      0.56       656
weighted avg       0.60      0.62      0.60       656
```

GridSearchCV

In [80]:

```
knn_class = KNeighborsClassifier()
hyperparameter = {"n_neighbors":np.arange(3,40),
                  "p":[1,2]}
gscv_knn_clf = GridSearchCV(knn_class,hyperparameter,cv = 10)
gscv_knn_clf.fit(x_train,y_train)
gscv_knn_clf.best_estimator_
```

Out[80]:

```
KNeighborsClassifier(n_neighbors=33)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [81]:

```
knn_class = KNeighborsClassifier(n_neighbors=33, p=2)
knn_class.fit(x_train,y_train)
```

Out[81]:

```
KNeighborsClassifier(n_neighbors=33)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Model training:

In [82]:

```
# Training
y_pred_train = knn_class.predict(x_train)

cnf_matrix = confusion_matrix(y_train,y_pred_train)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuarcy = accuracy_score(y_train,y_pred_train)
print("Accuarcy",accuarcy)
print("*"*50)
clf_report = classification_report(y_train,y_pred_train)
print("Classification report\n",clf_report)
```

```
Confusion matrix
 [[1527   71]
 [ 771  251]]
**************************************************
Accuarcy 0.6786259541984733
**************************************************
Classification report
               precision    recall  f1-score   support

           0       0.66      0.96      0.78      1598
           1       0.78      0.25      0.37      1022

    accuracy                           0.68      2620
   macro avg       0.72      0.60      0.58      2620
weighted avg       0.71      0.68      0.62      2620
```

In [83]:

```
# Testing
y_pred = knn_class.predict(x_test)

cnf_matrix = confusion_matrix(y_test,y_pred)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuarcy = accuracy_score(y_test,y_pred)
print("Accuarcy",accuarcy)
print("*"*50)
clf_report = classification_report(y_test,y_pred)
print("Classification report\n",clf_report)
```

```
Confusion matrix
 [[377  23]
 [216  40]]
**************************************************
Accuarcy 0.635670731707317
**************************************************
Classification report
               precision    recall  f1-score   support

           0       0.64      0.94      0.76       400
           1       0.63      0.16      0.25       256

    accuracy                           0.64       656
   macro avg       0.64      0.55      0.51       656
weighted avg       0.64      0.64      0.56       656
```

RandomizedSearchCV

In [84]:

```
knnclass = KNeighborsClassifier()
hyperparameter = {"n_neighbors":np.arange(3,40),
                  "p":[1,2]}
rscv_knn_clf = RandomizedSearchCV(knnclass,hyperparameter,cv = 10)
rscv_knn_clf.fit(x_train,y_train)
rscv_knn_clf.best_estimator_
```

Out[84]:

```
KNeighborsClassifier(n_neighbors=33)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [85]:

```
knnclass = KNeighborsClassifier(n_neighbors=33, p=2)
knnclass.fit(x_train,y_train)
```

Out[85]:

```
KNeighborsClassifier(n_neighbors=33)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Model Evaluation:

In [86]:

```
# Training
y_pred_train = knnclass.predict(x_train)

cnf_matrix = confusion_matrix(y_train,y_pred_train)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuarcy = accuracy_score(y_train,y_pred_train)
print("Accuarcy",accuarcy)
print("*"*50)
clf_report = classification_report(y_train,y_pred_train)
print("Classification report\n",clf_report)
```

```
Confusion matrix
 [[1527   71]
 [ 771  251]]
**************************************************
Accuarcy 0.6786259541984733
**************************************************
Classification report
               precision    recall  f1-score   support

           0       0.66      0.96      0.78      1598
           1       0.78      0.25      0.37      1022

    accuracy                           0.68      2620
   macro avg       0.72      0.60      0.58      2620
weighted avg       0.71      0.68      0.62      2620
```

In [87]:

```
# Testing
y_pred = knnclass.predict(x_test)

cnf_matrix = confusion_matrix(y_test,y_pred)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuarcy = accuracy_score(y_test,y_pred)
print("Accuarcy",accuarcy)
print("*"*50)
clf_report = classification_report(y_test,y_pred)
print("Classification report\n",clf_report)
```

```
Confusion matrix
 [[377  23]
 [216  40]]
**************************************************
Accuarcy 0.635670731707317
**************************************************
Classification report
               precision    recall  f1-score   support

           0       0.64      0.94      0.76       400
           1       0.63      0.16      0.25       256

    accuracy                           0.64       656
   macro avg       0.64      0.55      0.51       656
weighted avg       0.64      0.64      0.56       656
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

Decision Tree Classification algorithm

Model training :

In [88]:

```
dt_clf = DecisionTreeClassifier()
dt_clf.fit(x_train,y_train)
```

Out[88]:

```
DecisionTreeClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Model Evaluation:

In [89]:

