राजीव गाँधी पेट्रोलियम प्रौद्योगिकी संस्थान
(संसद के अधिनियम के अधीन स्थापित राष्ट्रीय महत्व का एक संस्थान)
Rajiv Gandhi Institute of Petroleum Technology
(An Institution of National Importance established under an Act of Parliament)
Jais, Amethi- 229304, UP, India. Website: www.rgipt.ac.in

BTP Report

on

Water Quality Prediction using Machine learning

Submitted by:

Animay Prakash

20CS3073

Submitted to:

Dr. Rahul Kumar

Assistant/Associate Professor

Department of Computer Science and Engineering

Abstract

Water quality refers to the suitability of water for different uses according to its physical, chemical, biological, and organoleptic (taste-related) properties. It is especially important to understand and measure water quality as it directly impacts human consumption and health, industrial and domestic use, and the natural environment. Water quality is measured using laboratory techniques or home kits. Laboratory testing measures multiple parameters and provides the most accurate results but takes the longest time. Home test kits, including test strips, provide rapid results but are less accurate So nowadays very important to control water pollution, as well as to alert users in case of poor quality detection. Motivated by these reasons, we choose this as our BTP. We take the advantages of machine learning algorithms to develop a model that is capable of predicting water quality. The method we propose is based on nine water parameters: **pH, Hardness, Solids, Chloramines, Sulfate, Conductivity, Organic carbon, Trihalomethanes, Turbidity**. The use of the **XGBoost Classifier** has proven to be important and effective in predicting the water quality index.

Keywords: water quality prediction, XGBoost Classifier , machine learning, organoleptic.

Table of Contents

1. Introduction

Water is a principal basic element on the earth. Living organisms, namely, humans, animals, vegetables and plants require water to survive. Without water, there would be no life on the Earth. According to the studies, approximately 66% of the Earth is made up of water with the availability of fresh or usable water being only 1%, while the rest of the water is saline or salt water. Due to the ever-increasing population of India, water resources are under pressure to provide basic functions to such a big population. To manage water resources, various water management systems have been invented recently. when it comes to water pollution water quality monitoring plays a major role. it contributes efficiently to the implementation of water resources protection plan envisioned for clean and pure water. water river quality is one of the main characteristics that needs full attention from environmental scholars. The quality of the water becomes a growing concern throughout the developing world.

The process of abstraction water for domestic use, agricultural production,

mining industrial production, power generation, and forestry practices can lead to deterioration in water quality and quantity that impact not only the aquatic ecosystem but also the availability of safe water for human consumption. Thus,

assessment of the quality of surface waters is important in hydro-environmental management and it is very significant in monitoring the concentration of pollutants in rivers. Nowadays, due to inadequate freshwater resources, people are extensively using groundwater for drinking, irrigation and industrial purposes. Generally ,Groundwater is considered to be safe and reliable source

of drinking water due to its natural, hidden existence and less vulnerable for contamination as compared with surface water.

Therefore, groundwater quality evaluation is very important based on properties such as physical, chemical and biological, with reference to naturally occurring quality, human health impacts, and proposed uses wherein it depends on the amount of rain, water harvesting system Thus, monitoring of

water quality is mandatory for the better management of accessible resources of water and to build up a proper solution for various environmental problems.

In this report, my goal is to suggest a new model for prediction water quality based on machine learning algorithms and with minimal parameters. In addition, the performance of the new model is compared with other models and evaluated according to their accuracies.

- **About Dataset:**

  The file contains water quality metrics for 3276 different water bodies.

1. **Ph value:** PH is an important parameter in evaluating the acid–base balance of water. It is also the indicator of acidic or alkaline condition of water According to WHO the maximum permissible limit of pH is from 6.5 to 8.5.

2. **Hardness:** Hardness is mainly caused by calcium and magnesium salts. These salts are dissolved from geologic deposits through which water travels. The length of time water is in contact with hardness producing material helps determine how much hardness there is in raw water. Hardness was originally defined as the capacity of water to precipitate soap caused by Calcium and Magnesium.

3. **Solids (Total dissolved solids - TDS):** Water has the ability to dissolve a wide range of inorganic and some organic minerals or salts such as potassium, calcium, sodium, bicarbonates, chlorides, magnesium, sulfates etc. This is the important parameter for the use of water. The water with high TDS value indicates that water is highly mineralized.

4. **Chloramines:** Chlorine and chloramine are the major disinfectants used in public water systems. Chloramines are most commonly formed when ammonia is added to chlorine to treat drinking water.

5. **Sulfates:** Sulfates are naturally occurring substances that are found in minerals, soil, and rocks. They are present in ambient air, groundwater, plants, and food. The principal commercial use of sulfate is in the chemical industry. It ranges from 3 to 30 mg/L in most freshwater supplies.

6. **Conductivity:** Pure water is not a good conductor of electric current rather a good insulator. Increase in ions concentration enhances the electrical conductivity of water. Generally, the amount of dissolved solids in water determines the electrical conductivity.

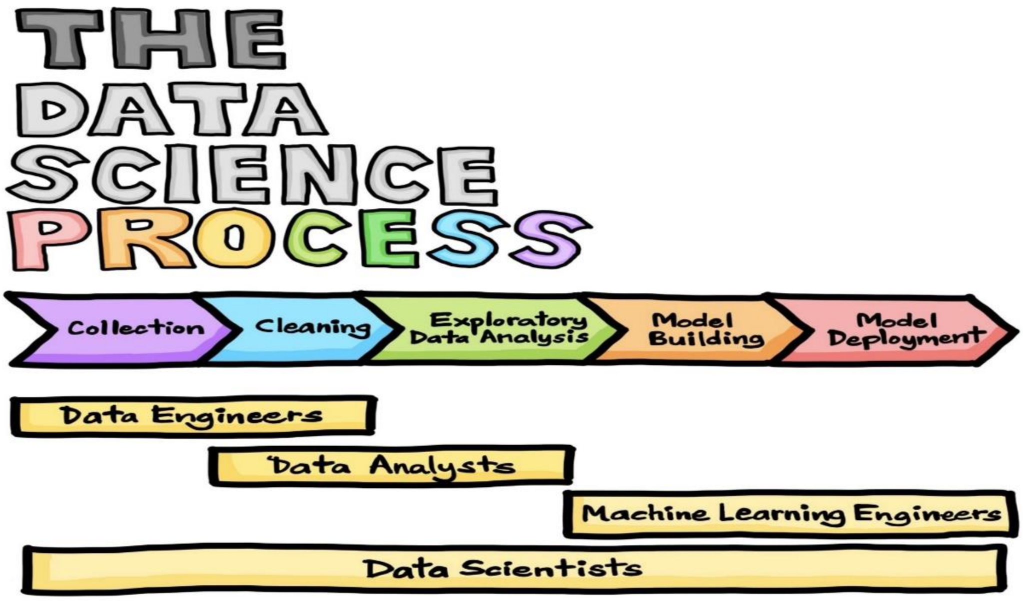
7. **Organic Carbon**: Total Organic Carbon (TOC) in source waters comes from decaying natural organic matter (NOM) as well as synthetic sources. it
measure of the total amount of carbon in organic compounds in pure water.

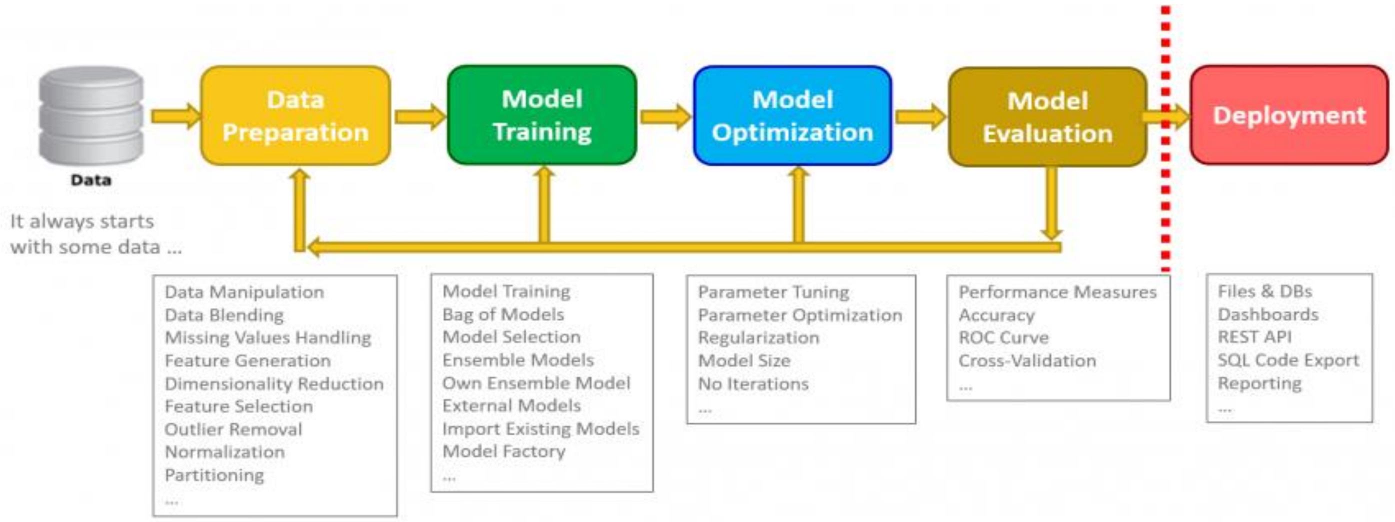
8. **Trihalomethanes:** THMs are chemicals which may be found in water treated with chlorine. The concentration of THMs in drinking water varies according to the level of organic material in the water, the amount of chlorine required to treat the water, and the temperature of the water that is being treated.

9. **Turbidity:** The turbidity of water depends on the quantity of solid matter present in the suspended state. It is a measure of light emitting properties of water.

10. **Potability**: Indicates if water is safe for drinking where 1 means Potable and 0 means Not potable.

