In [123]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split,RandomizedSearchCV
from sklearn.linear_model import LinearRegression,LogisticRegression,RidgeClassifier
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
from sklearn.ensemble import RandomForestClassifier
```

In [2]:

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

Out[2]:

	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

EDA

In [3]:

```
df.shape
```

Out[3]:

(3276, 10)

In [4]:

```
df.isna().sum()
```

Out[4]:

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

In [5]:

```
df.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
```

In [6]:

```
df.columns
```

Out[6]:

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

working on null values

In [7]:

```
df.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["ph"].fillna(df["ph"].mean(),inplace=True)
```

In [9]:

```
df["Sulfate"].fillna(df["Sulfate"].mean(), inplace=True)
```

In [10]:

```
df["Trihalomethanes"].fillna(df["Trihalomethanes"].mean(),inplace=True)
```

In [11]:

```
df.isna().sum()
```

Out[11]:

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

In [12]:

```
df.describe()
```

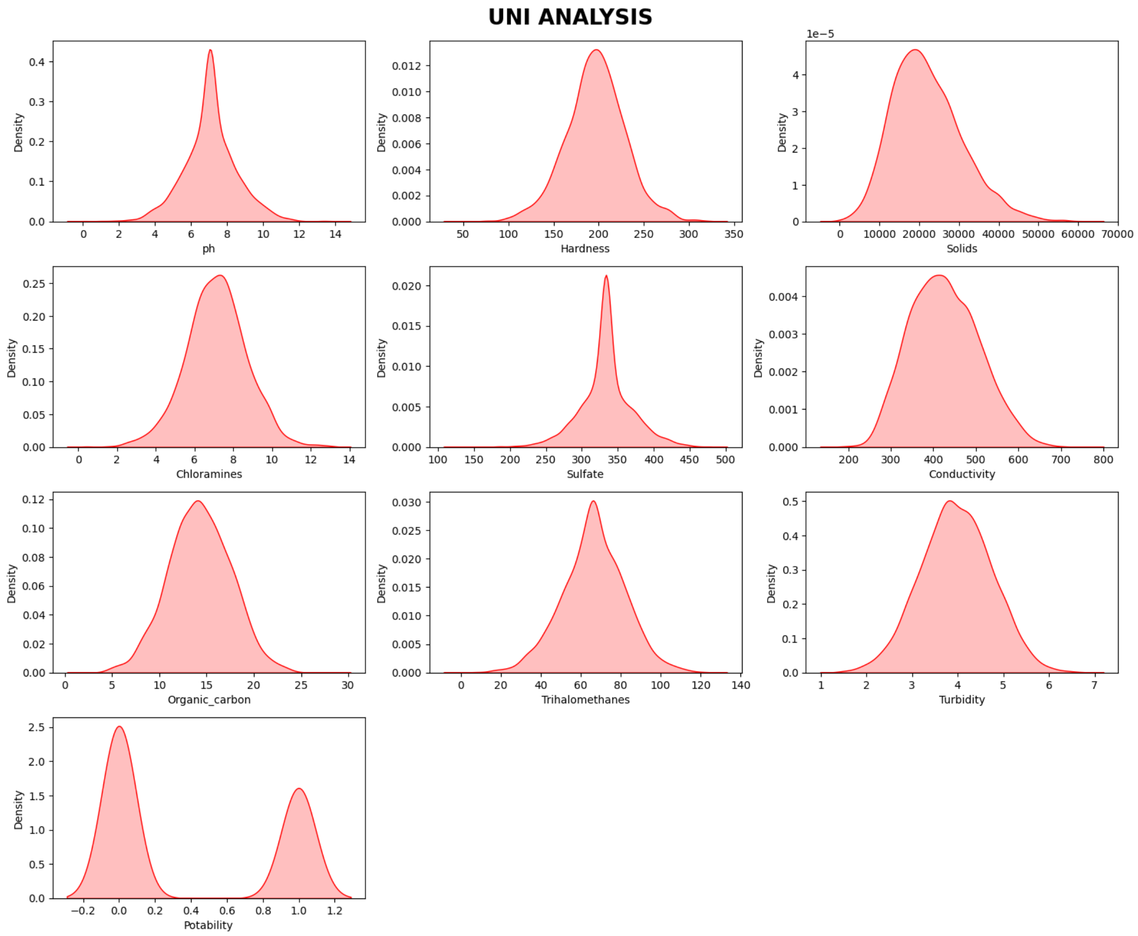
Out[12]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Org
count	3276.000000	3276.000000	3276.000000	3276.000000	3276.000000	3276.000000	
mean	7.080795	196.369496	22014.092526	7.122277	333.775777	426.205111	
std	1.469956	32.879761	8768.570828	1.583085	36.142612	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%	7.080795	196.967627	20927.833607	7.130299	333.775777	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	

univariate analysis

In [13]:

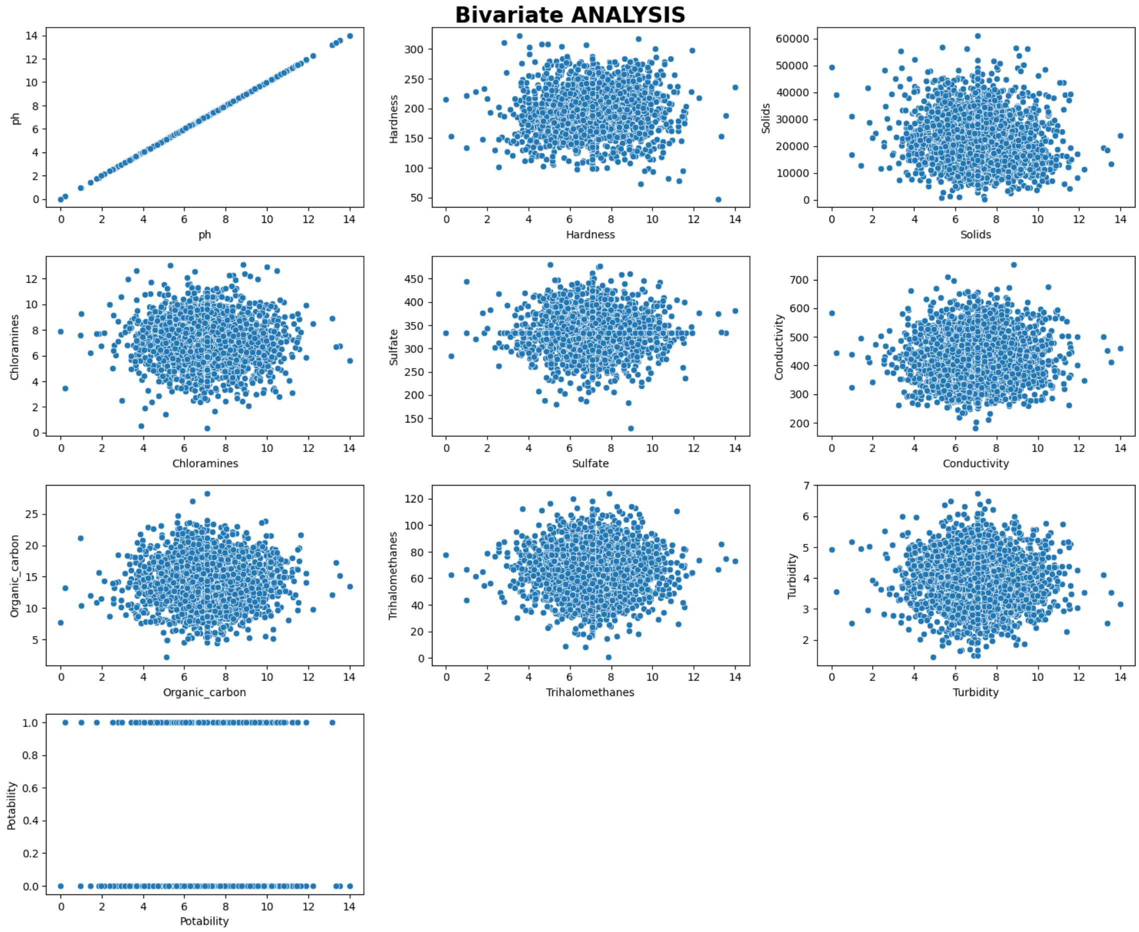
```
plt.figure(figsize=(15,15))
plt.suptitle("UNI ANALYSIS", fontsize=20 , fontweight="bold")
for i in range(len(df.columns)):
    plt.subplot(5,3,i+1)
    sns.kdeplot(x=df[df.columns[i]], shade=True , color = "r")
    plt.xlabel(df.columns[i])
    plt.tight_layout()
```



Bivariate Analysis

In [14]:

```
plt.figure(figsize=(15,15))
plt.suptitle("Bivariate ANALYSIS", fontsize=20 , fontweight="bold")
for i in range(len(df.columns)):
    plt.subplot(5,3,i+1)
    sns.scatterplot(x=df["ph"], y=df[df.columns[i]])
    plt.xlabel(df.columns[i])
    plt.tight_layout()
```



In [15]:

```
df.columns
```

Out[15]:

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

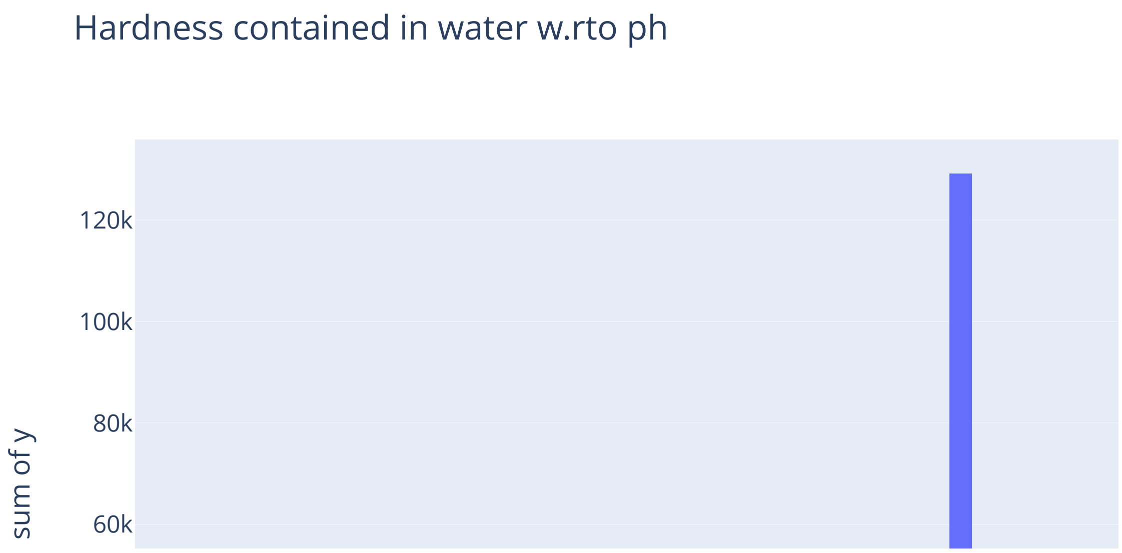
INSIGHTS

In [16]:

```
import plotly.express as px
```

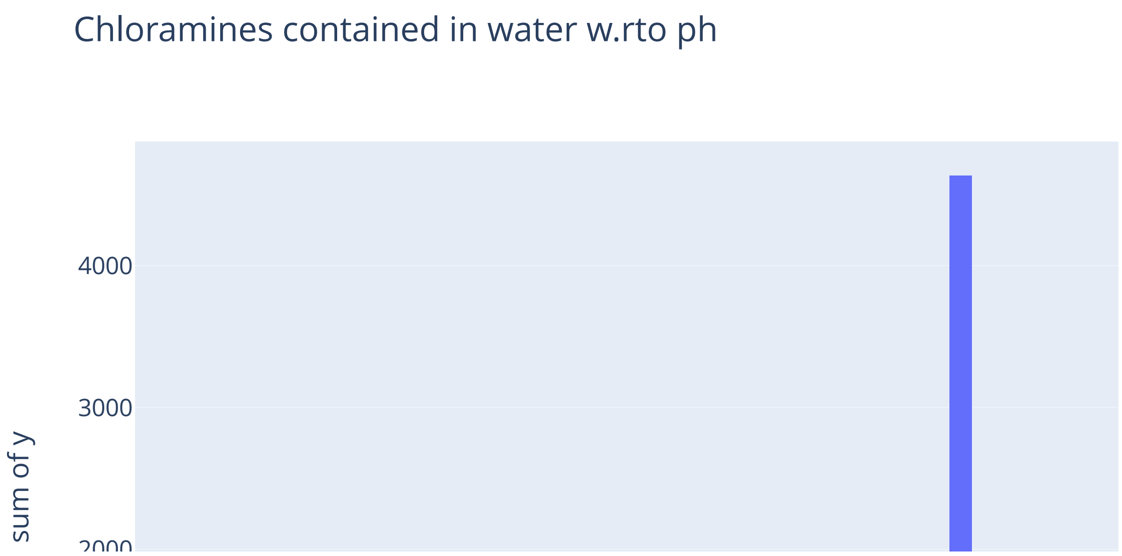
In [17]:

```
fig = px.histogram(x= df["ph"], y=df["Hardness"], title= "Hardness contained in water w.
fig.show()
```



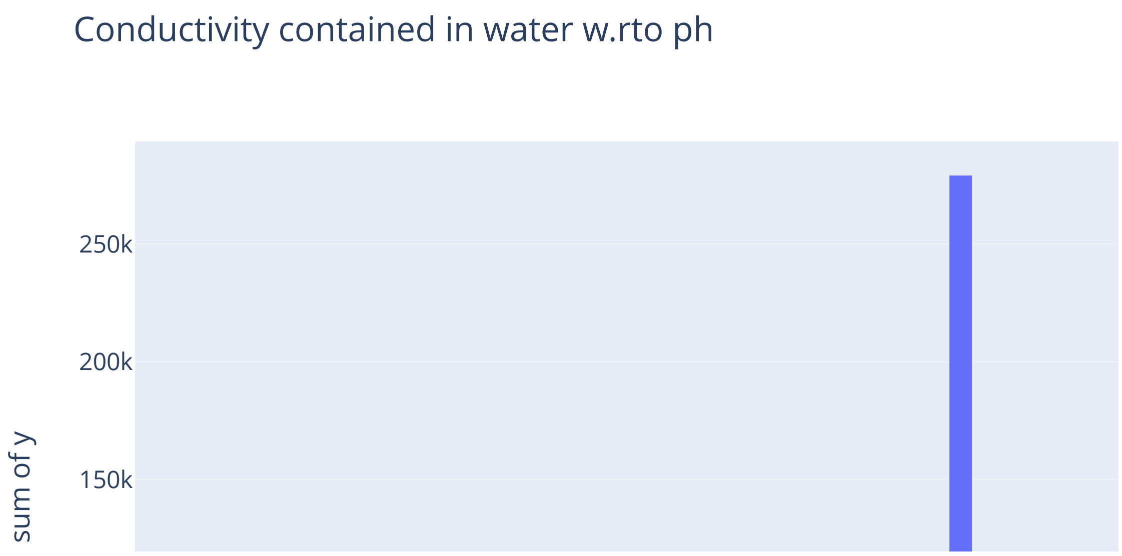
In [18]:

```
fig = px.histogram(x= df["ph"], y=df["Chloramines"], title= "Chloramines contained in wa
fig.show()
```



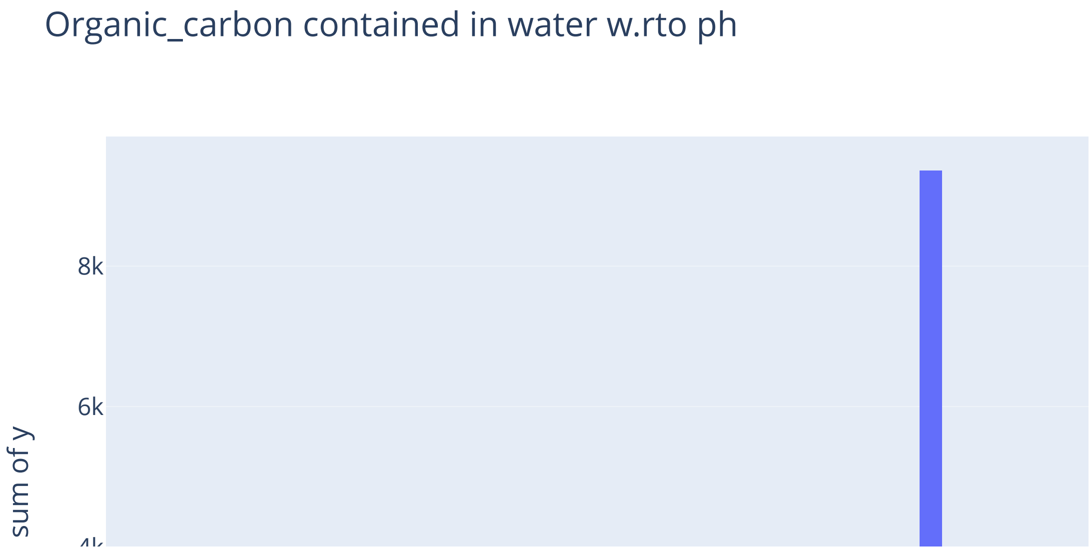
In [19]:

```
fig = px.histogram(x= df["ph"], y=df["Conductivity"], title= "Conductivity contained in 
fig.show()
```



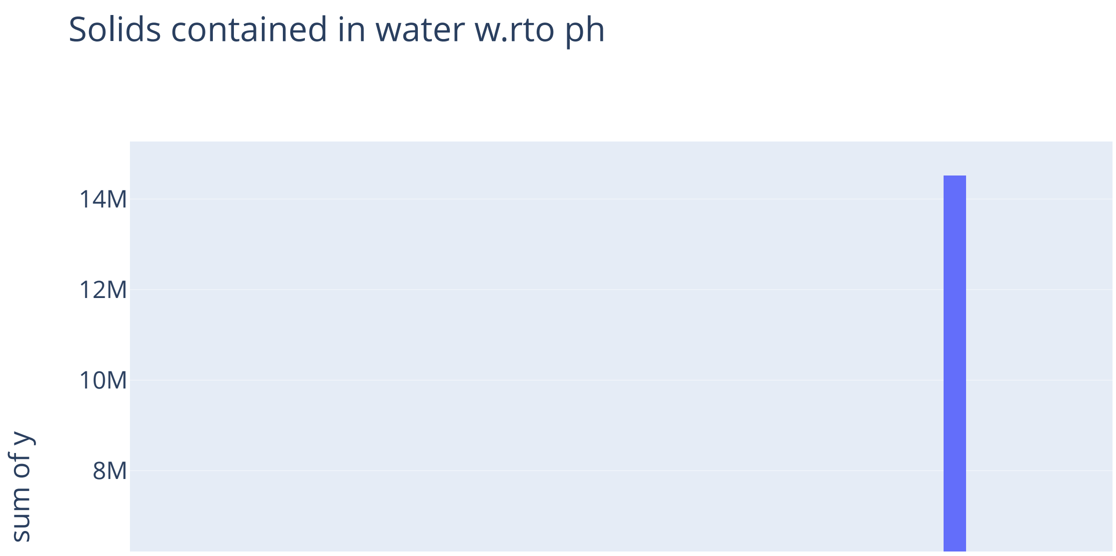
In [20]:

```
fig = px.histogram(x= df["ph"], y=df["Organic_carbon"], title= "Organic_carbon contained
fig.show()
```



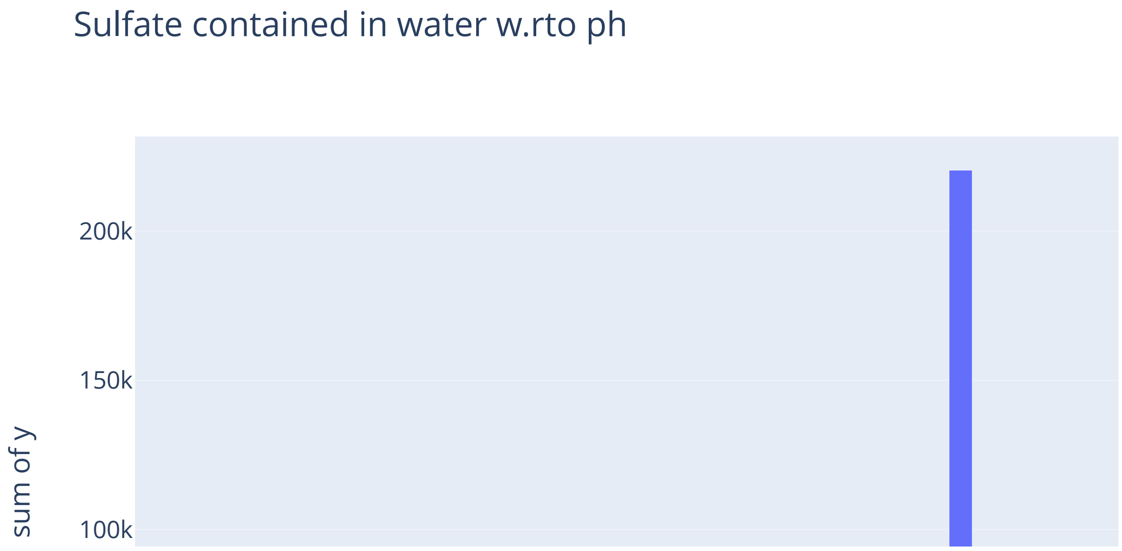
In [21]:

```
fig = px.histogram(x= df["ph"], y=df["Solids"], title= "Solids contained in water w.rto
fig.show()
```



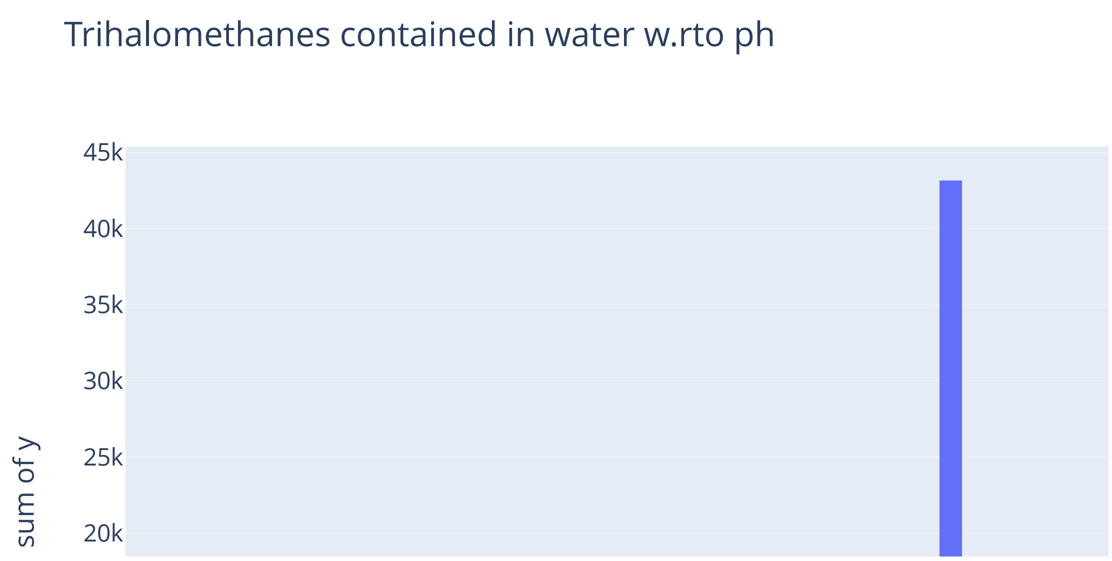
In [22]:

```
fig = px.histogram(x= df["ph"], y=df["Sulfate"], title= "Sulfate contained in water w.rt
fig.show()
```



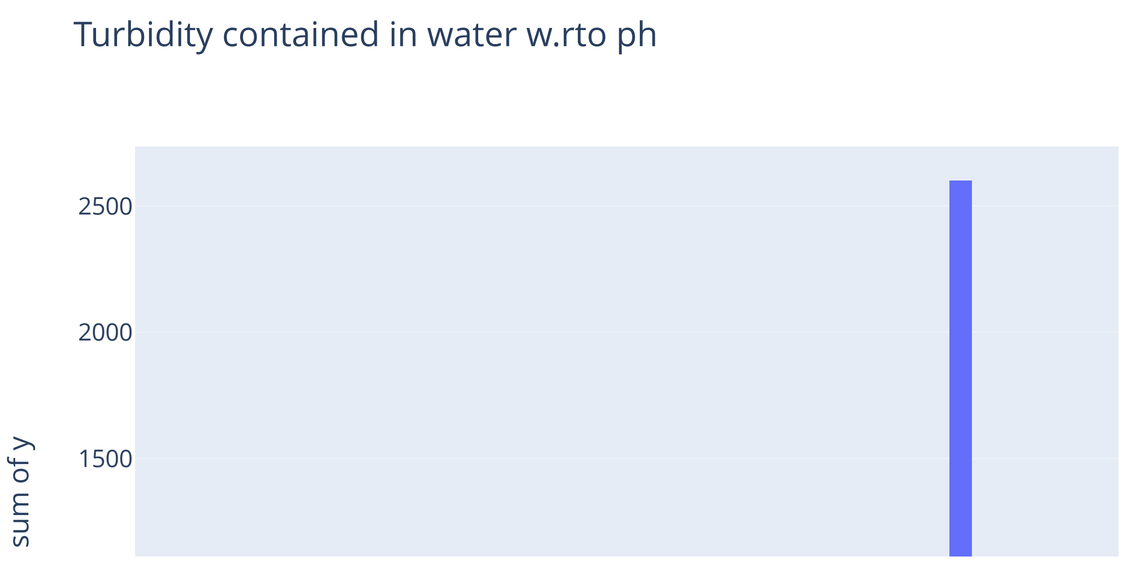
In [23]:

```
fig = px.histogram(x= df["ph"], y=df["Trihalomethanes"], title= "Trihalomethanes contain
fig.show()
```



In [24]:

```
fig = px.histogram(x= df["ph"], y=df["Turbidity"], title= "Turbidity contained in water 
fig.show()
```



In [25]:

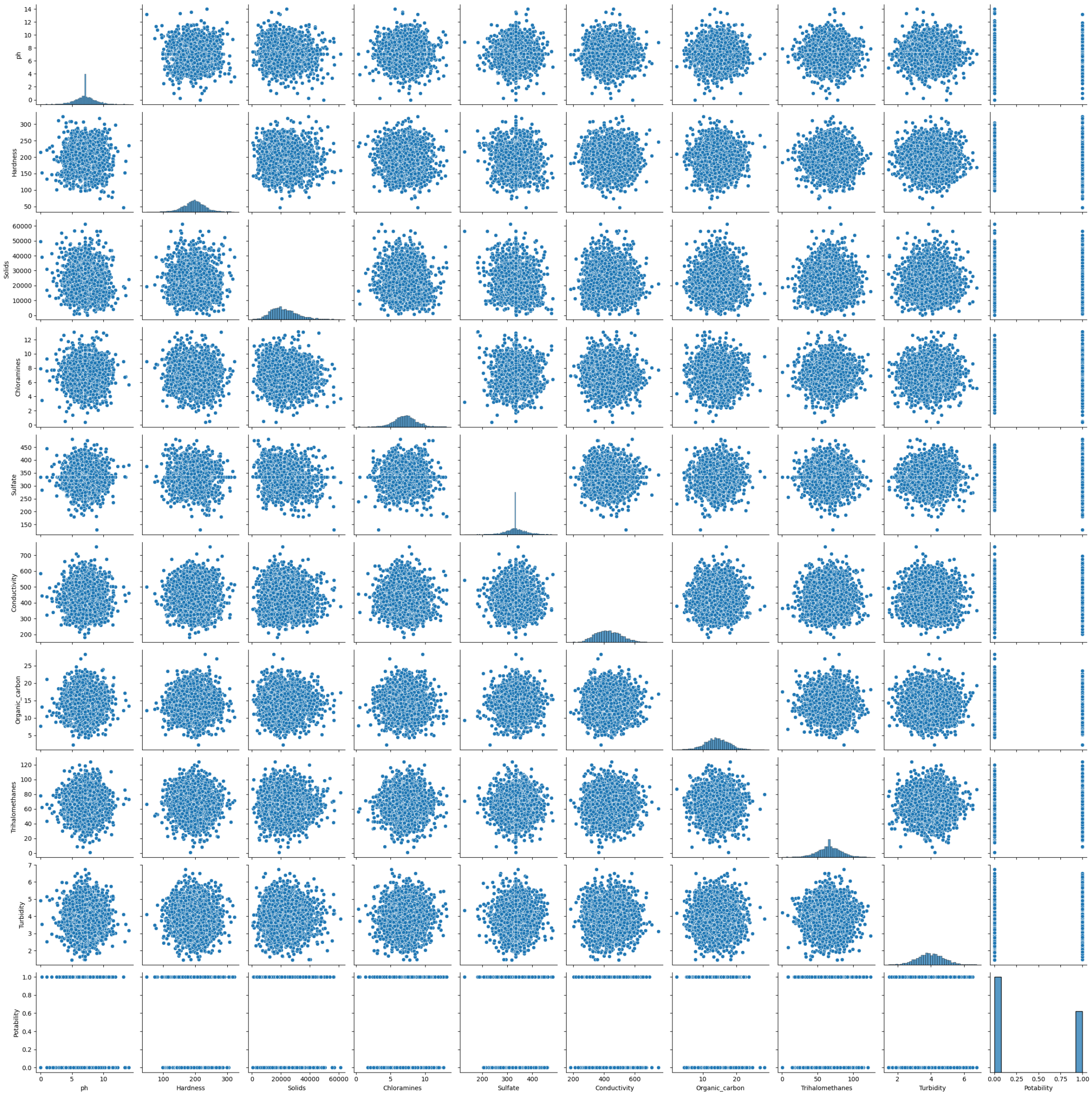
```
# fig = px.bar(df,df["Potability"] , y= df["ph"])
# fig.show()
```

In [26]:

```
sns.pairplot(df)
```

Out[26]:

<seaborn.axisgrid.PairGrid at 0x1f75f3d7130>

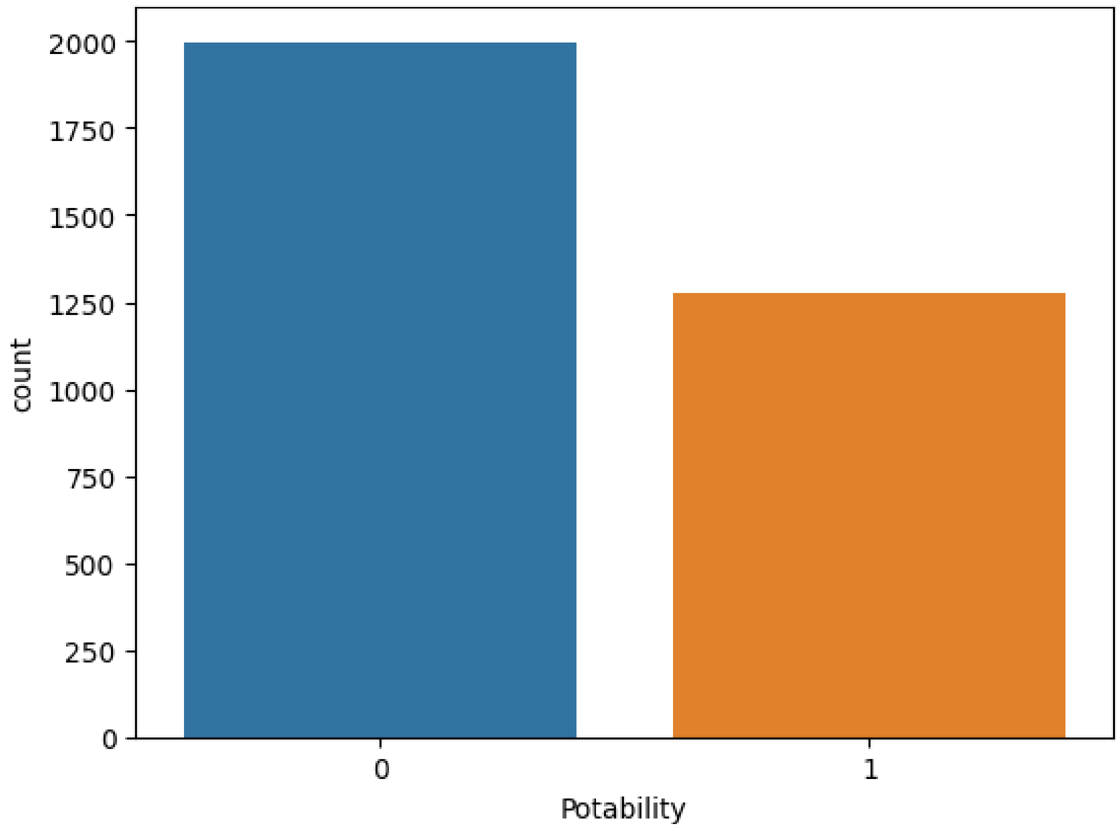


In [27]:

```
sns.countplot(x= df["Potability"])
```

Out[27]:

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



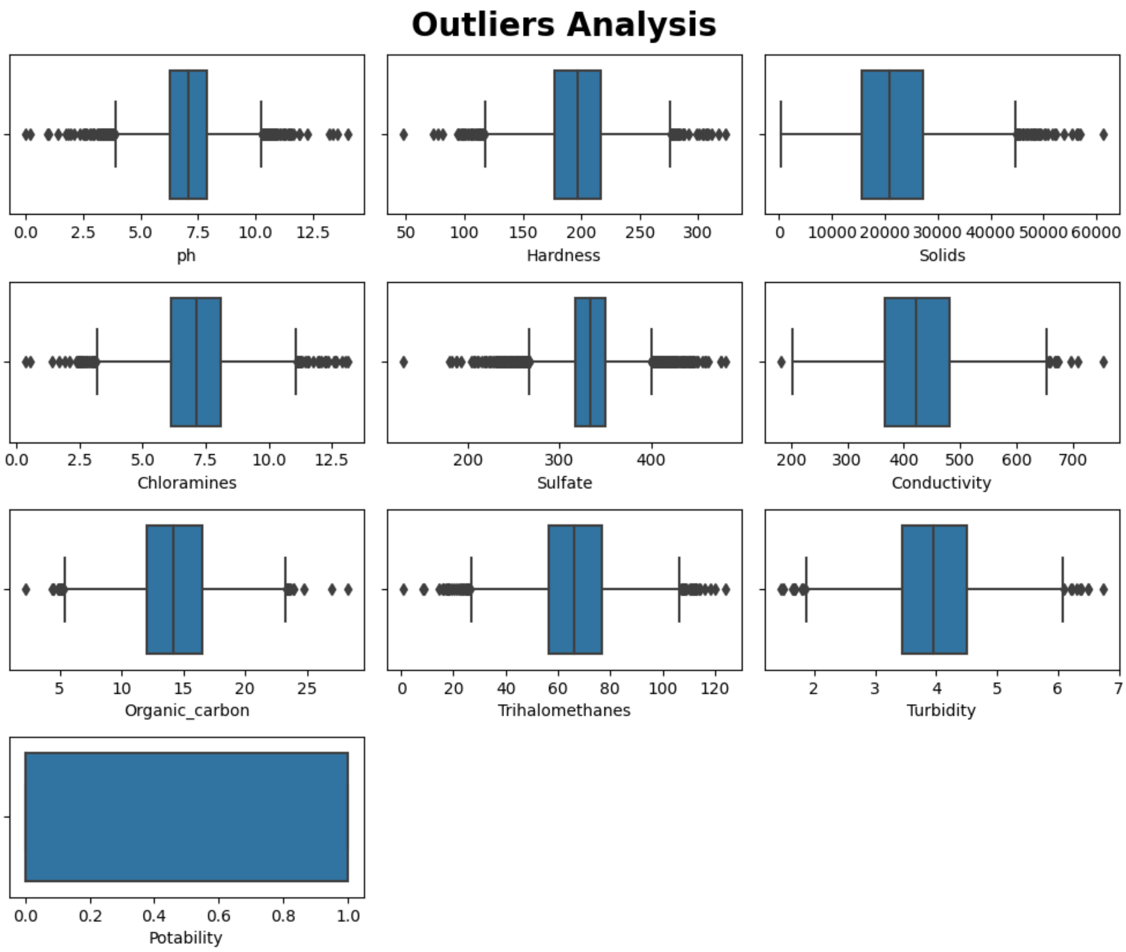
Target column is high

Feature Engineering

Detecting Outliers

In [28]:

```
plt.figure(figsize=(10,10))
plt.suptitle("Outliers Analysis", fontsize=20, fontweight="bold")
for i in range(0,len(df.columns)):
    plt.subplot(5,3,i+1)
    sns.boxplot(x=df[df.columns[i]])
    plt.xlabel(df.columns[i])
    plt.tight_layout()
```



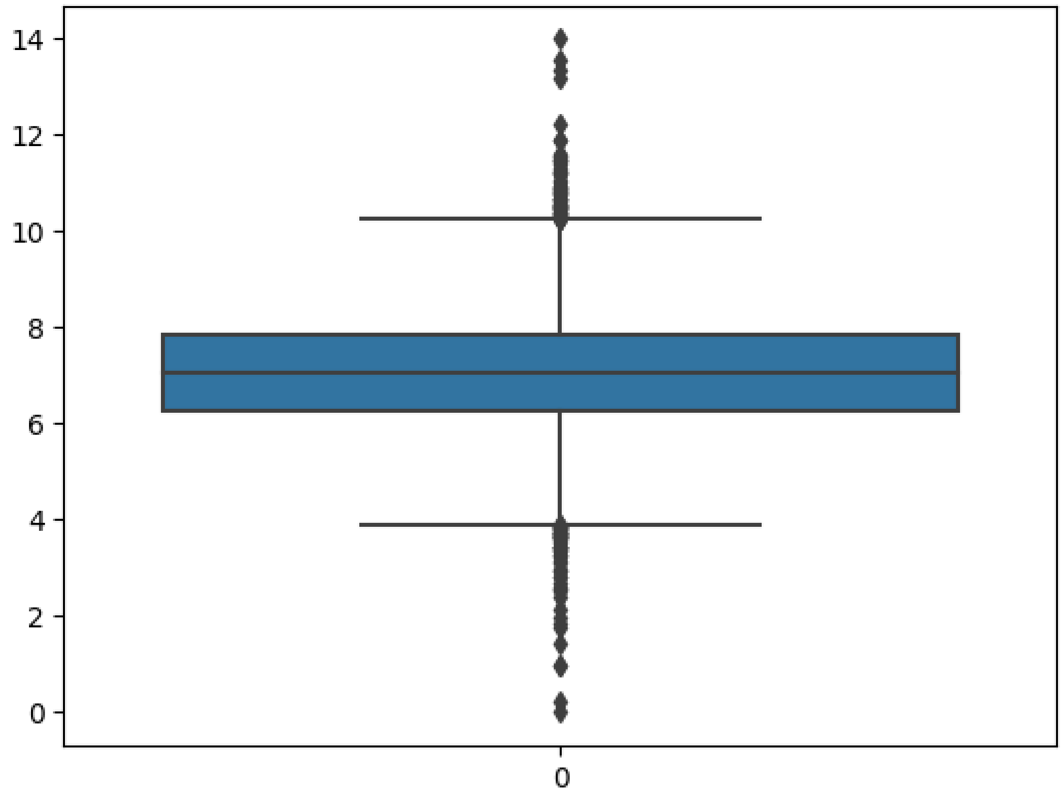
ph

In [29]:

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

Out[29]:

<AxesSubplot: >



In [30]:

```
df["ph"].describe()
```

Out[30]:

```
count    3276.000000
mean        7.080795
std         1.469956
min         0.000000
25%         6.277673
50%         7.080795
75%         7.870050
max        14.000000
Name: ph, dtype: float64
```

In [31]:

```
print(df["ph"].quantile(0.10))
print(df["ph"].quantile(0.90))
```

```
5.282194491912188
8.925046875415749
```

In [32]:

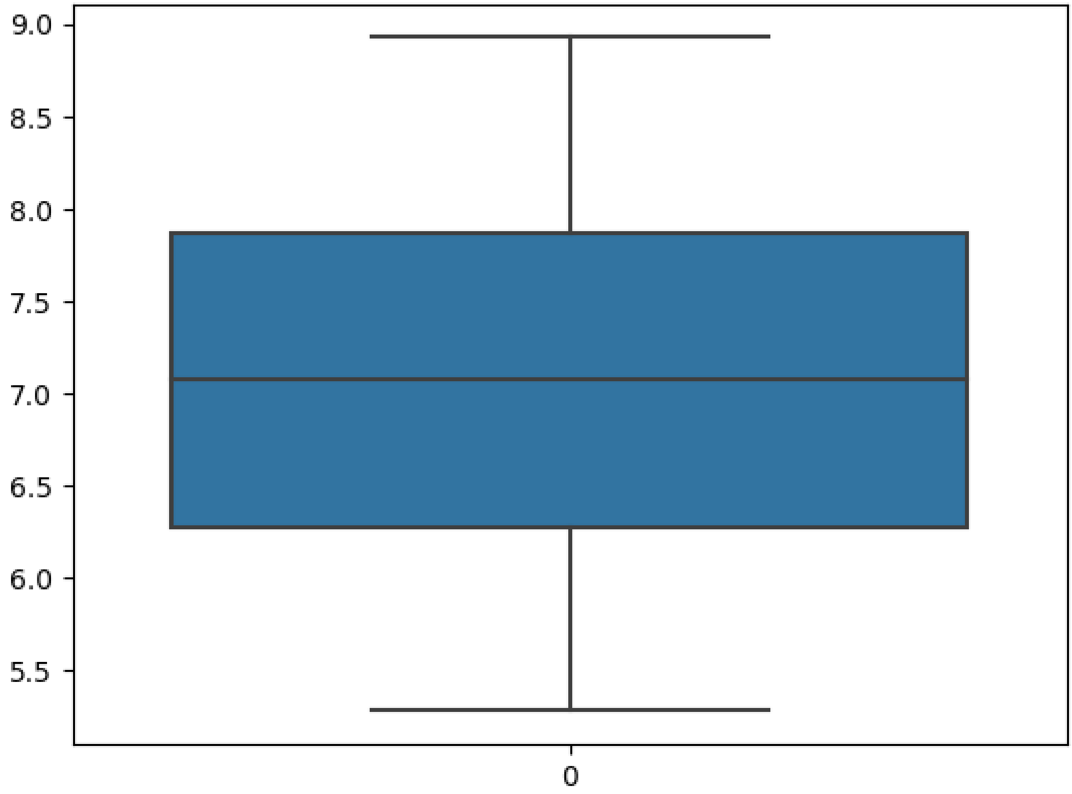
```
df["ph"] = np.where(df["ph"] <5.282194491912188, 5.282194491912188,df["ph"])
df["ph"] = np.where(df["ph"]>8.925046875415749, 8.925046875415749,df["ph"])
```

In [33]:

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

Out[33]:

```
<AxesSubplot: >
```



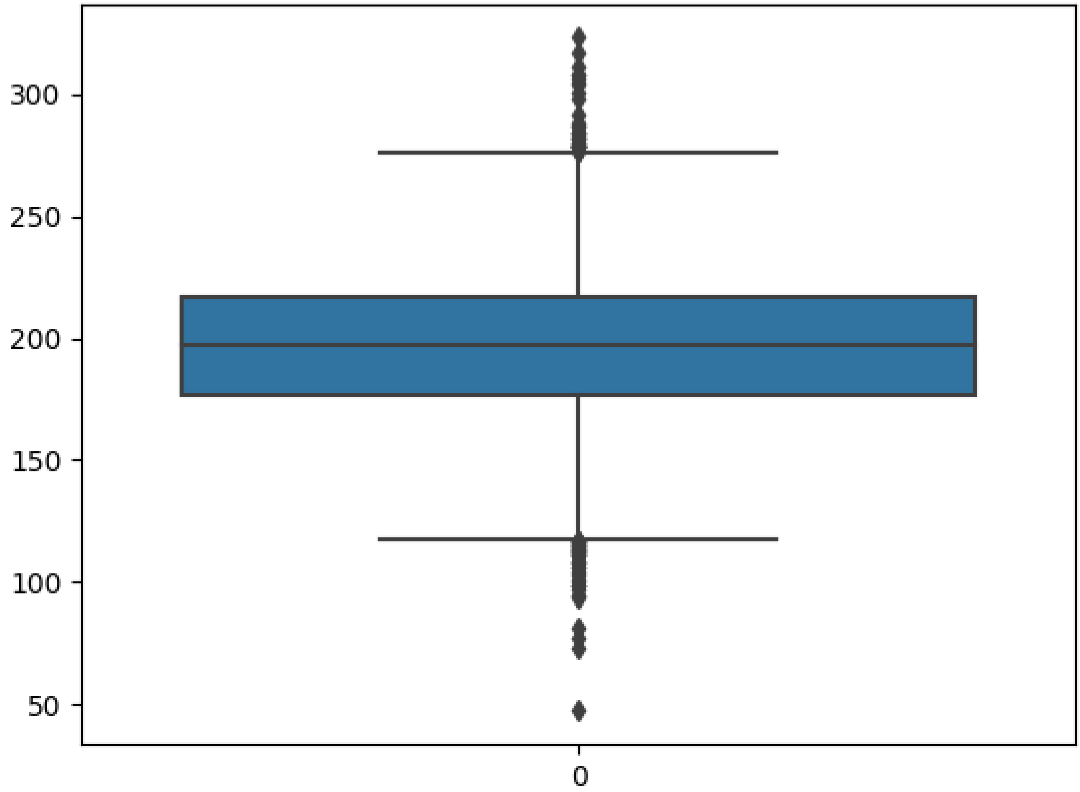
Hardness

In [34]:

```
sns.boxplot(df["Hardness"])
```

Out[34]:

```
<AxesSubplot: >
```



In [35]:

```
df["Hardness"].describe()
```

Out[35]:

```
count    3276.000000
mean      196.369496
std        32.879761
min        47.432000
25%       176.850538
50%       196.967627
75%       216.667456
max       323.124000
Name: Hardness, dtype: float64
```

In [36]:

```
print(df["Hardness"].quantile(0.10))
print(df["Hardness"].quantile(0.90))
```