```
# Training
y_pred_train = dt_clf.predict(x_train)

cnf_matrix = confusion_matrix(y_train,y_pred_train)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuracy = accuracy_score(y_train,y_pred_train)
print("accuarcy\n",accuracy)
print("*"*50)
clf_report = classification_report(y_train,y_pred_train)
print("classification report\n",clf_report)
```

```
Confusion matrix
 [[1598    0]
 [   0 1022]]
**************************************************
accuarcy
 1.0
**************************************************
classification report
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      1598
           1       1.00      1.00      1.00      1022

    accuracy                           1.00      2620
   macro avg       1.00      1.00      1.00      2620
weighted avg       1.00      1.00      1.00      2620
```

In [90]:

```
# Testing
y_pred = dt_clf.predict(x_test)

cnf_matrix = confusion_matrix(y_test,y_pred)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuracy = accuracy_score(y_test,y_pred)
print("accuarcy\n",accuracy)
print("*"*50)
clf_report = classification_report(y_test,y_pred)
print("classification report\n",clf_report)
```

```
Confusion matrix
 [[302  98]
 [105 151]]
**************************************************
accuarcy
 0.6905487804878049
**************************************************
classification report
               precision    recall  f1-score   support

           0       0.74      0.76      0.75       400
           1       0.61      0.59      0.60       256

    accuracy                           0.69       656
   macro avg       0.67      0.67      0.67       656
weighted avg       0.69      0.69      0.69       656
```

```
plt.figure(figsize=(200,150))
plot_tree(dt_clf,feature_names=x.columns,class_names=["0","1"],filled=True)
plt.savefig("Decision_Tree_without Hyperparameter tuining.png")
```

In [91]:

```
a=dt_clf.max_depth
a
```

Overfitting issue

- Hyperparameter tuning

In [92]:

```
dt_model = DecisionTreeClassifier()
hyperparameters = {"criterion":["gini","entropy"],
                   "max_depth": np.arange(3,8),
                   "min_samples_split":np.arange(2,10),
                   "min_samples_leaf":np.arange(2,15)}
gscvdt_clf = GridSearchCV(dt_model,hyperparameters,cv=10)
gscvdt_clf.fit(x_train,y_train)
gscvdt_clf.best_estimator_
```

Out[92]:

```
DecisionTreeClassifier(max_depth=6, min_samples_leaf=4, min_samples_split=
6)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [93]:

```
dt_model= DecisionTreeClassifier(criterion='gini', max_depth=6, min_samples_leaf=4,
                       min_samples_split=6)
dt_model.fit(x_train,y_train)
```

Out[93]:

```
DecisionTreeClassifier(max_depth=6, min_samples_leaf=4, min_samples_split=
6)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Model Evaluation:

In [94]:

```
# Training
y_pred_train = dt_model.predict(x_train)

cnf_matrix = confusion_matrix(y_train,y_pred_train)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuracy = accuracy_score(y_train,y_pred_train)
print("accuarcy\n",accuracy)
print("*"*50)
clf_report = classification_report(y_train,y_pred_train)
print("classification report\n",clf_report)
```

```
Confusion matrix
 [[1456  142]
 [ 433  589]]
**************************************************
accuarcy
 0.7805343511450382
**************************************************
classification report
               precision    recall  f1-score   support

           0       0.77      0.91      0.84      1598
           1       0.81      0.58      0.67      1022

    accuracy                           0.78      2620
   macro avg       0.79      0.74      0.75      2620
weighted avg       0.78      0.78      0.77      2620
```

In [95]:

```
# Testing
y_pred = dt_model.predict(x_test)

cnf_matrix = confusion_matrix(y_test,y_pred)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuracy = accuracy_score(y_test,y_pred)
print("accuarcy\n",accuracy)
print("*"*50)
clf_report = classification_report(y_test,y_pred)
print("classification report\n",clf_report)

```

```
Confusion matrix
 [[357  43]
 [127 129]]
**************************************************
accuarcy
 0.7408536585365854
**************************************************
classification report
               precision    recall  f1-score   support

           0       0.74      0.89      0.81       400
           1       0.75      0.50      0.60       256

    accuracy                           0.74       656
   macro avg       0.74      0.70      0.71       656
weighted avg       0.74      0.74      0.73       656
```

In [ ]:

```

```

RandomForest Classification Algorithm

In [96]:

```
rf_classifier = RandomForestClassifier()
rf_classifier.fit(x_train,y_train)
```

Out[96]:

```
RandomForestClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Model Evaluation :