2. Literature review/ Overview of Project





This Picture provides you a great overview of my Project it shows first of all we have to start with our dataset let me tell about my dataset in brief I got my water quality prediction data having 9 parameter from Kaggle after getting the raw data I have to prepare my data for training and for making our data fruitful for training we have to do data cleaning I perform missing values handling like how may null values are present in my data, getting my data shape & mean of a particular parameter etc. In performing missing values handling by removing null values and by filling mean in place of null values this things is called **Data imputations** in data manipulation part our data is well organised so we don't need to do data manipulation our data is in well csv format and is easily readable in data blending part we don't need to merge multiple files into a single file I have use only one file in csv format. So I don't need to do data blending too after doing all these I don't need to remove any of the feature we have 9 features so we don't remove our features and after performing all these let's move to **EDA – Exploratory Data Analysis** in EDA here we prepare the data by exploring the data and for dimensionality reduction we have to take use of heatmap here we are exploring it for reducing the dimension we need to check which feature is less important after checking data correlation's they aren't correlating even about 50% so for dimensionality reduction parameters have to correlating about 70% which is not in our case. Outlier removal means when you are too away from the mean we can check the outlier by using box plot

In our data we are having outlier in solids parameter so we have a choice to remove this outlier or not so I choose not to remove it because it might be possible that water is good or drinkable due to excess of solids.

Label encoding which I think not needed because it converts data to strings we don't need to convert potability which is 0 or 1 to drinkable or not   drinkable(ie true or false).our data dataset is well normalised so I don't need to do data normalisation. Our dataset is not imbalance we can check it by **sns.countplot**. after this we do more focus on EDA and plot various scatter plots.We do Partioning before Model Training.

For Model Training We perform 8 Algorithms and compare accuracies of all 8 algorithms are:

1. **Decision Tree**
2. **K-Nearest Neighbors(KNN)**
3. **Logistic Regression**
4. **Random Forest**
5. **XGBoost Classifier**
6. **Support Vector Machines**
7. **AdaBoost Classifier**
8. **Gaussian Naive Bayes**

After Model training we do model optimization of Decision tree and KNN in which KNN gives better results comparatively.

- **Some of the screen shots of my Code.**

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import plotly.express as px
        import seaborn as sns
```

```
In [2]: df=pd.read_csv(r'D:\Project\Water Quality Prediction using Machine Learning\Wate
        df.head()
```

Out[2]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Tril
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180013	
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.868637	
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	

- **Our dataset Fetching**

```
In [3]: print(df.columns)

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

- **Parameters.**

Exploratory Data Analysis

```
In [4]: df.shape

Out[4]: (3276, 10)
```

```
In [5]: df.isnull().sum()
Out[5]: ph                 491
        Hardness             0
        Solids               0
        Chloramines          0
        Sulfate            781
        Conductivity         0
        Organic_carbon       0
        Trihalomethanes    162
        Turbidity            0
        Potability           0
        dtype: int64
```

- **Null value count of a particular parameter**

```
In [6]: 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
```

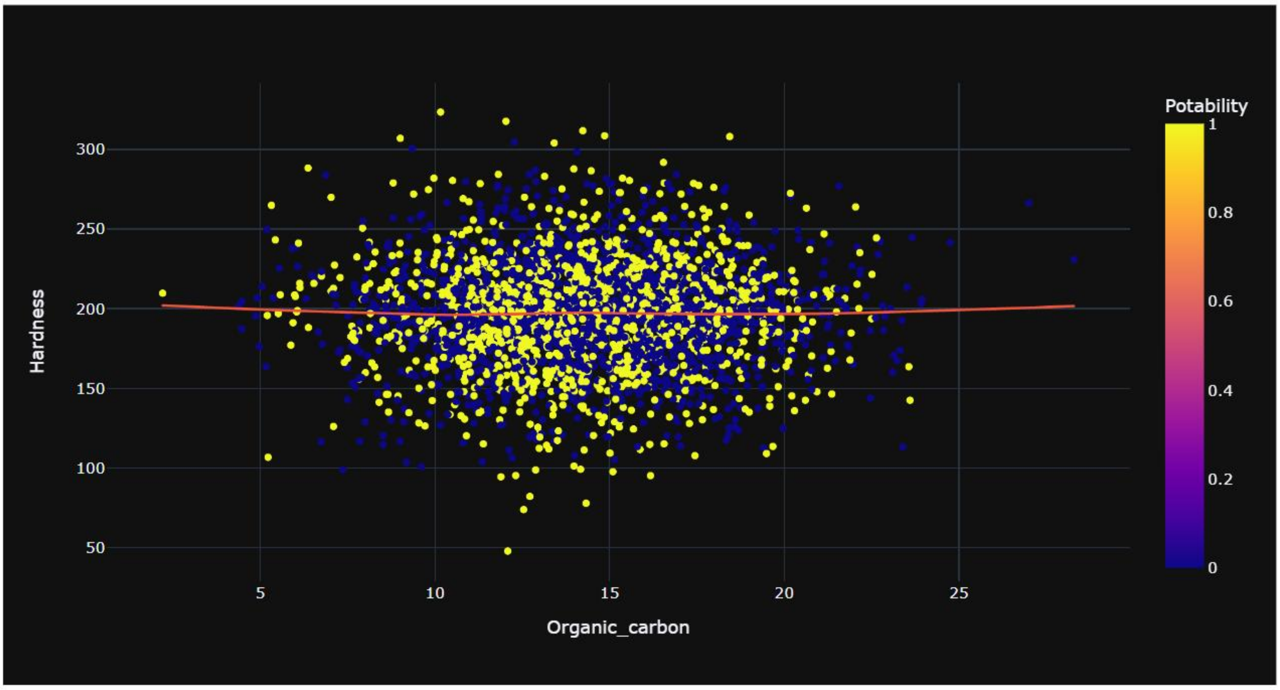
- **Getting Data Information.**

```
In [7]: df.describe()
```

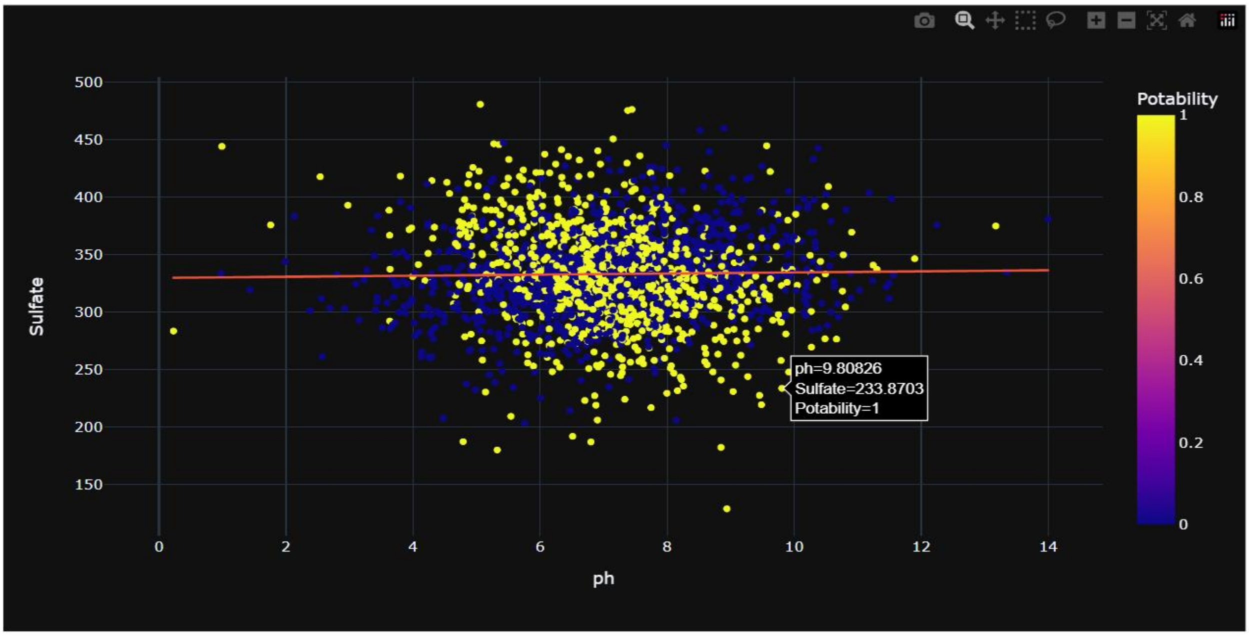
Out[7]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
count	2785.000000	3276.000000	3276.000000	3276.000000	2495.000000	3276.000000	3276.000000	3114.000000	3276.000000	3276.000000
mean	7.080795	196.369496	22014.092526	7.122277	333.775777	426.205111	14.284970	66.396293	3.966786	0.390110
std	1.594320	32.879761	8768.570828	1.583085	41.416840	80.824064	3.308162	16.175008	0.780382	0.487849
min	0.000000	47.432000	320.942611	0.352000	129.000000	181.483754	2.200000	0.738000	1.450000	0.000000
25%	6.093092	176.850538	15666.690297	6.127421	307.699498	365.734414	12.065801	55.844536	3.439711	0.000000
50%	7.036752	196.967627	20927.833607	7.130299	333.073546	421.884968	14.218338	66.622485	3.955028	0.000000
75%	8.062066	216.667456	27332.762127	8.114887	359.950170	481.792304	16.557652	77.337473	4.500320	1.000000
max	14.000000	323.124000	61227.196008	13.127000	481.030642	753.342620	28.300000	124.000000	6.739000	1.000000