```
155.2239641077801
236.35070740017414
```

In [37]:

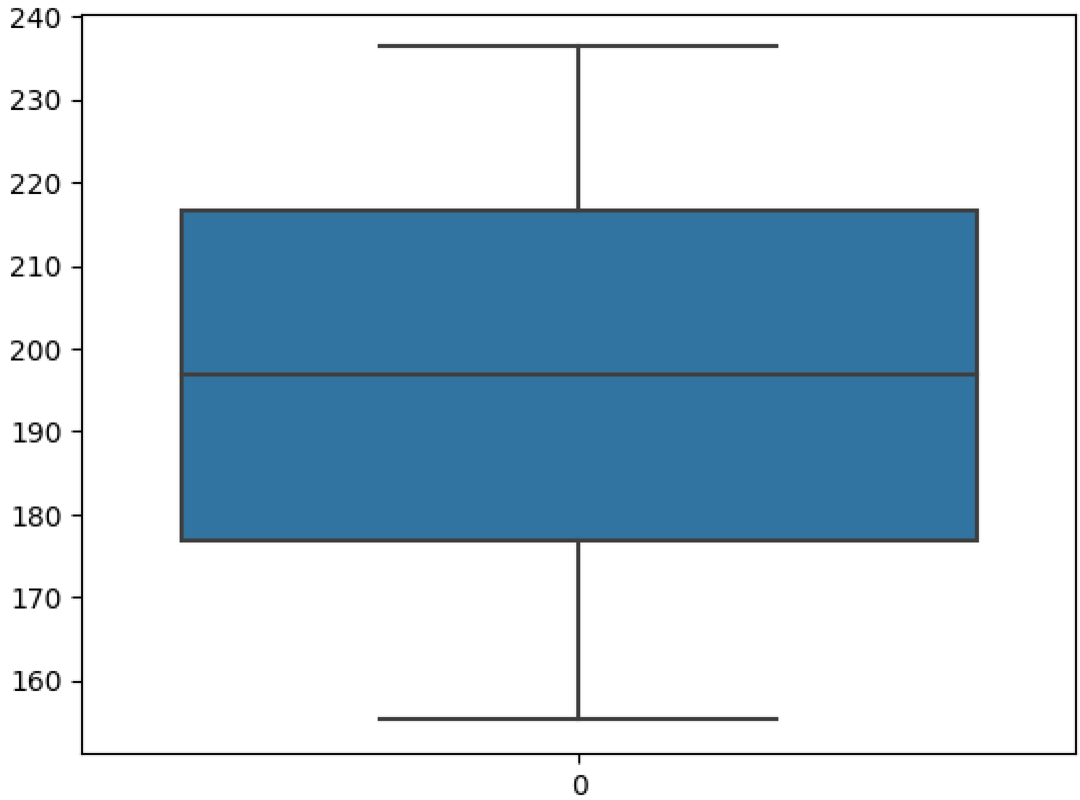
```
df["Hardness"] = np.where(df["Hardness"] <155.2239641077801, 155.2239641077801,df["Hardn
df["Hardness"] = np.where(df["Hardness"]>236.35070740017414, 236.35070740017414,df["Hard
```

In [38]:

```
sns.boxplot(df["Hardness"])
```

Out[38]:

```
<AxesSubplot: >
```



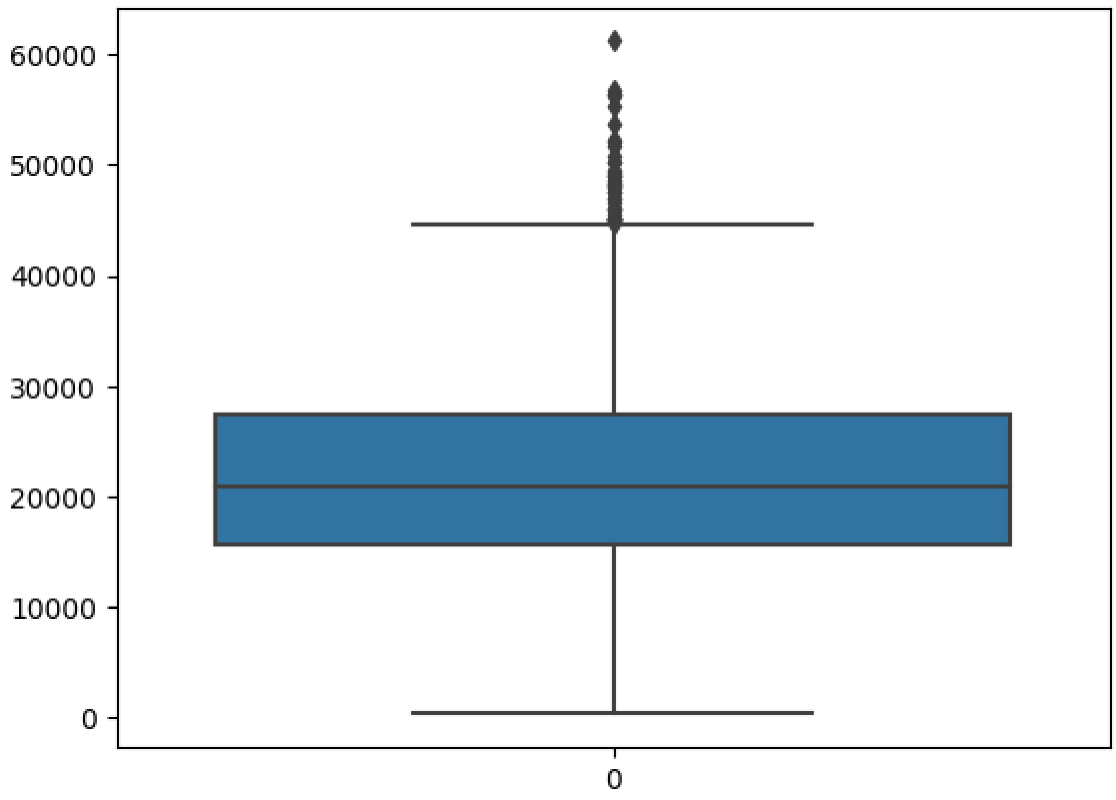
Solids

In [39]:

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

Out[39]:

```
<AxesSubplot: >
```



In [40]:

```
df["Solids"].describe()
```

Out[40]:

```
count     3276.000000
mean     22014.092526
std       8768.570828
min        320.942611
25%      15666.690297
50%      20927.833607
75%      27332.762127
max      61227.196008
Name: Solids, dtype: float64
```

In [41]:

```
print(df["Solids"].quantile(0.10))
print(df["Solids"].quantile(0.90))
```

```
11740.528189473214
33814.93523020222
```

In [42]:

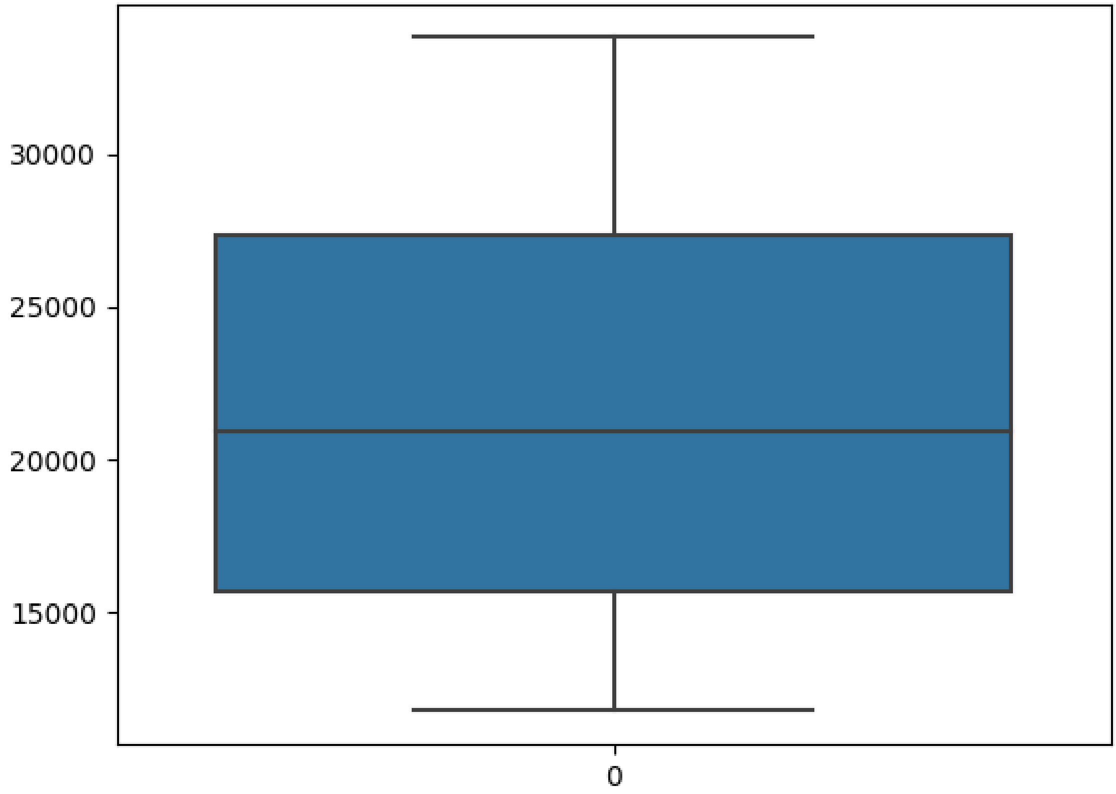
```
df["Solids"] = np.where(df["Solids"] <11740.528189473214, 11740.528189473214,df["Solids"]
df["Solids"] = np.where(df["Solids"]>33814.93523020222,33814.93523020222,df["Solids"])
```

In [43]:

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

Out[43]:

```
<AxesSubplot: >
```



In [44]:

```
df.columns
```

Out[44]:

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

Chloramines

In [45]:

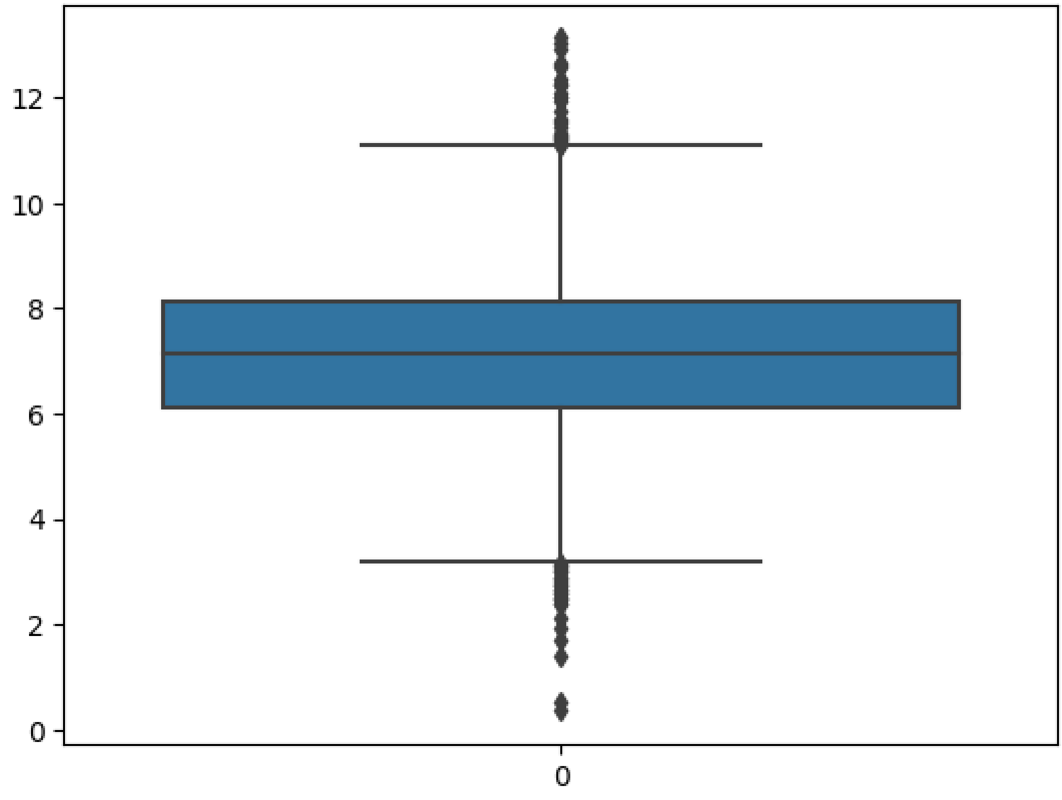
```
sns.boxplot(df["Chloramines"])

print(df["Chloramines"].quantile(0.10))
print(df["Chloramines"].quantile(0.90))
df["Chloramines"].describe()
```

```
5.181270677724534
9.122578323075329
```

Out[45]:

```
count    3276.000000
mean        7.122277
std         1.583085
min         0.352000
25%         6.127421
50%         7.130299
75%         8.114887
max        13.127000
Name: Chloramines, dtype: float64
```

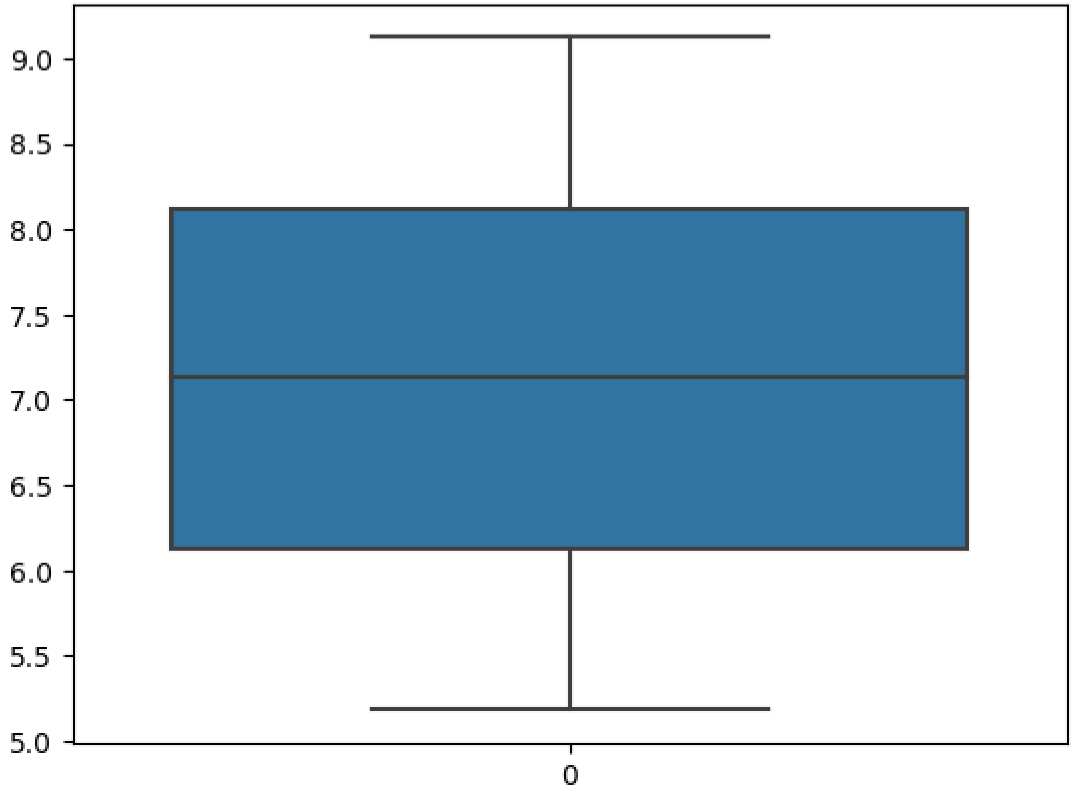


In [46]:

```
df["Chloramines"] = np.where(df["Chloramines"] <5.181270677724534, 5.181270677724534,df[
df["Chloramines"] = np.where(df["Chloramines"]>9.122578323075329,9.122578323075329,df["C
sns.boxplot(df["Chloramines"])
```

Out[46]:

<AxesSubplot: >



Sulfate

In [47]:

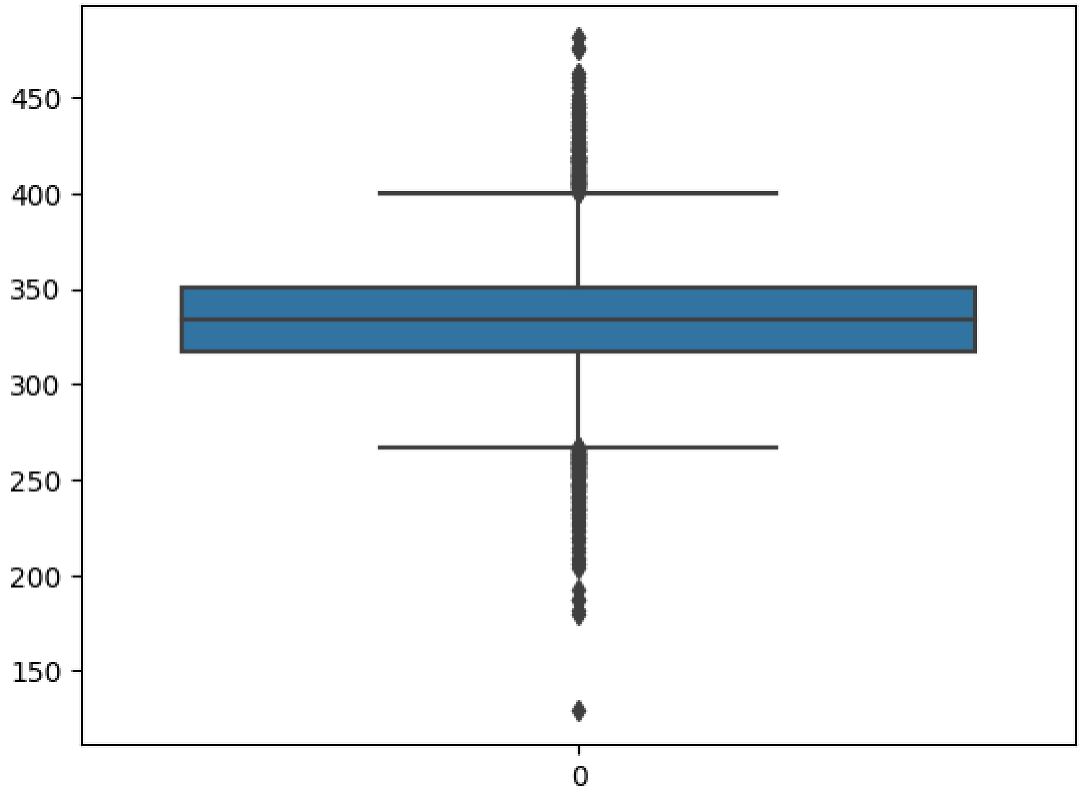
```
sns.boxplot(df["Sulfate"])

print(df["Sulfate"].quantile(0.10))
print(df["Sulfate"].quantile(0.90))
df["Sulfate"].describe()
```

```
290.055010521933
378.4783210497187
```

Out[47]:

```
count    3276.000000
mean      333.775777
std        36.142612
min       129.000000
25%       317.094638
50%       333.775777
75%       350.385756
max       481.030642
Name: Sulfate, dtype: float64
```

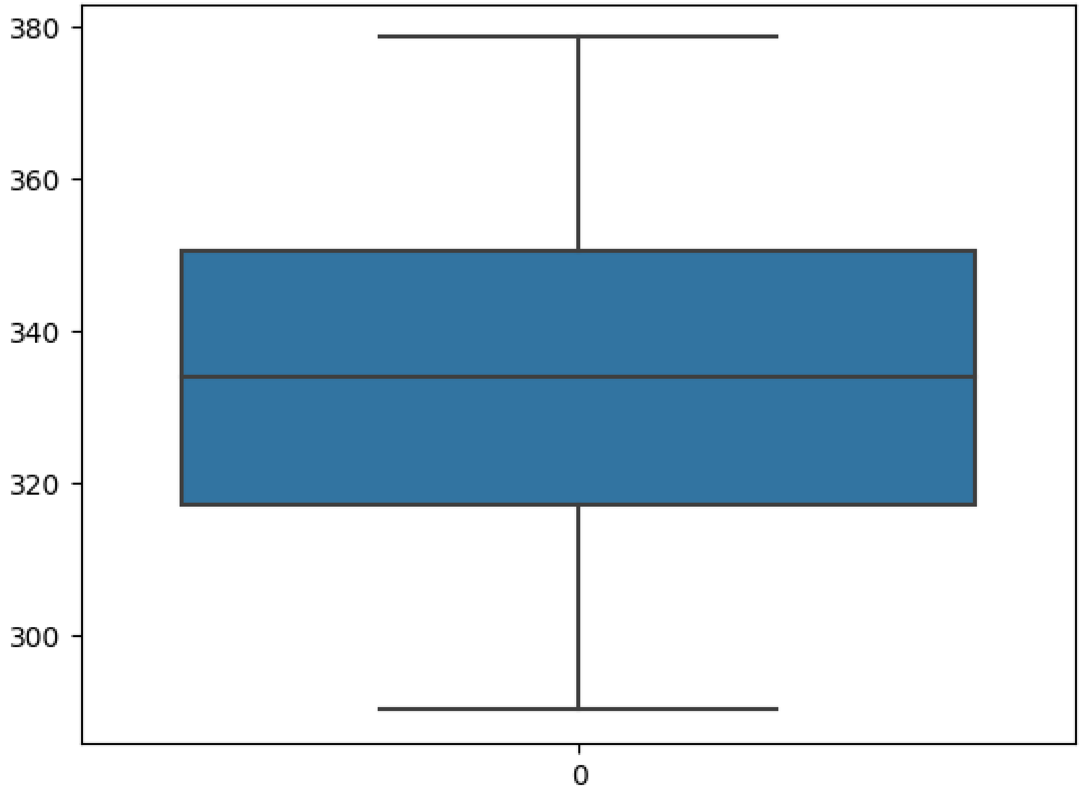


In [48]:

```
df["Sulfate"] = np.where(df["Sulfate"] <290.055010521933, 290.055010521933,df["Sulfate"]
df["Sulfate"] = np.where(df["Sulfate"]>378.4783210497187, 378.4783210497187,df["Sulfate"
sns.boxplot(df["Sulfate"])
```

Out[48]:

```
<AxesSubplot: >
```



Conductivity

In [49]:

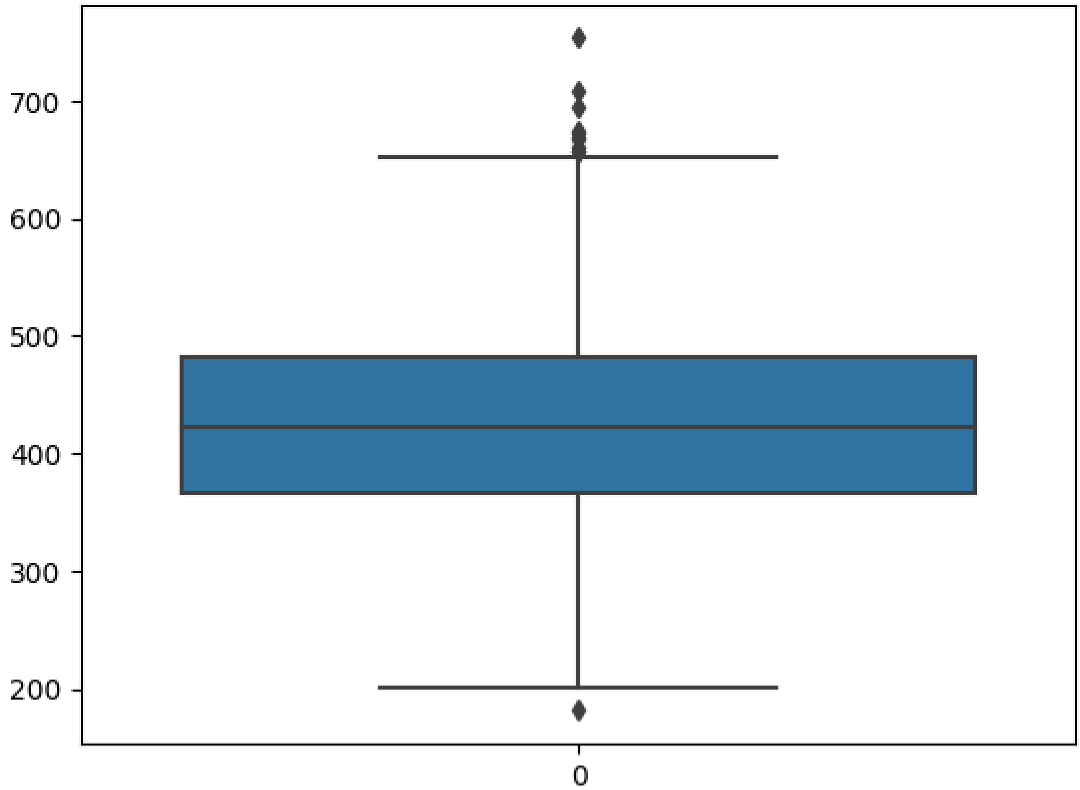
```
sns.boxplot(df["Conductivity"])

print(df["Conductivity"].quantile(0.10))
print(df["Conductivity"].quantile(0.90))
df["Conductivity"].describe()
```

```
325.1171240676143
533.2972414189196
```

Out[49]:

```
count    3276.000000
mean      426.205111
std        80.824064
min       181.483754
25%       365.734414
50%       421.884968
75%       481.792304
max       753.342620
Name: Conductivity, dtype: float64
```

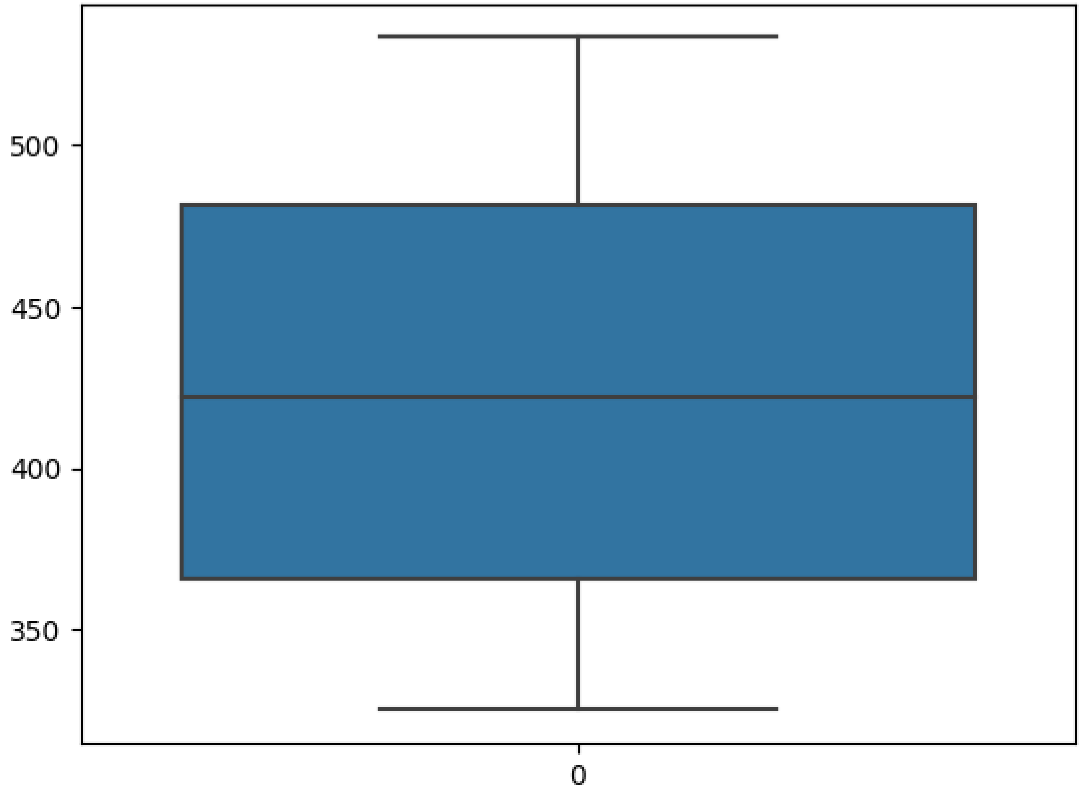


In [50]:

```
df["Conductivity"] = np.where(df["Conductivity"] <325.1171240676143, 325.1171240676143,d
df["Conductivity"] = np.where(df["Conductivity"]>533.2972414189196, 533.2972414189196,df
sns.boxplot(df["Conductivity"])
```

Out[50]:

<AxesSubplot: >



Organic_carbon

In [51]:

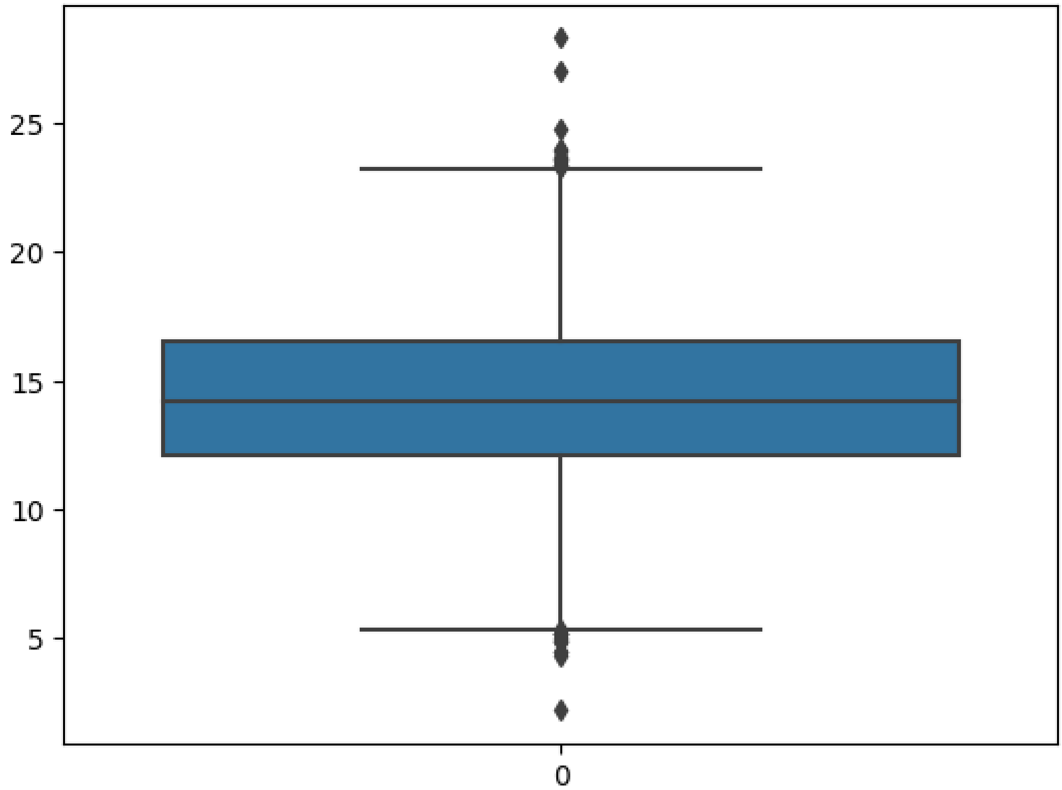
```
sns.boxplot(df["Organic_carbon"])

print(df["Organic_carbon"].quantile(0.10))
print(df["Organic_carbon"].quantile(0.90))
df["Organic_carbon"].describe()
```

```
10.123765383583503
18.50456708562831
```

Out[51]:

```
count    3276.000000
mean       14.284970
std         3.308162
min         2.200000
25%        12.065801
50%        14.218338
75%        16.557652
max        28.300000
Name: Organic_carbon, dtype: float64
```

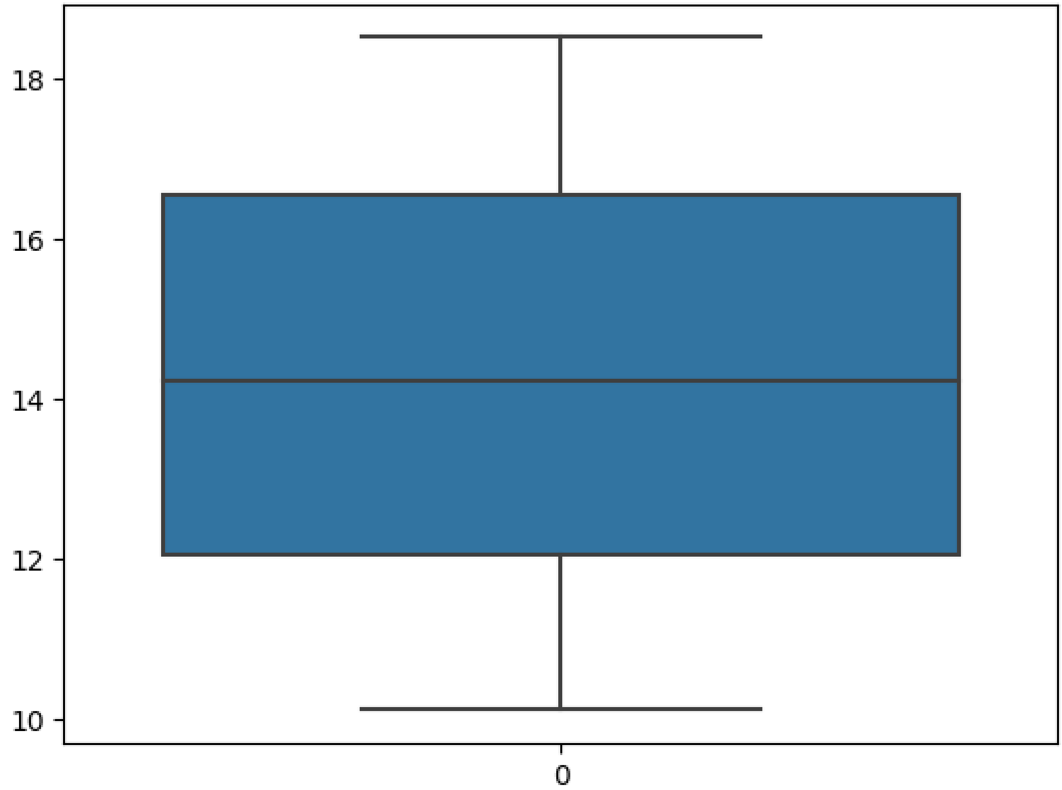


In [52]:

```
df["Organic_carbon"] = np.where(df["Organic_carbon"] < 10.123765383583503, 10.1237653835
df["Organic_carbon"] = np.where(df["Organic_carbon"]>18.50456708562831, 18.5045670856283
sns.boxplot(df["Organic_carbon"])
```

Out[52]:

```
<AxesSubplot: >
```



Trihalomethanes

In [53]:

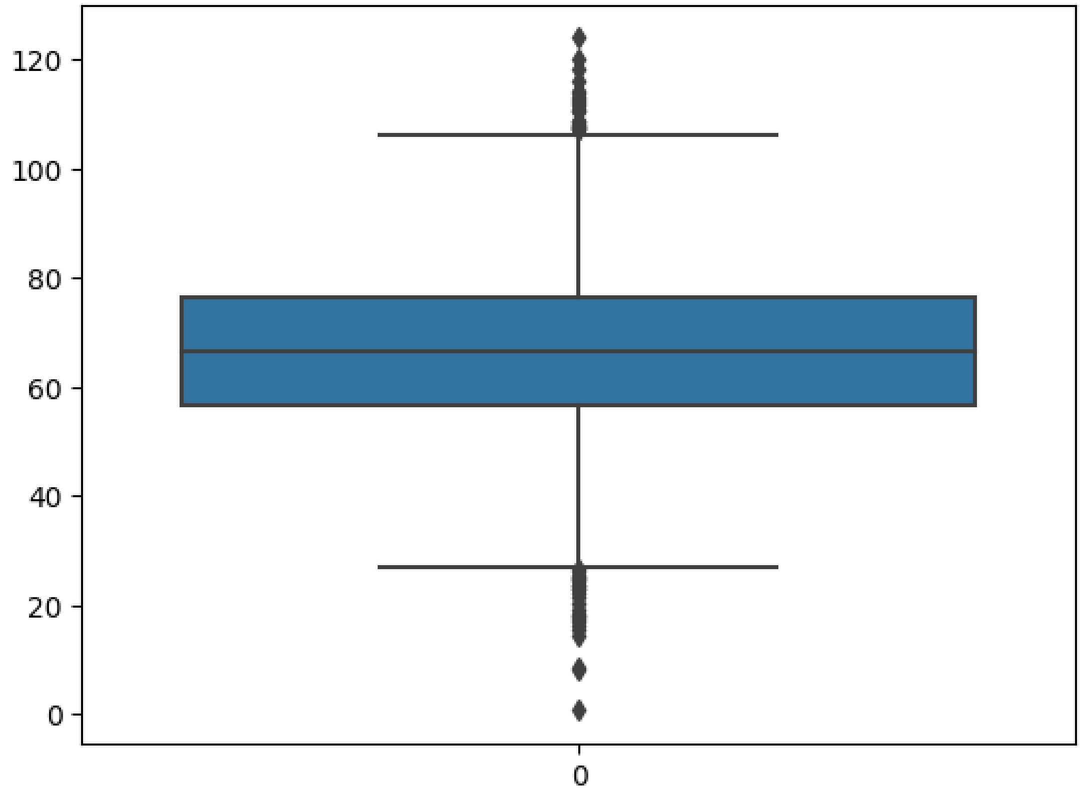
```
sns.boxplot(df["Trihalomethanes"])

print(df["Trihalomethanes"].quantile(0.10))
print(df["Trihalomethanes"].quantile(0.90))
df["Trihalomethanes"].describe()
```

```
46.209472917430155
85.90009466879661
```

Out[53]:

```
count    3276.000000
mean       66.396293
std        15.769881
min         0.738000
25%        56.647656
50%        66.396293
75%        76.666609
max       124.000000
Name: Trihalomethanes, dtype: float64
```

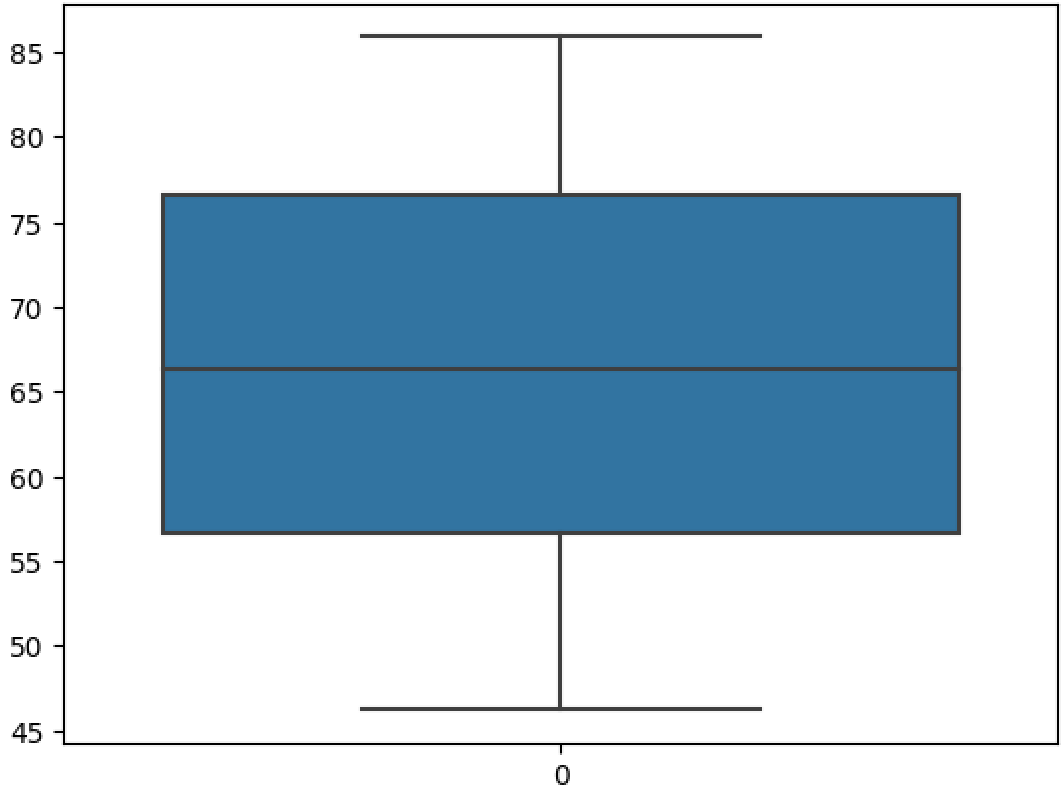


In [54]:

```
df["Trihalomethanes"] = np.where(df["Trihalomethanes"] < 46.209472917430155, 46.20947291
df["Trihalomethanes"] = np.where(df["Trihalomethanes"]>85.90009466879661, 85.90009466879
sns.boxplot(df["Trihalomethanes"])
```

Out[54]:

```
<AxesSubplot: >
```



Turbidity

In [55]:

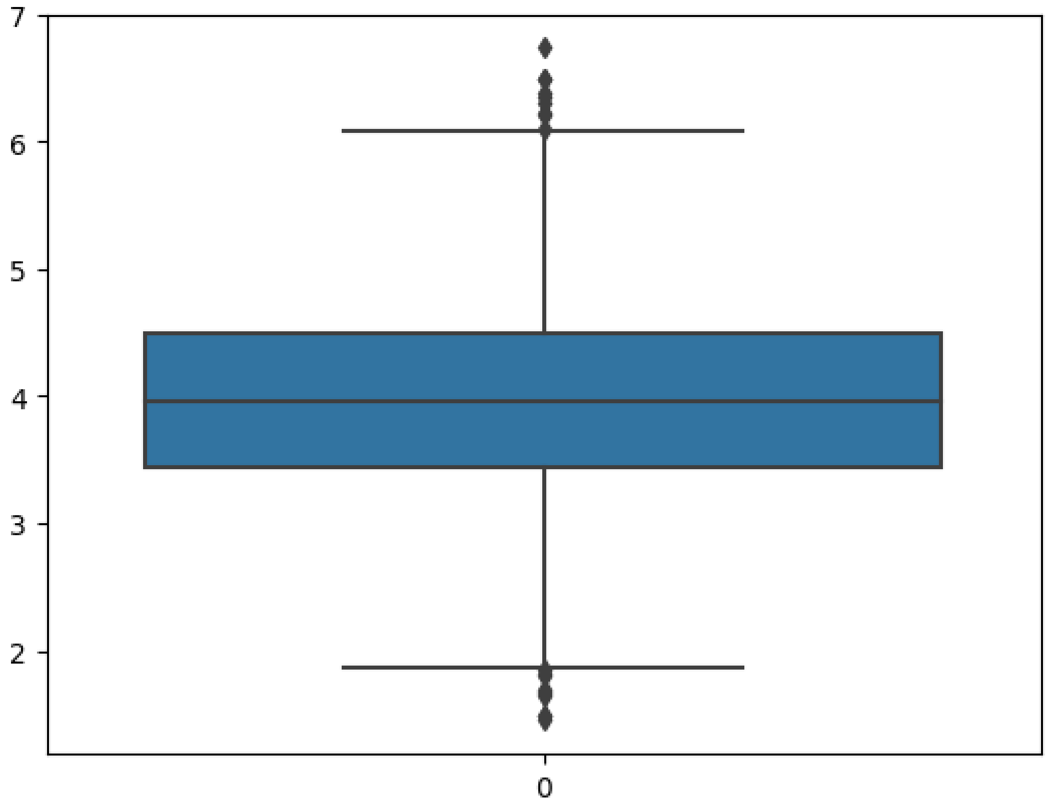
```
sns.boxplot(df["Turbidity"])

print(df["Turbidity"].quantile(0.10))
print(df["Turbidity"].quantile(0.90))
df["Turbidity"].describe()
```

```
2.9518028252699757
4.977140682806077
```

Out[55]:

```
count    3276.000000
mean        3.966786
std         0.780382
min         1.450000
25%         3.439711
50%         3.955028
75%         4.500320
max         6.739000
Name: Turbidity, dtype: float64
```

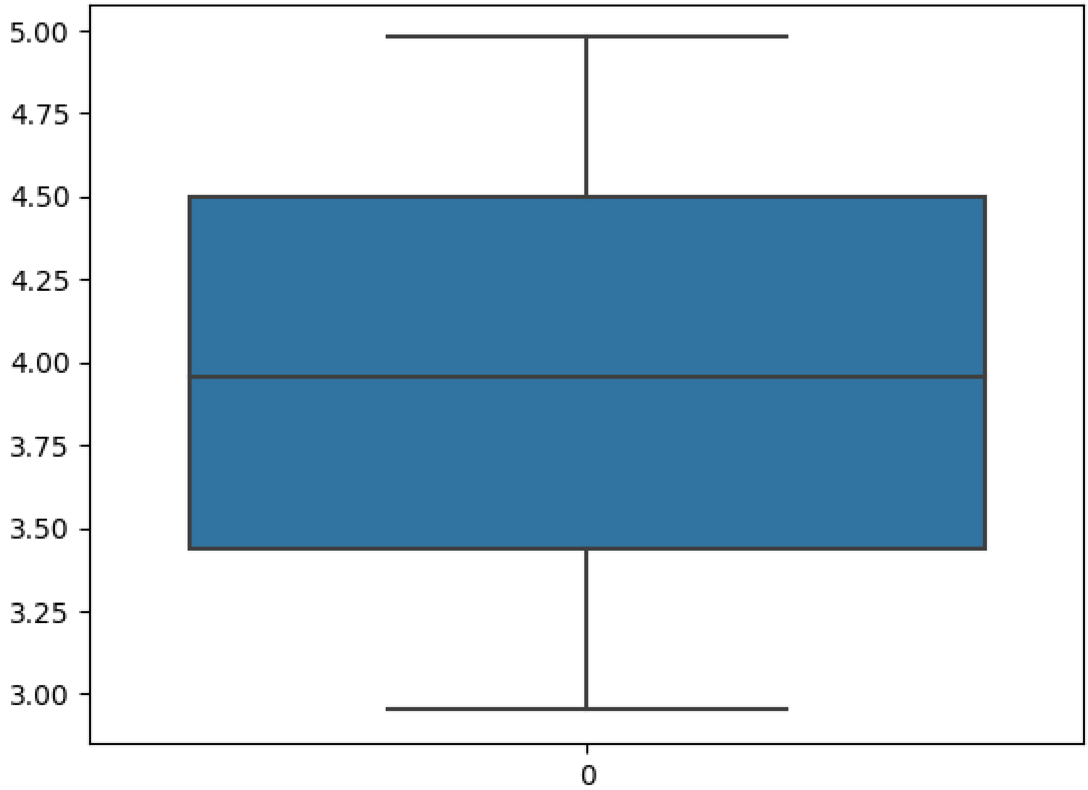


In [56]:

```
df["Turbidity"] = np.where(df["Turbidity"] < 2.9518028252699757, 2.9518028252699757,df["
df["Turbidity"] = np.where(df["Turbidity"]>4.977140682806077, 4.977140682806077,df["Turb
sns.boxplot(df["Turbidity"])
```

Out[56]:

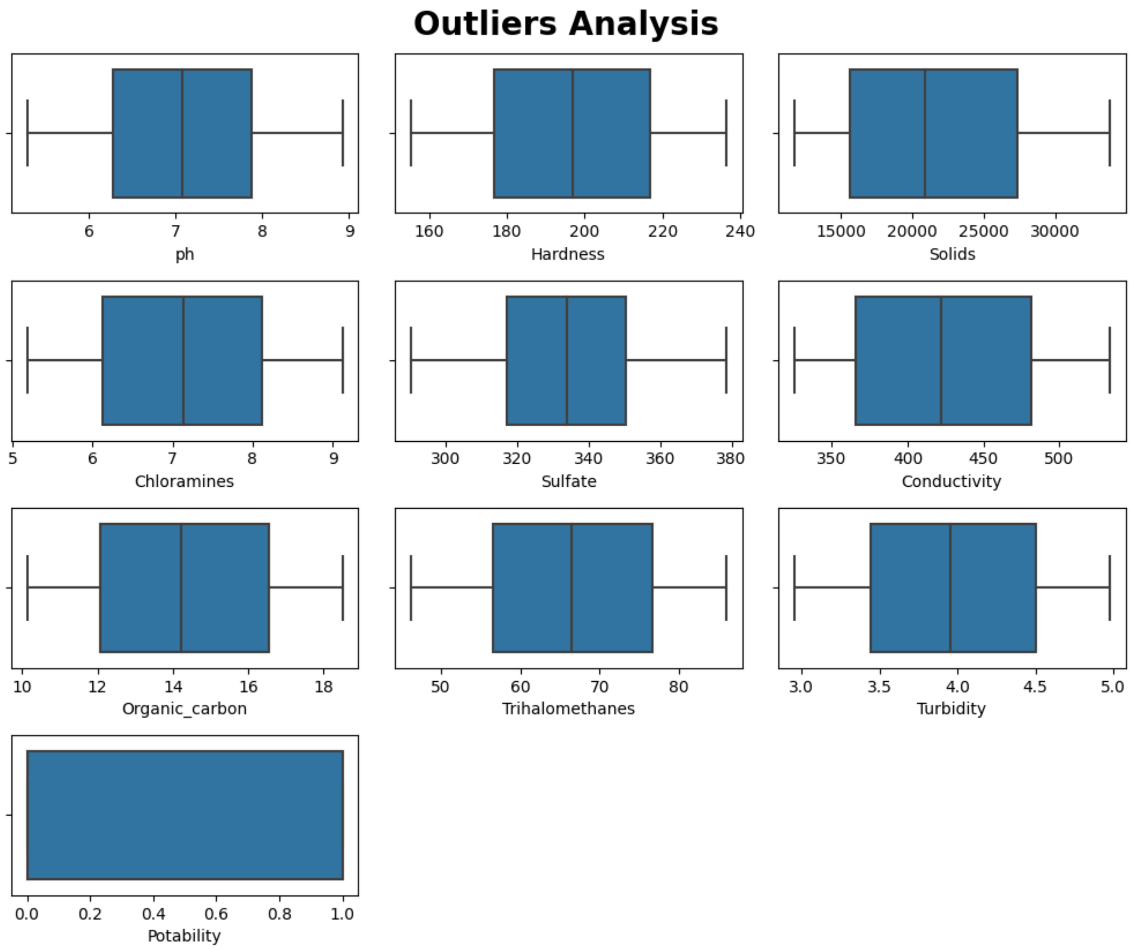
```
<AxesSubplot: >
```



Handled Outliers

In [57]:

```
plt.figure(figsize=(10,10))
plt.suptitle("Outliers Analysis", fontsize=20, fontweight="bold")
for i in range(0,len(df.columns)):
    plt.subplot(5,3,i+1)
    sns.boxplot(x=df[df.columns[i]])
    plt.xlabel(df.columns[i])
    plt.tight_layout()
```



Feature Selection

FILTER METHOD

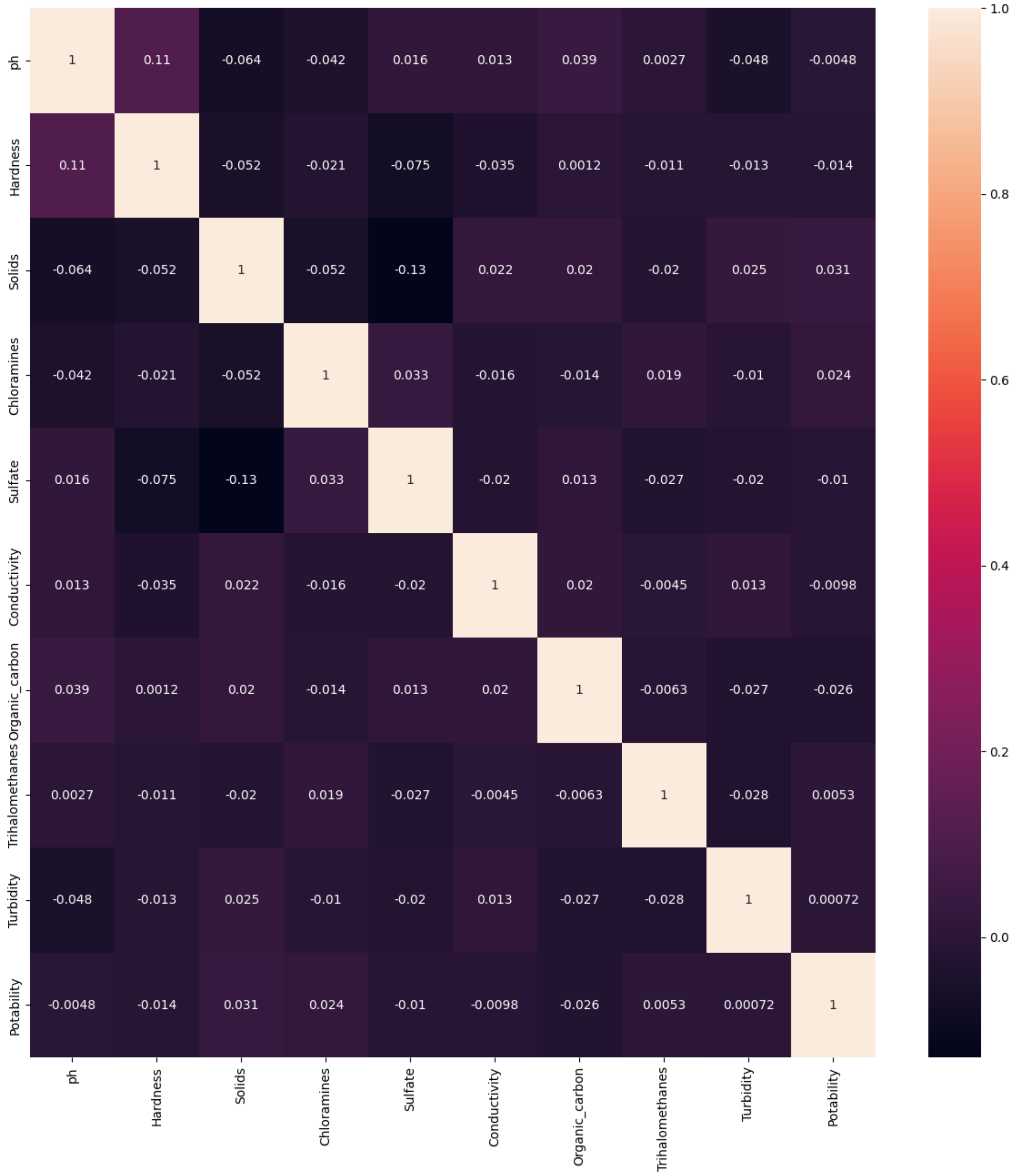
1. Pearson's correlation

In [58]:

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

Out[58]:

<AxesSubplot: >



fisher's score

In [59]:

```
from skfeature.function.similarity_based import fisher_score
```

In [60]:

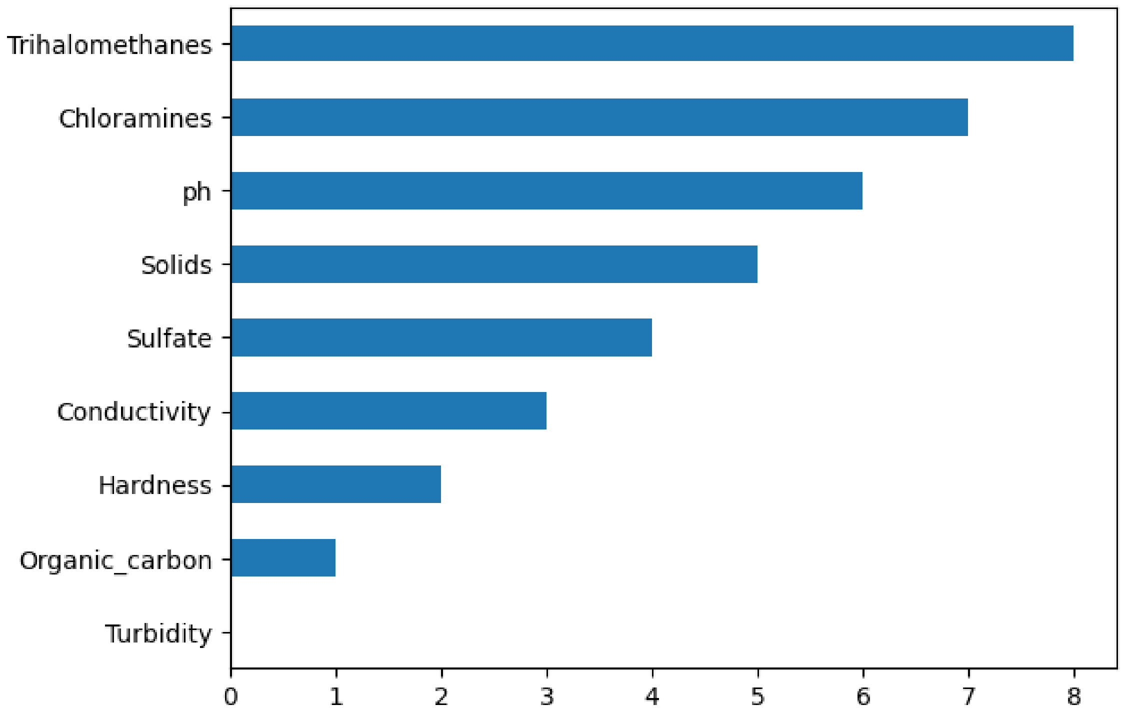
```
x = df.drop("Potability", axis=1)
y = df["Potability"]
```

In [61]:

```
fisher_rank = fisher_score.fisher_score(x.to_numpy(),y)
s1 = pd.Series(fisher_rank, index= x.columns)
s1.sort_values().plot(kind="barh")
```

Out[61]:

```
<AxesSubplot: >
```



Variance threshold method

In [62]:

```
from sklearn.feature_selection import VarianceThreshold
```

In [63]:

```
var_th = VarianceThreshold(threshold = 0.3)
var_th.fit_transform(df)
arr = var_th.get_support()
np.where(arr == False)
df.columns[np.where(arr == False)]
```

Out[63]:

```
Index(['Potability'], dtype='object')
```

variance inflationn factor

In [64]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [65]:

```
x.values
```

Out[65]:

```
array([[7.08079450e+00, 2.04890455e+02, 2.07913190e+04, ...,
        1.03797831e+01, 8.59000947e+01, 2.96313538e+00],
       [5.28219449e+00, 1.55223964e+02, 1.86300579e+04, ...,
        1.51800131e+01, 5.63290763e+01, 4.50065627e+00],
       [8.09912419e+00, 2.24236259e+02, 1.99095417e+04, ...,
        1.68686369e+01, 6.64200925e+01, 3.05593375e+00],
       ...,
       [8.92504688e+00, 1.75762646e+02, 3.31555782e+04, ...,
        1.10390697e+01, 6.98454003e+01, 3.29887550e+00],
       [5.28219449e+00, 2.30603758e+02, 1.19838694e+04, ...,
        1.11689462e+01, 7.74882131e+01, 4.70865847e+00],
       [7.87467136e+00, 1.95102299e+02, 1.74041771e+04, ...,
        1.61403676e+01, 7.86984463e+01, 2.95180283e+00]])
```

In [66]:

```
vif_lst = []
for i in range(x.shape[1]):
    vif = variance_inflation_factor(x.values,i)
    vif_lst.append(vif)
vif_lst
```

Out[66]:

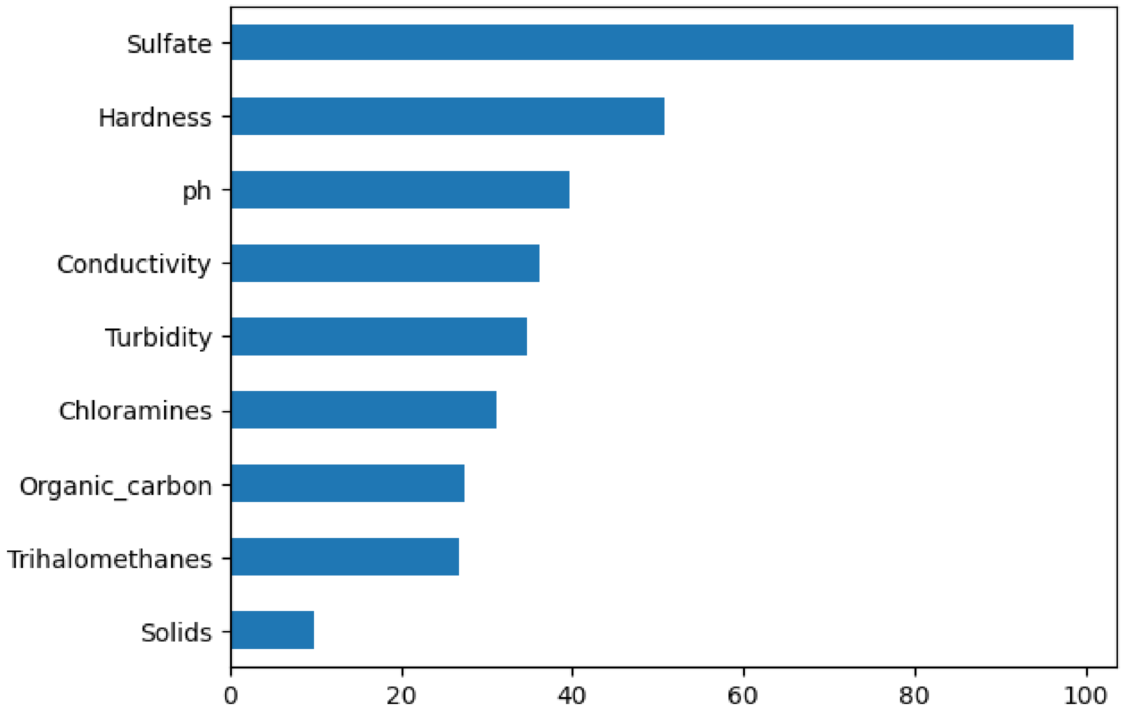
```
[39.60268140646638,
 50.81765606290291,
 9.854114392291905,
 31.12985728335243,
 98.62440998694287,
 36.253227605395594,
 27.448437065991804,
 26.83919183339501,
 34.67773797589433]
```

In [67]:

```
s1 = pd.Series(vif_lst, index= x.columns)
s1.sort_values()
s1.sort_values().plot(kind= "barh")
```

Out[67]:

```
<AxesSubplot: >
```



Information Gain

In [68]:

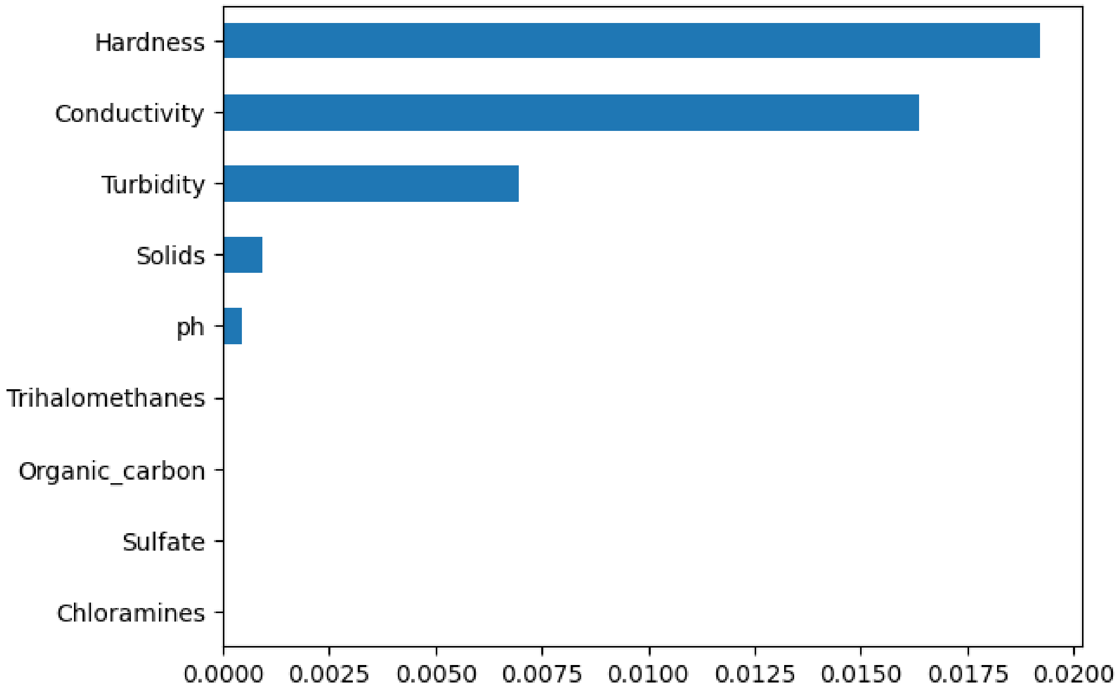
```
from sklearn.feature_selection import mutual_info_regression
```

In [69]:

```
a1=mutual_info_regression(x,y)
s2 = pd.Series(a1, index= x.columns)
s2.sort_values().plot(kind="barh")
```

Out[69]:

```
<AxesSubplot: >
```



sulphate and Trihalomethanes will be dropped as these two features don't contribute in our target column

In [70]:

```
df_backup = df
df_backup.head(2)
```

Out[70]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbo
0	7.080795	204.890455	20791.318981	7.300212	368.516441	533.297241	10.37978
1	5.282194	155.223964	18630.057858	6.635246	333.775777	533.297241	15.1800

In [71]:

```
df_new = df.drop(["Sulfate","Trihalomethanes"],axis=1)
```

In [72]:

```
df_new.head(2)
```

Out[72]:

	ph	Hardness	Solids	Chloramines	Conductivity	Organic_carbon	Turbidity
0	7.080795	204.890455	20791.318981	7.300212	533.297241	10.379783	2.963135
1	5.282194	155.223964	18630.057858	6.635246	533.297241	15.180013	4.500656

WRAPPER METHOD

Forward Elimination

In [73]:

```
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.neighbors import KNeighborsRegressor
import time
```

In [74]:

```
x1 = df.drop("Potability", axis=1)
y1 = df["Potability"]
```

In [75]:

```
strt_tm = time.time()
knn_model = KNeighborsRegressor()
sfs = SequentialFeatureSelector(knn_model, n_features_to_select=5, direction="forward",c
sfs.fit(x1,y1)
```

Out[75]:

```
▸ SequentialFeatureSelector
▸ estimator: KNeighborsRegressor
    ▸ KNeighborsRegressor
```

In [76]:

```
end_tm = time.time()
total = end_tm - strt_tm
print("total time taken: ", total)
a2 = sfs.get_support()
s3 = pd.Series(a2, x1.columns)
s3
```

```
total time taken:  4.894103765487671
```

Out[76]:

```
ph                   False
Hardness              True
Solids                True
Chloramines           True
Sulfate              False
Conductivity         False
Organic_carbon        True
Trihalomethanes      False
Turbidity             True
dtype: bool
```

Backward Elimination

In [77]:

```
strt_tm = time.time()
knn_model = KNeighborsRegressor()
sfs = SequentialFeatureSelector(knn_model, n_features_to_select=5, direction="backward",
sfs.fit(x1,y1)
end_tm = time.time()
total = end_tm - strt_tm
print("total time taken: ", total)
a2 = sfs.get_support()
s3 = pd.Series(a2, x1.columns)
s3
```

```
total time taken:  0.4389803409576416
```

Out[77]:

```
ph                   False
Hardness             False
Solids               False
Chloramines           True
Sulfate               True
Conductivity          True
Organic_carbon       False
Trihalomethanes       True
Turbidity             True
dtype: bool
```

In [78]:

```
df.drop("Sulfate",axis=1, inplace=True)
```

In [79]:

```
df.head()
```

Out[79]:

	ph	Hardness	Solids	Chloramines	Conductivity	Organic_carbon	Trihalom
0	7.080795	204.890455	20791.318981	7.300212	533.297241	10.379783	85
1	5.282194	155.223964	18630.057858	6.635246	533.297241	15.180013	56
2	8.099124	224.236259	19909.541732	9.122578	418.606213	16.868637	66
3	8.316766	214.373394	22018.417441	8.059332	363.266516	18.436524	85
4	8.925047	181.101509	17978.986339	6.546600	398.410813	11.558279	46

MOdel Evaluation

with df >> Sulfhate column in dropped

In [94]:

```
df.columns
```

Out[94]:

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

In [82]:

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

In [83]:

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

In [84]:

```
x_train.shape , x_test.shape
```

Out[84]:

```
((2620, 8), (656, 8))
```

In [85]:

```
y_train.shape , y_test.shape
```

Out[85]:

```
((2620,), (656,))
```

Logistic Regression

In [86]:

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

Out[86]:

```
▾ LogisticRegression
LogisticRegression()
```

testing

In [87]:

```
y_pred = lg_model.predict(x_test)
acc_score = accuracy_score(y_test,y_pred)
print("Test_LG_Acc: ",acc_score)

con_matrix = confusion_matrix(y_test,y_pred)
print("Confusion_Matrix:\n "  ,con_matrix)

clf_report = classification_report(y_test,y_pred)
print("Classification_report: \n" , clf_report )
```

```
Test_LG_Acc:  0.6280487804878049
Confusion_Matrix:
  [[412   0]
 [244   0]]
Classification_report: 
               precision    recall  f1-score   support

           0       0.63      1.00      0.77       412
           1       0.00      0.00      0.00       244

    accuracy                           0.63       656
   macro avg       0.31      0.50      0.39       656
weighted avg       0.39      0.63      0.48       656
```

Training

In [88]:

```
y_pred_train = lg_model.predict(x_train)
acc_score = accuracy_score(y_train,y_pred_train)
print("Train_LG_Acc: ",acc_score)

con_matrix = confusion_matrix(y_train,y_pred_train)
print("Confusion_Matrix:\n "  ,con_matrix)

clf_report = classification_report(y_train,y_pred_train)
print("Classification_report: \n" , clf_report )
```

```
Train_LG_Acc:  0.6053435114503817
Confusion_Matrix:
  [[1586    0]
 [1034    0]]
Classification_report: 
               precision    recall  f1-score   support

           0       0.61      1.00      0.75      1586
           1       0.00      0.00      0.00      1034

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

Random Forest

In [91]:

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

Out[91]:

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [92]:

```
y_pred_test = rf_model.predict(x_test)
conf_matrix = confusion_matrix(y_test,y_pred_test)
print("Confusion Matrix: \n",conf_matrix)

Acc = accuracy_score(y_test,y_pred_test)
print("Accuracy: ",Acc)

Clss_report = classification_report(y_test,y_pred_test)
print("Classification Report: \n", Clss_report)
```

```
Confusion Matrix: 
 [[357  55]
 [180  64]]
Accuracy:  0.6417682926829268
Classification Report: 
               precision    recall  f1-score   support

           0       0.66      0.87      0.75       412
           1       0.54      0.26      0.35       244

    accuracy                           0.64       656
   macro avg       0.60      0.56      0.55       656
weighted avg       0.62      0.64      0.60       656
```

In [93]:

```
y_pred_train = rf_model.predict(x_train)
conf_matrix = confusion_matrix(y_train,y_pred_train)
print("Confusion Matrix: \n",conf_matrix)

Acc = accuracy_score(y_train,y_pred_train)
print("Accuracy: ",Acc)

Clss_report = classification_report(y_train,y_pred_train)
print("Classification Report: \n", Clss_report)
```

```
Confusion Matrix: 
 [[1586    0]
 [   0 1034]]
Accuracy:  1.0
Classification Report: 
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      1586
           1       1.00      1.00      1.00      1034

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

with df_new >> after feature selection sulfate,Trihalomethanes was dropped

In [97]:

```
df_new.columns
```

Out[97]:

```
Index(['ph', 'Hardness', 'Solids', 'Chloramines', 'Conductivity',
       'Organic_carbon', 'Turbidity', 'Potability'],
      dtype='object')
```

In [98]:

```
x1 = df_new.drop("Potability",axis=1)
y1 = df_new["Potability"]
```

In [99]:

```
x_train,x_test,y_train,y_test = train_test_split(x1,y1,test_size=0.2,random_state=42)
```

Logistic Regression

In [100]:

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

Out[100]:

```
▾ LogisticRegression
LogisticRegression()
```

In [101]:

```
y_pred = lg_model.predict(x_test)
acc_score = accuracy_score(y_test,y_pred)
print("Test_LG_Acc: ",acc_score)

con_matrix = confusion_matrix(y_test,y_pred)
print("Confusion_Matrix:\n "  ,con_matrix)

clf_report = classification_report(y_test,y_pred)
print("Classification_report: \n" , clf_report )
```

```
Test_LG_Acc:  0.6280487804878049
Confusion_Matrix:
  [[412   0]
 [244   0]]
Classification_report:
               precision    recall  f1-score   support

           0       0.63      1.00      0.77       412
           1       0.00      0.00      0.00       244

    accuracy                           0.63       656
   macro avg       0.31      0.50      0.39       656
weighted avg       0.39      0.63      0.48       656
```

In [102]:

```
y_pred_train = lg_model.predict(x_train)
acc_score = accuracy_score(y_train,y_pred_train)
print("Train_LG_Acc: ",acc_score)

con_matrix = confusion_matrix(y_train,y_pred_train)
print("Confusion_Matrix:\n "  ,con_matrix)

clf_report = classification_report(y_train,y_pred_train)
print("Classification_report: \n" , clf_report )
```

```
Train_LG_Acc:  0.6053435114503817
Confusion_Matrix:
  [[1586    0]
 [1034    0]]
Classification_report:
               precision    recall  f1-score   support

           0       0.61      1.00      0.75      1586
           1       0.00      0.00      0.00      1034

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

df_backup >> all the features are present

In [103]:

```
df_backup.columns
```

Out[103]:

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

In [104]:

```
x2 = df_backup.drop("Potability",axis=1)
y2 = df_backup["Potability"]
```

In [109]:

```
x2_train,x2_test,y2_train,y2_test = train_test_split(x2,y2,test_size=0.2,random_state=42
```

In [110]:

```
lg_model = LogisticRegression()
lg_model.fit(x2_train,y2_train)
```

Out[110]:

```
▾ LogisticRegression
LogisticRegression()
```

In [112]:

```
y2_pred = lg_model.predict(x2_test)
acc_score = accuracy_score(y2_test,y2_pred)
print("Test_LG_Acc: ",acc_score)

con_matrix = confusion_matrix(y2_test,y2_pred)
print("Confusion_Matrix:\n "  ,con_matrix)

clf_report = classification_report(y2_test,y2_pred)
print("Classification_report: \n" , clf_report )
```

```
Test_LG_Acc:  0.6280487804878049
Confusion_Matrix:
  [[412   0]
 [244   0]]
Classification_report:
               precision    recall  f1-score   support

           0       0.63      1.00      0.77       412
           1       0.00      0.00      0.00       244

    accuracy                           0.63       656
   macro avg       0.31      0.50      0.39       656
weighted avg       0.39      0.63      0.48       656
```

In [113]:

```
y2_pred_train = lg_model.predict(x2_train)
acc_score = accuracy_score(y2_train,y2_pred_train)
print("Train_LG_Acc: ",acc_score)

con_matrix = confusion_matrix(y2_train,y2_pred_train)
print("Confusion_Matrix:\n "  ,con_matrix)

clf_report = classification_report(y2_train,y2_pred_train)
print("Classification_report: \n" , clf_report )
```

```
Train_LG_Acc:  0.6053435114503817
Confusion_Matrix:
  [[1586    0]
 [1034    0]]
Classification_report:
               precision    recall  f1-score   support

           0       0.61      1.00      0.75      1586
           1       0.00      0.00      0.00      1034

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

Random Forest

In [119]:

```
rf_model = RandomForestClassifier()
rf_model.fit(x2_train,y2_train)
```

Out[119]:

```
▾ RandomForestClassifier

RandomForestClassifier()
```

In [120]:

```
y2_pred_test = rf_model.predict(x2_test)
conf_matrix = confusion_matrix(y2_test,y2_pred_test)
print("Confusion Matrix: \n",conf_matrix)

Acc = accuracy_score(y2_test,y2_pred_test)
print("Accuracy: ",Acc)

Clss_report = classification_report(y2_test,y2_pred_test)
print("Classification Report: \n", Clss_report)
```

```
Confusion Matrix:
 [[350  62]
 [184  60]]
Accuracy:  0.625
Classification Report:
               precision    recall  f1-score   support

           0       0.66      0.85      0.74       412
           1       0.49      0.25      0.33       244

    accuracy                           0.62       656
   macro avg       0.57      0.55      0.53       656
weighted avg       0.59      0.62      0.59       656
```

In [121]:

```
y2_pred_train = rf_model.predict(x2_train)
conf_matrix = confusion_matrix(y2_train,y2_pred_train)
print("Confusion Matrix: \n",conf_matrix)

Acc = accuracy_score(y2_train,y2_pred_train)
print("Accuracy: ",Acc)

Clss_report = classification_report(y2_train,y2_pred_train)
print("Classification Report: \n", Clss_report)
```

```
Confusion Matrix:
 [[1586    0]
 [   0 1034]]
Accuracy:  1.0
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      1586
           1       1.00      1.00      1.00      1034

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

hyperparameter tunning

In [124]:

```
hyperparameters = {"n_estimators": np.arange(10,100),
                  "criterion": ["gini","entropy"],
                  "max_depth": np.arange(3,8),
                  "min_samples_split": np.arange(2,20),
                  "min_samples_leaf": np.arange(2,15),
                  "random_state": [11],
                  "oob_score": [True],
                  "max_features": ["auto"]}
rscv_rf_model = RandomizedSearchCV(rf_model,hyperparameters,cv=7)
rscv_rf_model.fit(x2_train,y2_train)
```

Out[124]:

```
▸ RandomizedSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [125]:

```
y2_pred_test = rscv_rf_model.predict(x2_test)
conf_matrix = confusion_matrix(y2_test,y2_pred_test)
print("Confusion Matrix: \n",conf_matrix)

Test_RF_Acc = accuracy_score(y2_test,y2_pred_test)
print("Accuracy: ",Test_RF_Acc)

Clss_report = classification_report(y2_test,y2_pred_test)
print("Classification Report: \n", Clss_report)
```

```
Confusion Matrix: 
 [[401  11]
 [224  20]]
Accuracy:  0.6417682926829268
Classification Report: 
               precision    recall  f1-score   support

           0       0.64      0.97      0.77       412
           1       0.65      0.08      0.15       244

    accuracy                           0.64       656
   macro avg       0.64      0.53      0.46       656
weighted avg       0.64      0.64      0.54       656
```

In [126]:

```
y2_pred_train = rscv_rf_model.predict(x2_train)
conf_matrix = confusion_matrix(y2_train,y2_pred_train)
print("Confusion Matrix: \n",conf_matrix)

Train_RF_Acc = accuracy_score(y2_train,y2_pred_train)
print("Accuracy: ",Train_RF_Acc)

Clss_report = classification_report(y2_train,y2_pred_train)
print("Classification Report: \n", Clss_report)
```

```
Confusion Matrix: 
 [[1575   11]
 [ 847  187]]
Accuracy:  0.6725190839694657
Classification Report: 
               precision    recall  f1-score   support

           0       0.65      0.99      0.79      1586
           1       0.94      0.18      0.30      1034

    accuracy                           0.67      2620
   macro avg       0.80      0.59      0.54      2620
weighted avg       0.77      0.67      0.60      2620
```

finally got accuracy with orignal data tessting = 64 and trAining=67

In [ ]:

```

```