In [97]:

```
# Training
y_pred_train = rf_classifier.predict(x_train)

cnf_matrix = confusion_matrix(y_train,y_pred_train)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuracy = accuracy_score(y_train,y_pred_train)
print("accuarcy\n",accuracy)
print("*"*50)
clf_report = classification_report(y_train,y_pred_train)
print("classification report\n",clf_report)
```

```
Confusion matrix
 [[1598    0]
 [   0 1022]]
**************************************************
accuarcy
 1.0
**************************************************
classification report
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      1598
           1       1.00      1.00      1.00      1022

    accuracy                           1.00      2620
   macro avg       1.00      1.00      1.00      2620
weighted avg       1.00      1.00      1.00      2620
```

In [98]:

```
# Testing
y_pred = rf_classifier.predict(x_test)

cnf_matrix = confusion_matrix(y_test,y_pred)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuracy = accuracy_score(y_test,y_pred)
print("accuarcy\n",accuracy)
print("*"*50)
clf_report = classification_report(y_test,y_pred)
print("classification report\n",clf_report)
```

```
Confusion matrix
 [[354  46]
 [117 139]]
**************************************************
accuarcy
 0.7515243902439024
**************************************************
classification report
               precision    recall  f1-score   support

           0       0.75      0.89      0.81       400
           1       0.75      0.54      0.63       256

    accuracy                           0.75       656
   macro avg       0.75      0.71      0.72       656
weighted avg       0.75      0.75      0.74       656
```

In [ ]:

```

```

GridSearchCV - Randomforest

In [99]:

```
rfc = RandomForestClassifier(random_state=42, n_jobs=-1)
params = {
    'max_depth': [2,10,20],
    'min_samples_leaf': [5,7,10],
    'n_estimators': [50,100,200,500,700],
    'random_state':[42]}
grid_rfc = GridSearchCV(RandomForestClassifier(), params , scoring = "accuracy", cv=
grid_rfc.fit(x_train, y_train)
rfc_params = grid_rfc.best_estimator_
rfc_params
```

Out[99]:

```
RandomForestClassifier(max_depth=10, min_samples_leaf=10, n_estimators=20
0,
                       random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [100]:

```
rfc= RandomForestClassifier(max_depth=10, min_samples_leaf=10, n_estimators=200,
                       random_state=42)
rfc.fit(x_train,y_train)
```

Out[100]:

```
RandomForestClassifier(max_depth=10, min_samples_leaf=10, n_estimators=20
0,
                       random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Model Evaluation:

In [101]:

```
# Training
y_pred_train = rfc.predict(x_train)

cnf_matrix = confusion_matrix(y_train,y_pred_train)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuracy = accuracy_score(y_train,y_pred_train)
print("accuarcy\n",accuracy)
print("*"*50)
clf_report = classification_report(y_train,y_pred_train)
print("classification report\n",clf_report)
```

```
Confusion matrix
 [[1552   46]
 [ 363  659]]
**************************************************
accuarcy
 0.8438931297709924
**************************************************
classification report
               precision    recall  f1-score   support

           0       0.81      0.97      0.88      1598
           1       0.93      0.64      0.76      1022

    accuracy                           0.84      2620
   macro avg       0.87      0.81      0.82      2620
weighted avg       0.86      0.84      0.84      2620
```

In [102]:

```
# Testing
y_pred = rfc.predict(x_test)

cnf_matrix = confusion_matrix(y_test,y_pred)
print("Confusion matrix\n",cnf_matrix)
print("*"*50)
accuracy = accuracy_score(y_test,y_pred)
print("accuarcy\n",accuracy)
print("*"*50)
clf_report = classification_report(y_test,y_pred)
print("classification report\n",clf_report)
```

```
Confusion matrix
 [[364  36]
 [130 126]]
**************************************************
accuarcy
 0.7469512195121951
**************************************************
classification report
               precision    recall  f1-score   support

           0       0.74      0.91      0.81       400
           1       0.78      0.49      0.60       256

    accuracy                           0.75       656
   macro avg       0.76      0.70      0.71       656
weighted avg       0.75      0.75      0.73       656
```

In [ ]:

```

```

In [ ]:

```
Models = ["Logistic Regression","KNN Classifier","KNN GSCV","KNN RSCV","KNN after ST
Testing_Accuracy = [0.6097,0.5670,0.6143,0.6143,0.6204,0.6356,0.6356,0.6951,0.7408,0
Training_Accuracy = [ 0.6099,0.7095,0.6171,0.6312,0.7625,0.6786,0.6786,1.0,0.7805,1.
```

In [ ]:

```
df = pd.DataFrame({'Models':Models ,'Training_Accuracy': Training_Accuracy,'Testing_
df
```

In [ ]:

```
index = np.arange(11)
bar_width = 0.35

fig, ax = plt.subplots()
Training_Accuracy = ax.bar(index, df["Training_Accuracy"], bar_width,
                label="Training Accuracy")

Testing_Accuracy = ax.bar(index+bar_width, df["Testing_Accuracy"],
                 bar_width, label="Testing_Accuracy")

ax.set_xlabel('Models')
ax.set_ylabel('Accuracies')
ax.set_title('Models with their accuracies')
ax.set_xticks(index + bar_width/2 )
ax.set_xticklabels(Models)
ax.legend()
plt.show()
```