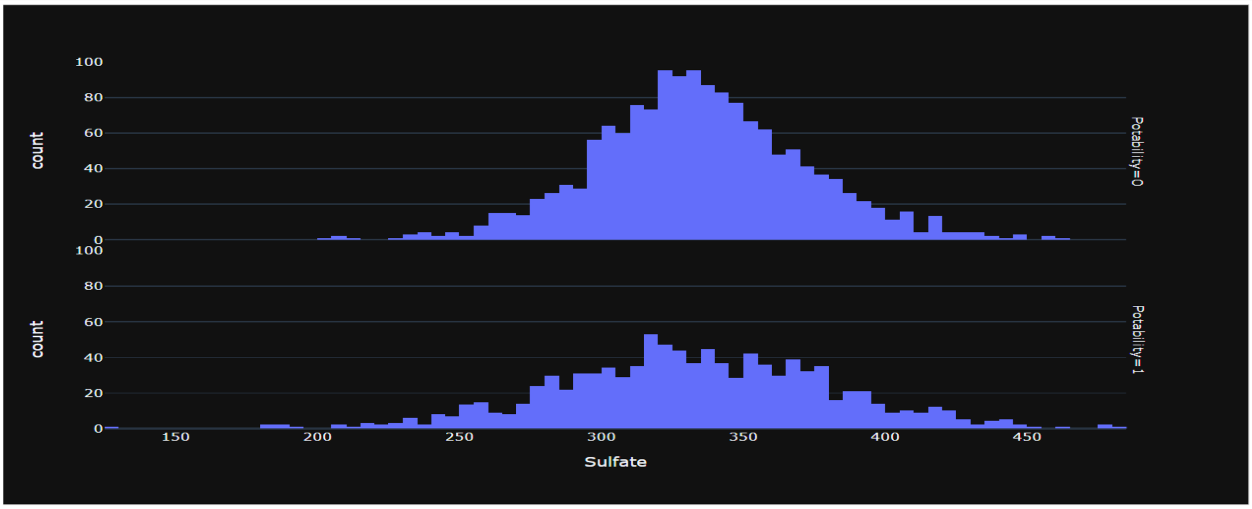
In [9]:
```
fig = px.scatter (df, x = "Organic_carbon", y = "Hardness", color = "Potability", template = "plotly_dark",  trendline="lowess")
fig.show ()
```



In [10]:
```
fig = px.scatter (df, x = "ph", y = "Sulfate", color = "Potability", template = "plotly_dark",  trendline="ols")
fig.show ()
```



In [11]:
```
fig = px.histogram (df, x = "Sulfate",  facet_row = "Potability",  template = 'plotly_dark')
fig.show()
```



In [12]:
```
df.fillna(df.mean(), inplace=True)
df.head()
```

Out[12]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	7.080795	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	0
1	3.716080	129.422921	18630.057858	6.635246	333.775777	592.885359	15.180013	56.329076	4.500656	0
2	8.099124	224.236259	19909.541732	9.275884	333.775777	418.606213	16.868637	66.420093	3.055934	0
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.341674	4.628771	0
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	0

- **Missing value handling**

In [13]:
```
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               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
```

In [15]:
```
df.isnull().sum()
```

Out[15]:
```
ph                 0
Hardness           0
Solids             0
Chloramines        0
Sulfate            0
Conductivity       0
Organic_carbon     0
Trihalomethanes    0
Turbidity          0
Potability         0
dtype: int64
```

In [16]:
```
df.describe()
```

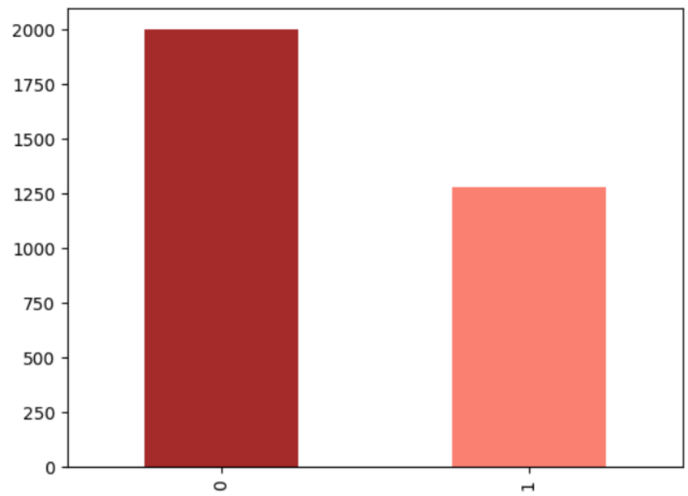
Out[16]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
count	3276.000000	3276.000000	3276.000000	3276.000000	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	14.284970	66.396293	3.966786	0.390110
std	1.469956	32.879761	8768.570828	1.583085	36.142612	80.824064	3.308162	15.769881	0.780382	0.487849
min	0.000000	47.432000	320.942611	0.352000	129.000000	181.483754	2.200000	0.738000	1.450000	0.000000
25%	6.277673	176.850538	15666.690297	6.127421	317.094638	365.734414	12.065801	56.647656	3.439711	0.000000
50%	7.080795	196.967627	20927.833607	7.130299	333.775777	421.884968	14.218338	66.396293	3.955028	0.000000
75%	7.870050	216.667456	27332.762127	8.114887	350.385756	481.792304	16.557652	76.666609	4.500320	1.000000
max	14.000000	323.124000	61227.196008	13.127000	481.030642	753.342620	28.300000	124.000000	6.739000	1.000000

In [17]:
```
df.Potability.value_counts()
```

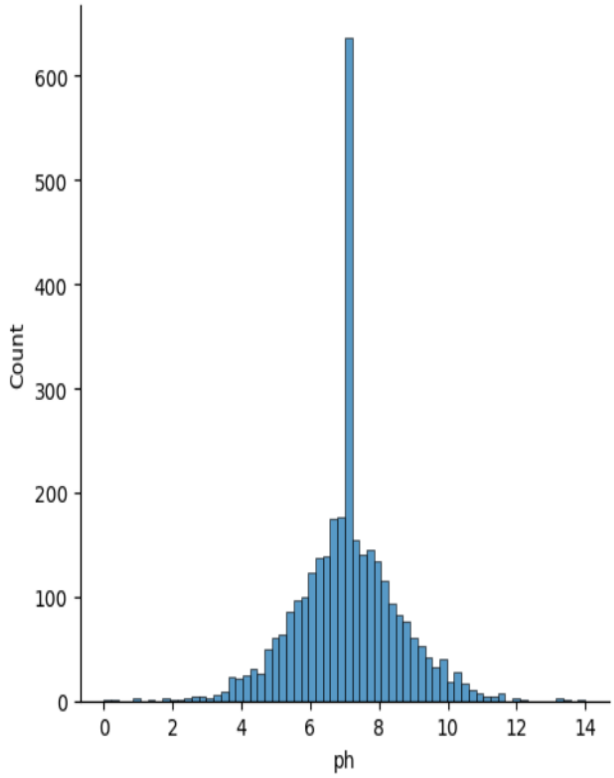
Out[17]:
```
0    1998
1    1278
Name: Potability, dtype: int64
```

In [18]:
```
df.Potability.value_counts().plot(kind="bar", color=["brown", "salmon"])
plt.show()
```

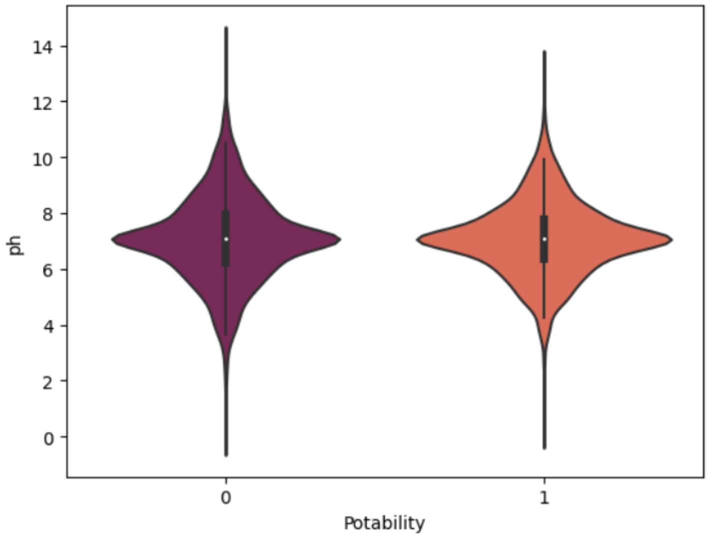


- **Showing the Potability to count to check whether our data is imbalance or not ie I Try to find good data or bad data.**

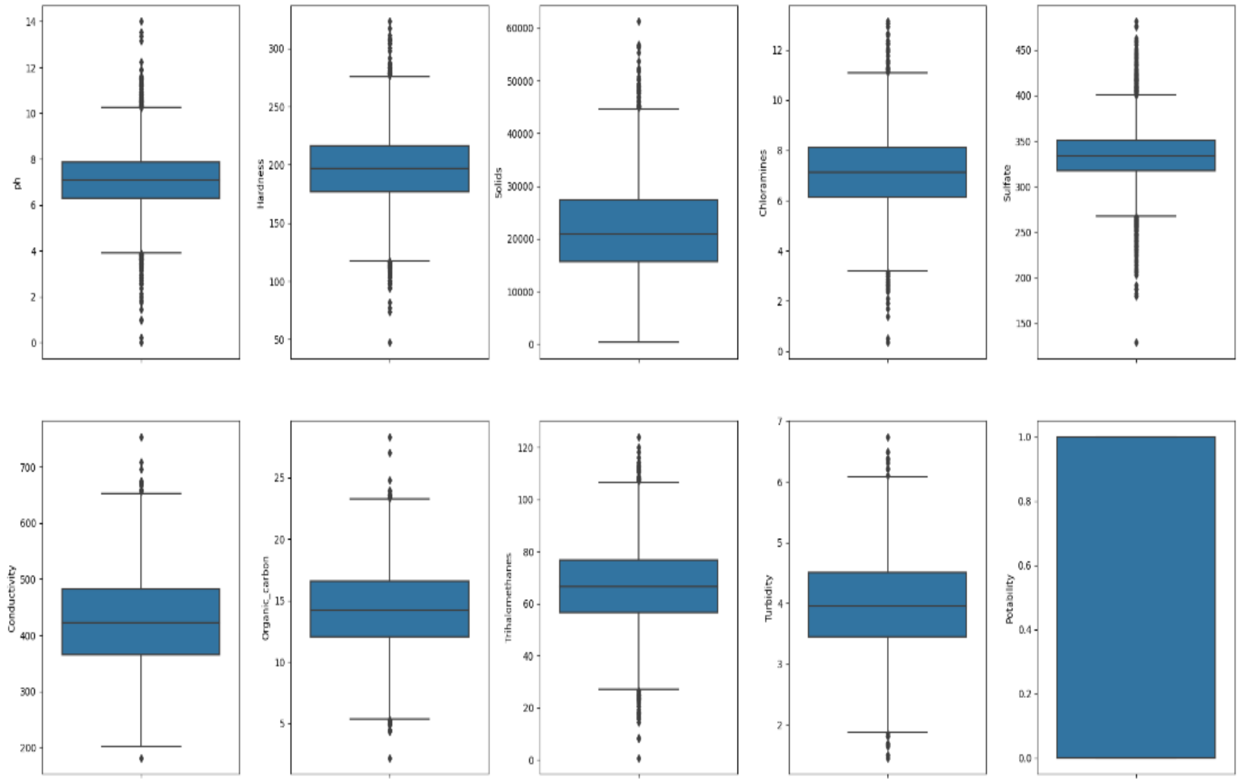
In [19]:
```
sns.displot(df['ph'])
plt.show()
```