Out of all the predictions KNN RSCV model can be used for prediction of water quality analysis

User input

In [103]:

```
x.columns
```

Out[103]:

```
Index(['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivit
y',
       'Organic_carbon', 'Trihalomethanes', 'Turbidity'],
      dtype='object')
```

In [104]:

```
data.iloc[0]
```

Out[104]:

```
ph                    7.036752
Hardness            204.890455
Solids            20791.318981
Chloramines           7.300212
Sulfate             368.516441
Conductivity        564.308654
Organic_carbon       10.379783
Trihalomethanes      86.990970
Turbidity             2.963135
Potability            0.000000
Name: 0, dtype: float64
```

In [105]:

```
Columns_list = {"Columns":list(x.columns)}
```

In [106]:

```
import json
with open("Labelled_columns.json","w") as f:
    json.dump(Columns_list,f)
```

In [107]:

```
ph              =       7.036752
Hardness        =     204.890455
Solids          =   20791.318981
Chloramines     =       7.300212
Sulfate         =     368.516441
Conductivity    =     564.308654
Organic_carbon  =      10.379783
Trihalomethanes =      86.990970
Turbidity       =       2.963135
```

In [108]:

```
array = np.zeros(len(x.columns),dtype=float)
array[0] = ph
array[1] = Hardness
array[2] = Solids
array[3] = Chloramines
array[4] = Sulfate
array[5] = Conductivity
array[6] = Organic_carbon
array[7] = Trihalomethanes
array[8] = Turbidity
```

In [109]:

```
array
```

Out[109]:

```
array([7.03675200e+00, 2.04890455e+02, 2.07913190e+04, 7.30021200e+00,
       3.68516441e+02, 5.64308654e+02, 1.03797830e+01, 8.69909700e+01,
       2.96313500e+00])
```

In [110]:

```
Water_Potability_prediction = KNN_clf.predict([array])
if Water_Potability_prediction==1:
    print("The above sample of water is potable i.e is eligible for consumption.")
else:
    print("The above sample of water is not potable i.e is not eligible for consumpt
```

```
The above sample of water is not potable i.e is not eligible for consumpti
on.
```

In [111]:

```
import pickle
with open("KNN_clf.pkl","wb") as file:
    pickle.dump(KNN_class,file)
```

In [112]:

```
!pip show scikit-learn
```

```
Name: scikit-learn
Version: 1.2.2
Summary: A set of python modules for machine learning and data mining
Home-page: http://scikit-learn.org (http://scikit-learn.org)
Author:
Author-email:
License: new BSD
Location: C:\Users\yargu\AppData\Local\Programs\Python\Python311\Lib\site-
packages
Requires: joblib, numpy, scipy, threadpoolctl
Required-by: imbalanced-learn
```

In [113]:

```
!pip show python
```

```
WARNING: Package(s) not found: python
```

In [114]:

```
from platform import python_version
python_version()
```

Out[114]:

```
'3.9.7'
```

In [115]:

```
!pip show numpy
```

```
Name: numpy
Version: 1.24.2
Summary: Fundamental package for array computing in Python
Home-page: https://www.numpy.org (https://www.numpy.org)
Author: Travis E. Oliphant et al.
Author-email:
License: BSD-3-Clause
Location: C:\Users\yargu\AppData\Local\Programs\Python\Python311\Lib\site-
packages
Requires:
Required-by: contourpy, imbalanced-learn, matplotlib, pandas, scikit-lear
n, scipy, seaborn
```

In [116]:

```
!pip show pandas
```

```
Name: pandas
Version: 1.5.3
Summary: Powerful data structures for data analysis, time series, and stat
istics
Home-page: https://pandas.pydata.org (https://pandas.pydata.org)
Author: The Pandas Development Team
Author-email: pandas-dev@python.org
License: BSD-3-Clause
Location: C:\Users\yargu\AppData\Local\Programs\Python\Python311\Lib\site-
packages
Requires: numpy, numpy, python-dateutil, pytz
Required-by: seaborn
```

In [117]:

```
!pip show flask
```

```
Name: Flask
Version: 2.2.2
Summary: A simple framework for building complex web applications.
Home-page: https://palletsprojects.com/p/flask (https://palletsprojects.co
m/p/flask)
Author: Armin Ronacher
Author-email: armin.ronacher@active-4.com
License: BSD-3-Clause
Location: C:\Users\yargu\AppData\Local\Programs\Python\Python311\Lib\site-
packages
Requires: click, itsdangerous, Jinja2, Werkzeug
Required-by: Flask-MySQLdb
```

In [ ]:

```

```