In [20]:
```
sns.violinplot(x='Potability', y='ph', data=df, palette='rocket')
plt.show()
```

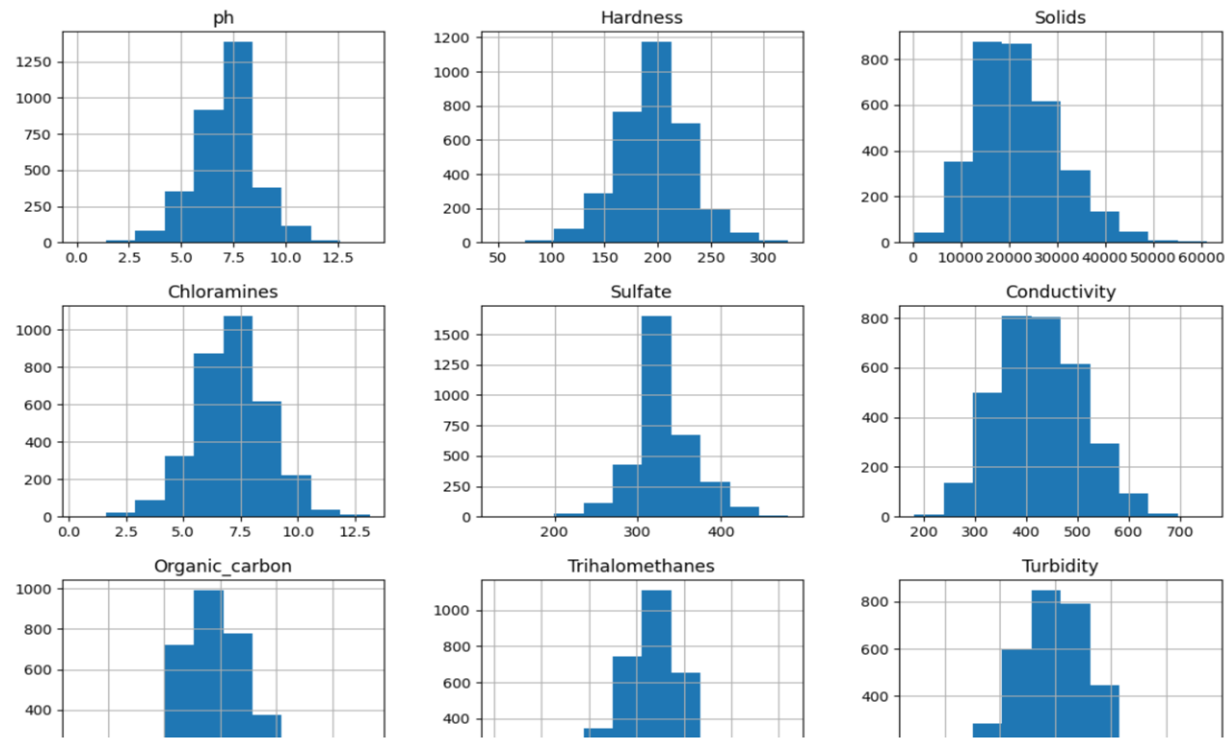


```
In [21]: # Visualizing dataset and also checking for outliers

fig, ax = plt.subplots(ncols = 5, nrows = 2, figsize = (20, 10))
index = 0
ax = ax.flatten()

for col, value in df.items():
    sns.boxplot(y=col, data=df, ax=ax[index])
    index += 1
plt.tight_layout(pad = 0.5, w_pad=0.7, h_pad=5.0)
plt.show()
```



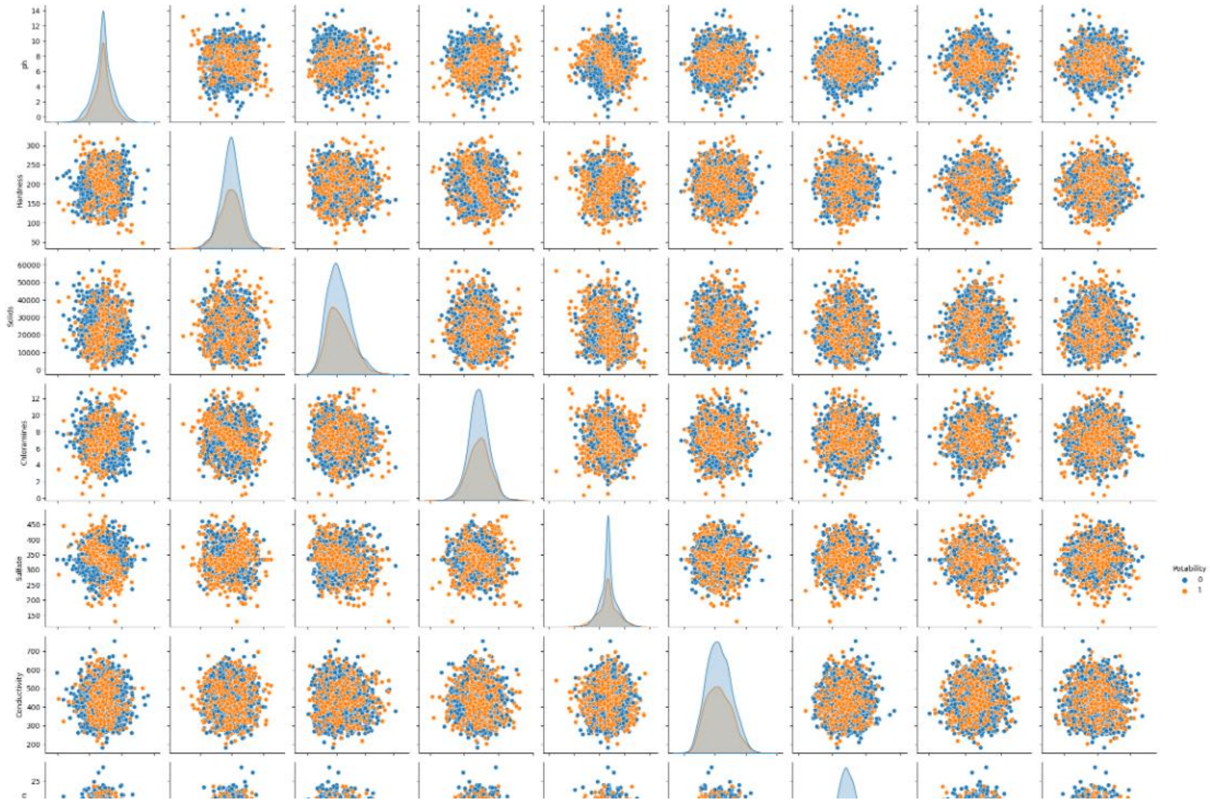
- **Checking outliers using box plot we can remove outliers but in this case I choose to keep it.**

```
In [22]: df.hist(figsize=(14,14))
         plt.show()
```



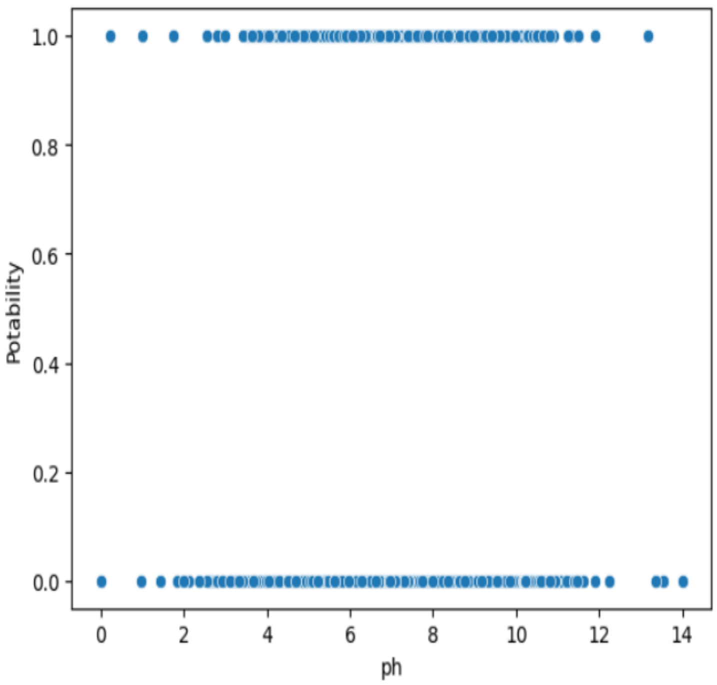
- **This hist shows that our data is Normalised**

In [23]:
```
sns.pairplot(df,hue='Potability')
plt.show()
```

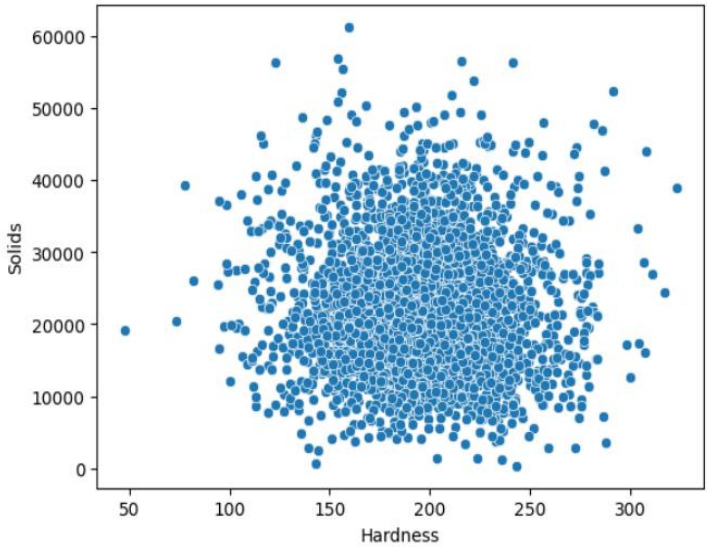


- **Pair Plot**

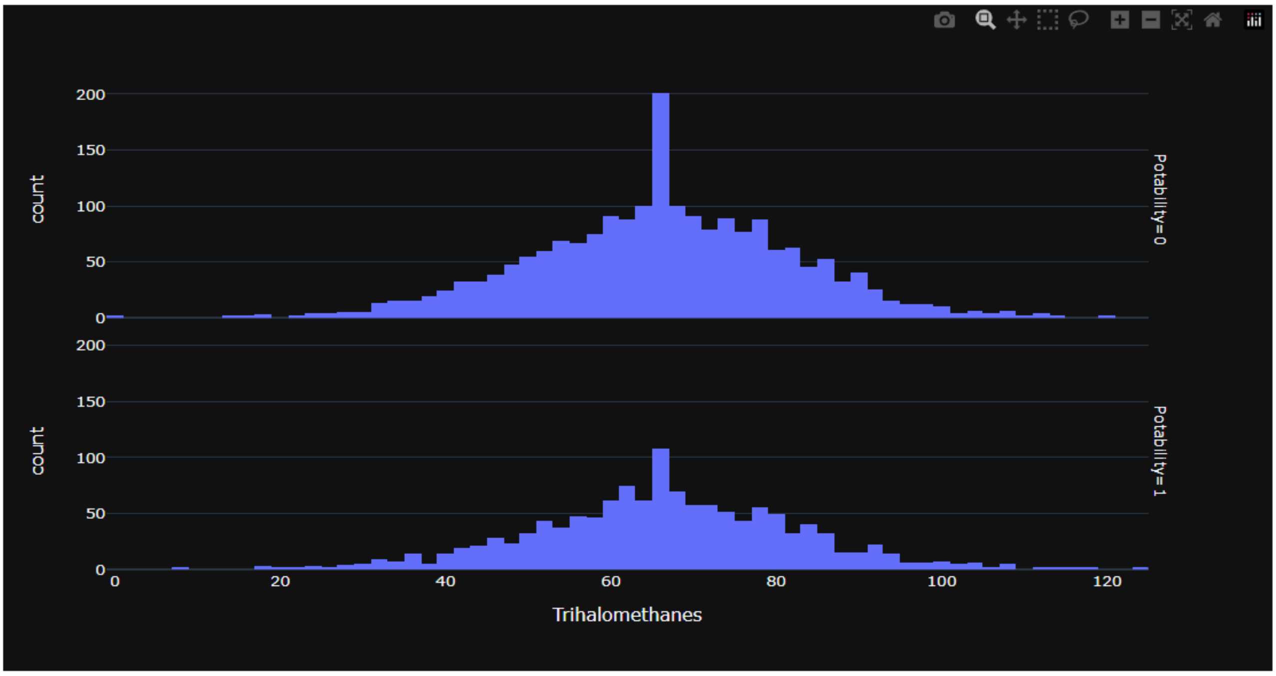
In [24]:
```
sns.scatterplot(x=df['ph'],y=df['Potability'])
plt.show()
```



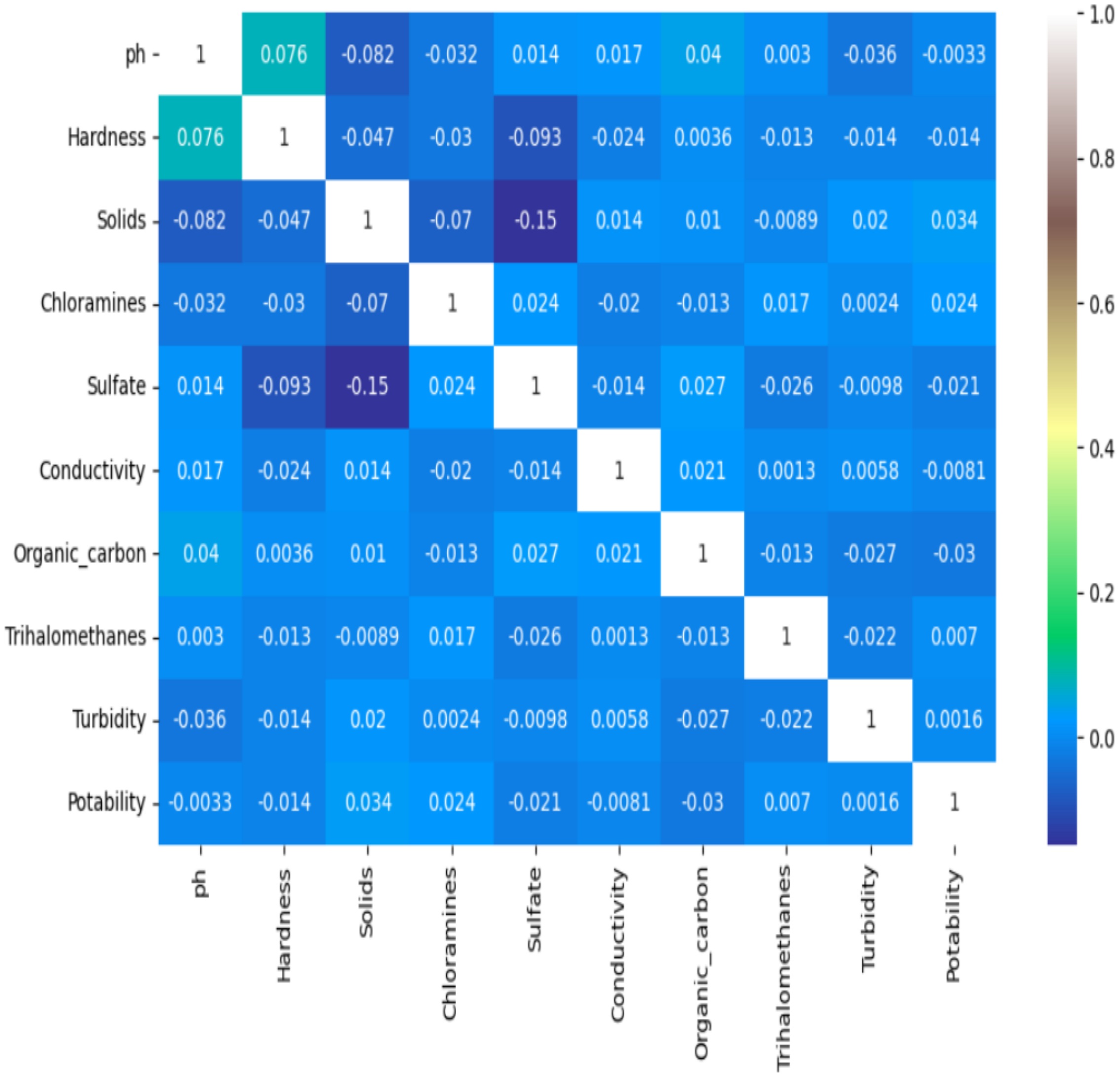
In [25]:
```
sns.scatterplot(x=df['Hardness'],y=df['Solids'])
plt.show()
```



In [26]:
```
fig = px.histogram (df, x = "Trihalomethanes",  facet_row = "Potability",  template = 'plotly_dark')
fig.show ()
```



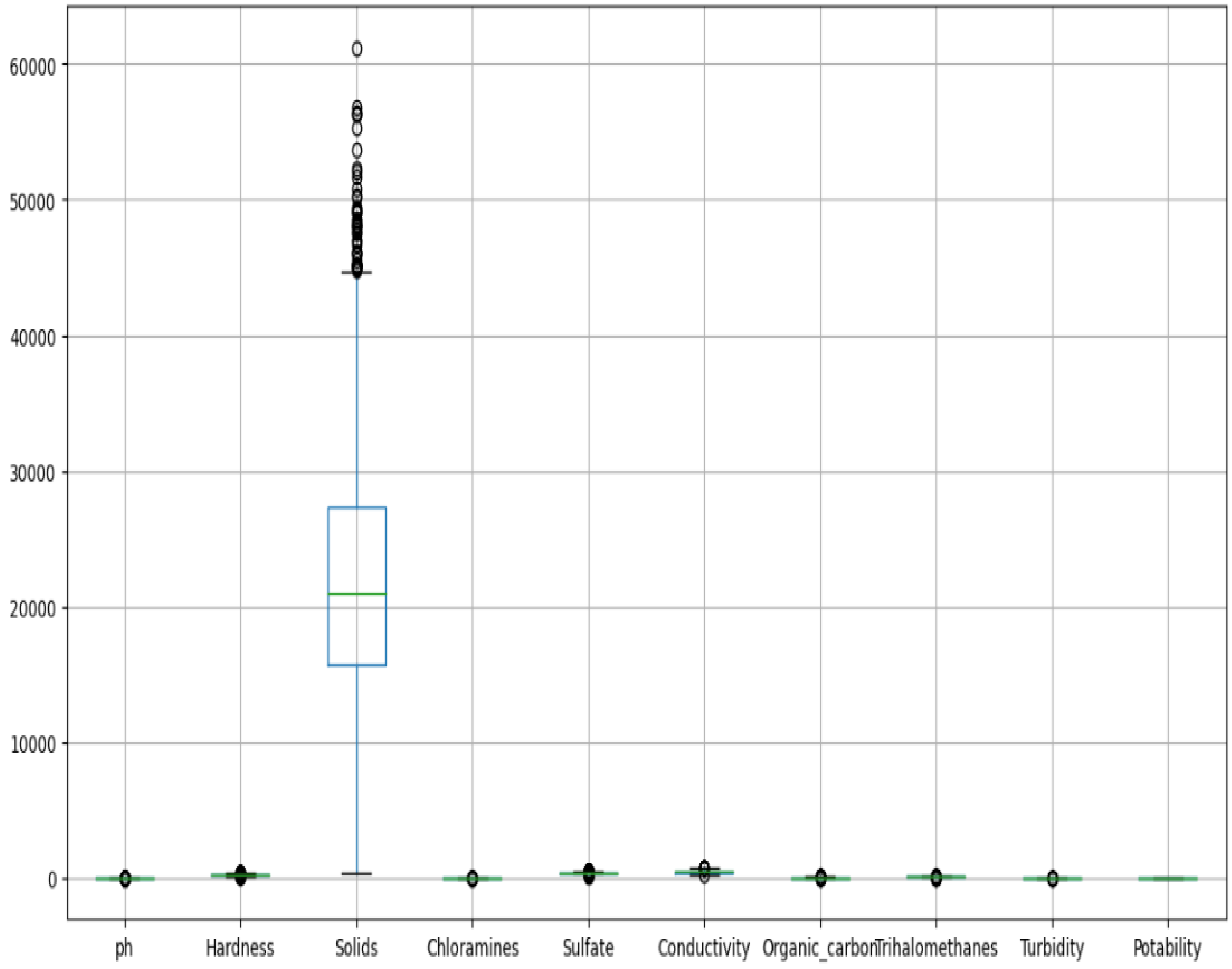
```
In [27]: # create a correlation heatmap
sns.heatmap(df.corr(),annot =True,cmap='terrain')
fig= plt.gcf()
fig.set_size_inches(11,6)
plt.show()
```



- **Correlation Heatmap used For Dimensionality Reduction**

In [28]:
```
df.boxplot(figsize=(14,7))
```

Out[28]:
```
<AxesSubplot: >
```



In [29]:
```
df['Solids'].describe()
```

Out[29]:
```
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
```

Partitioning

In [30]:
```
X = df.drop('Potability',axis=1)
```

In [31]:
```
Y = df['Potability']
```

In [32]:
```
X
```

Out[32]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity
0	7.080795	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135
1	3.716080	129.422921	18630.057858	6.635246	333.775777	592.885359	15.180013	56.329076	4.500656
2	8.099124	224.236259	19909.541732	9.275884	333.775777	418.606213	16.868637	66.420093	3.055934
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.341674	4.628771
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075
...	...	...	...	...	...	...	...	...	...
3271	4.668102	193.681735	47580.991603	7.166639	359.948574	526.424171	13.894419	66.687695	4.435821
3272	7.808856	193.553212	17329.802160	8.061362	333.775777	392.449580	19.903225	66.396293	2.798243
3273	9.419510	175.762646	33155.578218	7.350233	333.775777	432.044783	11.039070	69.845400	3.298875
3274	5.126763	230.603758	11983.869376	6.303357	333.775777	402.883113	11.168946	77.488213	4.708658
3275	7.874671	195.102299	17404.177061	7.509306	333.775777	327.459760	16.140368	78.698446	2.309149

3276 rows × 9 columns

In [33]:
```
Y
```

Out[33]:
```
0       0
1       0
2       0
3       0
4       0
       ..
3271    1
3272    1
3273    1
3274    1
3275    1
Name: Potability, Length: 3276, dtype: int64
```

- **Data Partitioning**

Data set is initially divided into two parts X and Y where X contains the input data and Y contains the output data or Y is the Target variable in this data set, we have Potability as Y and rest other variables included in X- Ph, Hardness, Solids, Chloramines, Sulphate, Conductivity, Organic_carbon, Trihalomethanes, Turbidity Further X is divided into X_train and X_test similarly Y is divided into Y_train and Y_test.

X_train, Y_train => used for Model Training

X_test, Y_test => used for Model Testing when we test for accuracy

In [34]:
```
from sklearn.model_selection import train_test_split
```

In [35]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=42)
```

In [36]:
```
X_train
```

Out[36]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity
1556	7.080795	158.207647	45243.028953	4.064878	247.180038	399.766965	16.086232	53.502086	4.108857
1362	8.143483	182.432457	10673.582674	6.607835	333.775777	427.545219	13.719331	77.769334	2.572830
2787	5.376078	185.540478	36026.401556	9.649943	343.486633	347.565066	14.004449	66.396293	3.629250
1134	7.535700	221.792481	14829.745971	6.701159	366.412200	583.436488	17.731882	59.686076	4.208354
1509	6.618187	164.254565	13776.621792	5.925462	333.775777	315.199393	12.082169	61.474423	3.797068
...	...	...	...	...	...	...	...	...	...
1095	4.187491	208.374188	21809.709834	5.846112	327.474203	264.508083	11.235144	46.682597	4.592959
1130	7.793915	164.958947	25506.912237	7.868036	358.259200	398.460312	15.297496	66.396293	4.220028
1294	6.630364	186.761088	30939.023214	7.703481	333.775777	330.876083	13.815757	86.753117	3.490588
860	8.783168	218.032840	16183.586649	7.390474	334.053885	389.021616	16.354520	47.100982	4.274137
3174	6.698154	198.286268	34675.862845	6.263602	360.232834	430.935009	12.176678	66.396293	3.758180

2194 rows × 9 columns

In [37]:
```
X_test
```

Out[37]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity
2947	7.080795	183.521107	20461.252710	7.333212	333.119476	356.369022	20.179029	67.019903	4.886634
2782	6.643159	188.913541	32873.820022	6.791509	333.848842	336.561501	14.706810	67.844849	4.562198
1644	7.846058	224.058877	23264.109968	5.922367	300.402620	387.971336	13.406737	43.075186	2.487969
70	7.160467	183.089310	6743.346066	3.803036	277.599099	428.036344	9.799625	90.035374	3.884891
2045	6.615350	179.240661	26392.863612	9.309160	333.775777	496.363562	12.786595	78.262369	4.453443
...	...	...	...	...	...	...	...	...	...
1662	6.006769	226.874099	20279.701038	8.166416	225.516628	275.986595	9.650786	52.640025	4.034755
445	6.728004	201.126896	22888.788065	7.663988	319.463491	325.537539	16.788306	58.961220	4.410697
617	6.284985	196.775056	29213.620386	8.528793	334.477795	574.540671	11.095893	66.396293	5.703288
1474	5.821262	204.048890	37174.005414	7.867815	329.019554	466.783264	13.988707	96.826961	4.371079
2555	5.681811	151.085937	26373.495428	5.651589	333.775777	468.472601	8.623795	102.246447	5.375157

1082 rows × 9 columns

In [38]:
```
Y_train.value_counts()
```

Out[38]:
```
0    1318
1     876
Name: Potability, dtype: int64
```

In [39]:
```
Y_test.value_counts()
```

Out[39]:
```
0    680
1    402
Name: Potability, dtype: int64
```

Normalization

In [40]:
```
#from sklearn.preprocessing import StandardScaler
#sc=StandardScaler()
```

In [41]:
```
#X_train = sc.fit_transform(X_train)
#X_test = sc.transform(X_test)
```

- # Model training

Decision Tree

In [168]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(criterion = 'entropy', min_samples_split=9,splitter = 'best')
```

In [169]:
```
dt.fit(X_train,Y_train)
```

Out[169]:
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', min_samples_split=9)
```

In [170]:
```
from sklearn.metrics import accuracy_score, confusion_matrix
```

In [171]:
```
prediction=dt.predict(X_test)
accuracy_dt=accuracy_score(Y_test,prediction)*100
accuracy_dt
```

Out[171]:
```
58.59519408502772
```

In [172]:
```
print("Accuracy on training set: {:.3f}".format(dt.score(X_train, Y_train)))
print("Accuracy on test set: {:.3f}".format(dt.score(X_test, Y_test)))
```

```
Accuracy on training set: 0.938
Accuracy on test set: 0.586
```

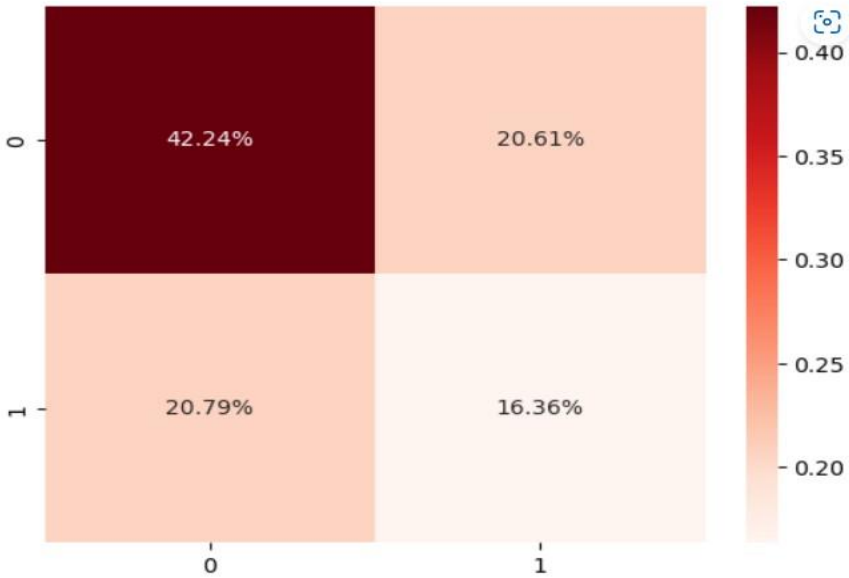
In [173]:
```
accuracy_score(prediction,Y_test)
```

Out[173]:
```
0.5859519408502772
```

In [175]:
```
confusion_matrix(prediction,Y_test)
```

Out[175]:
```
array([[457, 225],
       [223, 177]], dtype=int64)
```

In [176]:
```
cm1 = confusion_matrix(Y_test, prediction)
sns.heatmap(cm1/np.sum(cm1), annot = True, fmt=  '0.2%', cmap = 'Reds')
plt.show()
```



Prediction on only one set of data

In [60]:
```
X_DT=dt.predict([[5.735724, 158.318741,25363.016594,7.728601,377.543291,568.304671,13.626624,75.952337,4.732954]])
```

```
C:\Users\ANIMAY PRAKASH\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:409: UserWarning:

X does not have valid feature names, but DecisionTreeClassifier was fitted with feature names
```

In [61]:
```
X_DT
```

Out[61]:
```
array([0], dtype=int64)
```

KNN

In [190]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [191]:
```
knn=KNeighborsClassifier(metric= 'euclidean', n_neighbors= 24, weights= 'uniform')
knn.fit(X_train,Y_train)
```

Out[191]:
```
KNeighborsClassifier
KNeighborsClassifier(metric='euclidean', n_neighbors=24)
```

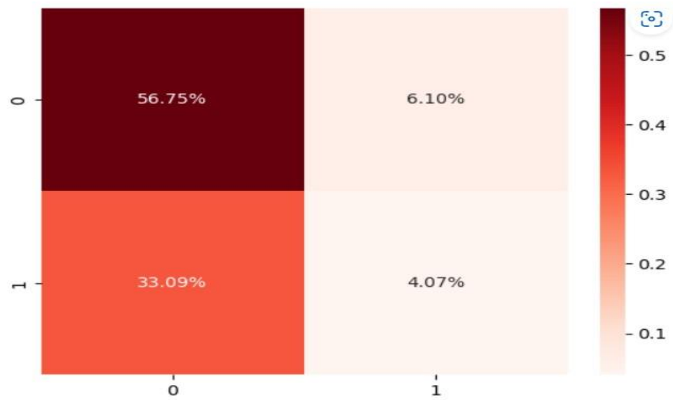
In [192]:
```
prediction_knn=knn.predict(X_test)
accuracy_knn=accuracy_score(Y_test,prediction_knn)*100
print('accuracy_score score      : ',accuracy_score(Y_test,prediction_knn)*100,'%')
```

```
accuracy_score score      :  60.0739371534196 %
```

In [193]:
```
confusion_matrix(prediction,Y_test)
```

Out[193]:
```
array([[457, 225],
       [223, 177]], dtype=int64)
```

In [194]:
```
cm1 = confusion_matrix(Y_test, prediction_knn)
sns.heatmap(cm1/np.sum(cm1), annot = True, fmt=  '0.2%', cmap = 'Reds')
plt.show()
```



Logistic Regression

In [65]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
```

In [66]:
```
lg = LogisticRegression(max_iter=120,random_state=0, n_jobs=20)
```

In [67]:
```
lg.fit(X_train, Y_train)
```

Out[67]:
```
LogisticRegression
LogisticRegression(max_iter=120, n_jobs=20, random_state=0)
```

In [68]:
```
prediction_lg=lg.predict(X_test)
accuracy_lg=accuracy_score(Y_test,prediction_lg)*100
print('accuracy_score score     : ',accuracy_score(Y_test,prediction_lg)*100,'%')
```

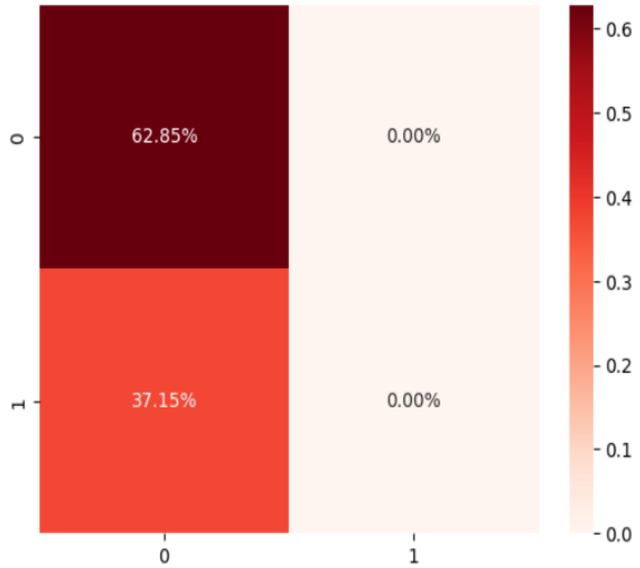
```
accuracy_score score     :  62.84658040665434 %
```

In [69]:
```
print(classification_report(Y_test,prediction_lg))
```

```
              precision    recall  f1-score   support

           0       0.63      1.00      0.77       680
           1       0.00      0.00      0.00       402

    accuracy                           0.63      1082
   macro avg       0.31      0.50      0.39      1082
weighted avg       0.39      0.63      0.49      1082
```

```
C:\Users\ANIMAY PRAKASH\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: Unde
finedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.

C:\Users\ANIMAY PRAKASH\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: Unde
finedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.

C:\Users\ANIMAY PRAKASH\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: Unde
finedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.
```

```
In [119]: cm24 = confusion_matrix(Y_test, prediction_lg)
          sns.heatmap(cm24/np.sum(cm24), annot = True, fmt=  '0.2%', cmap = 'Reds')
          plt.show()
```



Random Forest

```
In [73]: from sklearn.ensemble import RandomForestClassifier
```

```
In [74]: rf = RandomForestClassifier(n_estimators=300,min_samples_leaf=0.16, random_state=42)
```

```
In [75]: rf.fit(X_train, Y_train)
```

```
Out[75]: ▾ RandomForestClassifier
         RandomForestClassifier(min_samples_leaf=0.16, n_estimators=300, random_state=42)
```

```
In [76]: prediction_rf=rf.predict(X_test)
         accuracy_rf=accuracy_score(Y_test,prediction_rf)*100
         print('accuracy_score score     : ',accuracy_score(Y_test,prediction_rf)*100,'%')
```

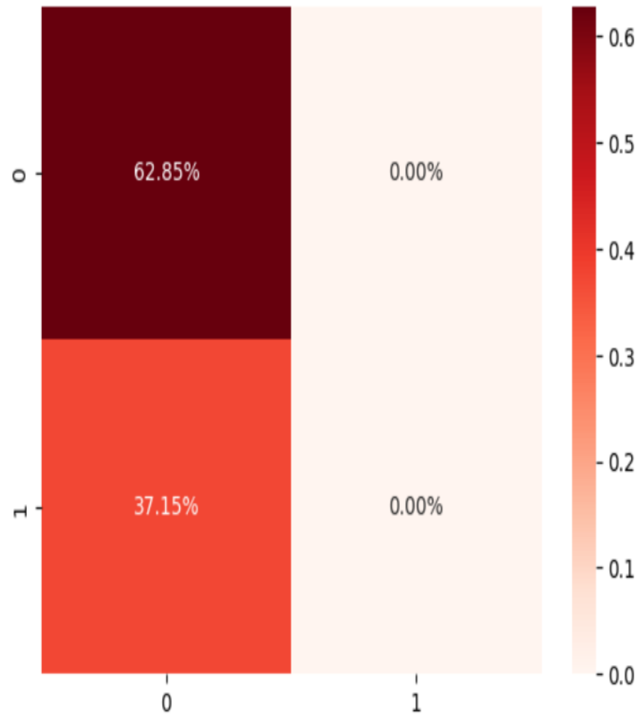
```
accuracy_score score     :  62.84658040665434 %
```

In [77]:
```
print(classification_report(Y_test,prediction_rf))
```

```
              precision    recall  f1-score   support

           0       0.63      1.00      0.77       680
           1       0.00      0.00      0.00       402

    accuracy                           0.63      1082
   macro avg       0.31      0.50      0.39      1082
weighted avg       0.39      0.63      0.49      1082
```

```
C:\Users\ANIMAY PRAKASH\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: Unde
finedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.

C:\Users\ANIMAY PRAKASH\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: Unde
finedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.

C:\Users\ANIMAY PRAKASH\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: Unde
finedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.
```

In [78]:
```
cm3 = confusion_matrix(Y_test, prediction_rf)
sns.heatmap(cm3/np.sum(cm3), annot = True, fmt=  '0.2%', cmap = 'Reds')
plt.show()
```



Gaussian Naive Bayes

In [113]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
```

In [114]:
```
gnb = GaussianNB()
```

In [115]:
```
gnb.fit(X_train,Y_train)
```

Out[115]:

```
▾ GaussianNB
GaussianNB()
```

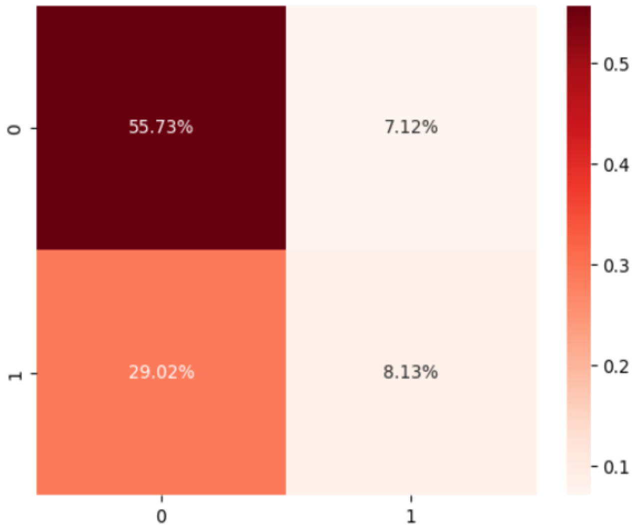
In [116]:
```
prediction_gnb=gnb.predict(X_test)
accuracy_gnb=accuracy_score(Y_test,prediction_gnb)*100
print('accuracy_score score     : ',accuracy_score(Y_test,prediction_gnb)*100,'%')
```

```
accuracy_score score     :  63.86321626617375 %
```

In [117]:
```
print(classification_report(Y_test,prediction_gnb))
```

```
              precision    recall  f1-score   support

           0       0.66      0.89      0.76       680
           1       0.53      0.22      0.31       402

    accuracy                           0.64      1082
   macro avg       0.60      0.55      0.53      1082
weighted avg       0.61      0.64      0.59      1082
```

In [120]:
```
cm8 = confusion_matrix(Y_test, prediction_gnb)
sns.heatmap(cm8/np.sum(cm8), annot = True, fmt=  '0.2%', cmap = 'Reds')
plt.show()
```



XGBoost Classifier

In [79]:
```
from xgboost import XGBClassifier
```

In [80]:
```
xgb = XGBClassifier(max_depth= 8, n_estimators= 125, random_state= 0,  learning_rate= 0.03, n_jobs=5)
```

In [81]:
```
xgb.fit(X_train, Y_train)
```

Out[81]:

XGBClassifier
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', learning_rate=0.03, max_bin=256,
              max_cat_threshold=64, max_cat_to_onehot=4, max_delta_step=0,
              max_depth=8, max_leaves=0, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=125, n_jobs=5,
              num_parallel_tree=1, predictor='auto', random_state=0, ...)
```

In [82]:
```
prediction_xgb = xgb.predict(X_test)
```

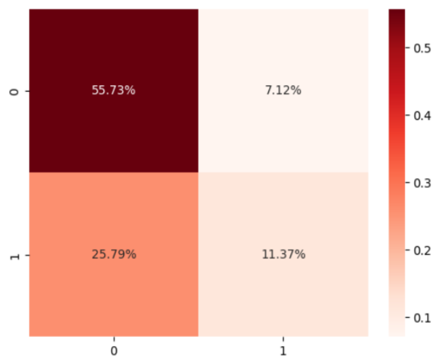
In [83]:
```
accuracy_xgb=accuracy_score(Y_test,prediction_xgb)*100
print('accuracy_score score     : ',accuracy_score(Y_test,prediction_xgb)*100,'%')
```

```
accuracy_score score     :  67.09796672828097 %
```

In [84]:
```
print(classification_report(Y_test,prediction_xgb))
```

```
              precision    recall  f1-score   support

           0       0.68      0.89      0.77       680
           1       0.61      0.31      0.41       402

    accuracy                           0.67      1082
   macro avg       0.65      0.60      0.59      1082
weighted avg       0.66      0.67      0.64      1082
```

In [85]:
```
cm4 = confusion_matrix(Y_test, prediction_xgb)
sns.heatmap(cm4/np.sum(cm4), annot = True, fmt=  '0.2%', cmap = 'Reds')
plt.show()
```



SVM

In [86]:
```python
from sklearn.svm import SVC, LinearSVC
```

In [87]:
```python
svm = SVC(kernel='rbf', random_state = 42)
```

In [88]:
```python
svm.fit(X_train, Y_train)
```

Out[88]:
```
▾ SVC
SVC(random_state=42)
```

In [89]:
```python
prediction_svm = svm.predict(X_test)
```

In [90]:
```python
accuracy_svm=accuracy_score(Y_test,prediction_svm)*100
print('accuracy_score score     : ',accuracy_score(Y_test,prediction_svm)*100,'%')
```

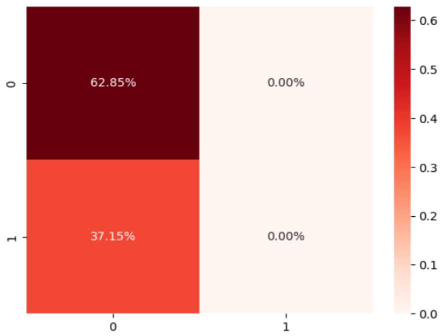
```
accuracy_score score     :  62.84658040665434 %
```

In [91]:
```python
print(classification_report(Y_test,prediction_svm))
```

```
              precision    recall  f1-score   support

           0       0.63      1.00      0.77       680
           1       0.00      0.00      0.00       402

    accuracy                           0.63      1082
   macro avg       0.31      0.50      0.39      1082
weighted avg       0.39      0.63      0.49      1082
```

```
C:\Users\ANIMAY PRAKASH\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.

C:\Users\ANIMAY PRAKASH\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.

C:\Users\ANIMAY PRAKASH\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
```

In [92]:
```python
cm6 = confusion_matrix(Y_test, prediction_svm)
sns.heatmap(cm6/np.sum(cm6), annot = True, fmt=  '0.2%', cmap = 'Reds')
plt.show()
```



AdaBoost Classifier

In [93]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [94]:
```
ada = AdaBoostClassifier(learning_rate= 0.002,n_estimators= 205,random_state=42)
```

In [95]:
```
ada.fit(X_train, Y_train)
```

Out[95]:
```
AdaBoostClassifier
AdaBoostClassifier(learning_rate=0.002, n_estimators=205, random_state=42)
```

In [96]:
```
prediction_ada = ada.predict(X_test)
```

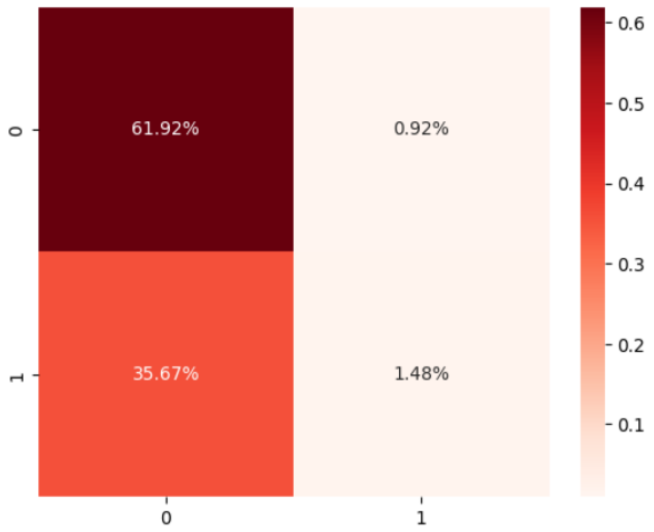
In [97]:
```
accuracy_ada=accuracy_score(Y_test,prediction_ada)*100
print('accuracy_score score      : ',accuracy_score(Y_test,prediction_ada)*100,'%')
```

```
accuracy_score score      :  63.4011090573013 %
```

In [98]:
```
print(classification_report(Y_test,prediction_ada))
```

```
              precision    recall  f1-score   support

           0       0.63      0.99      0.77       680
           1       0.62      0.04      0.07       402

    accuracy                           0.63      1082
   macro avg       0.62      0.51      0.42      1082
weighted avg       0.63      0.63      0.51      1082
```

In [99]:
```
cm7 = confusion_matrix(Y_test, prediction_ada)
sns.heatmap(cm7/np.sum(cm7), annot = True, fmt=  '0.2%', cmap = 'Reds')
plt.show()
```



- **Model optimization**
  Decision Tree

Hyperparameter Tuning / Model Optimization

DT HPT

In [101]:
```
dt.get_params().keys()
```

Out[101]:
```
dict_keys(['ccp_alpha', 'class_weight', 'criterion', 'max_depth', 'max_features', 'max_leaf_nodes', 'min_impurity_decrease', 'm
in_samples_leaf', 'min_samples_split', 'min_weight_fraction_leaf', 'random_state', 'splitter'])
```

In [102]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
dt = DecisionTreeClassifier()
criterion = ["gini", "entropy"]
splitter = ["best", "random"]
min_samples_split = range(1,10)
parameters = dict(splitter=splitter, criterion=criterion, min_samples_split=min_samples_split)
cv = RepeatedStratifiedKFold(n_splits=5, random_state=101)
grid_search_dt = GridSearchCV(estimator=dt, param_grid=parameters, cv=cv,scoring='accuracy')
```

In [103]:
```
grid_search_dt.fit(X_train,Y_train)
```

Out[103]:
```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
    ▸ DecisionTreeClassifier
```

In [104]:
```
print(grid_search_dt.best_params_)
```

```
{'criterion': 'gini', 'min_samples_split': 8, 'splitter': 'random'}
```

In [105]:
```
dt_y_predicted = grid_search_dt.predict(X_test)
dt_y_predicted
```

Out[105]:
```
array([0, 0, 0, ..., 1, 0, 1], dtype=int64)
```

In [106]:
```
dt_grid_score=accuracy_score(Y_test, dt_y_predicted)
dt_grid_score
```

Out[106]:
```
0.6164510166358595
```

In [107]:
```
confusion_matrix(Y_test, dt_y_predicted)
```

Out[107]:
```
array([[477, 203],
       [212, 190]], dtype=int64)
```

- **KNN**

KNN HPT

In [108]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
model = KNeighborsClassifier()
n_neighbors = range(1, 31)
weights = ['uniform', 'distance']
metric = ['euclidean', 'manhattan', 'minkowski']
grid = dict(n_neighbors=n_neighbors,weights=weights,metric=metric)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=1, random_state=1)
grid_search_knn = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv,scoring='accuracy',error_score=0)
grid_search_knn.fit(X_train, Y_train)
print(f"Best: {grid_search_knn.best_score_:.3f} using {grid_search_knn.best_params_}")
means = grid_search_knn.cv_results_['mean_test_score']
stds = grid_search_knn.cv_results_['std_test_score']
params = grid_search_knn.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print(f"{mean:.3f} ({stdev:.3f}) with: {param}")
```

```
Best: 0.588 using {'metric': 'manhattan', 'n_neighbors': 26, 'weights': 'uniform'}
0.536 (0.033) with: {'metric': 'euclidean', 'n_neighbors': 1, 'weights': 'uniform'}
0.536 (0.033) with: {'metric': 'euclidean', 'n_neighbors': 1, 'weights': 'distance'}
0.580 (0.016) with: {'metric': 'euclidean', 'n_neighbors': 2, 'weights': 'uniform'}
0.536 (0.033) with: {'metric': 'euclidean', 'n_neighbors': 2, 'weights': 'distance'}
0.548 (0.019) with: {'metric': 'euclidean', 'n_neighbors': 3, 'weights': 'uniform'}
0.545 (0.017) with: {'metric': 'euclidean', 'n_neighbors': 3, 'weights': 'distance'}
0.584 (0.021) with: {'metric': 'euclidean', 'n_neighbors': 4, 'weights': 'uniform'}
0.553 (0.025) with: {'metric': 'euclidean', 'n_neighbors': 4, 'weights': 'distance'}
0.561 (0.021) with: {'metric': 'euclidean', 'n_neighbors': 5, 'weights': 'uniform'}
0.563 (0.016) with: {'metric': 'euclidean', 'n_neighbors': 5, 'weights': 'distance'}
0.584 (0.023) with: {'metric': 'euclidean', 'n_neighbors': 6, 'weights': 'uniform'}
0.563 (0.019) with: {'metric': 'euclidean', 'n_neighbors': 6, 'weights': 'distance'}
0.560 (0.018) with: {'metric': 'euclidean', 'n_neighbors': 7, 'weights': 'uniform'}
0.562 (0.016) with: {'metric': 'euclidean', 'n_neighbors': 7, 'weights': 'distance'}
0.579 (0.017) with: {'metric': 'euclidean', 'n_neighbors': 8, 'weights': 'uniform'}
0.566 (0.020) with: {'metric': 'euclidean', 'n_neighbors': 8, 'weights': 'distance'}
0.558 (0.024) with: {'metric': 'euclidean', 'n_neighbors': 9, 'weights': 'uniform'}
0.560 (0.023) with: {'metric': 'euclidean', 'n_neighbors': 9, 'weights': 'distance'}
```

In [109]:
```
knn_y_predicted = grid_search_knn.predict(X_test)
knn_y_predicted
```

Out[109]:
```
array([1, 0, 0, ..., 0, 1, 0], dtype=int64)
```

In [110]:
```
knn_grid_score=accuracy_score(Y_test, knn_y_predicted)
knn_grid_score
```

Out[110]:
```
0.6118299445471349
```

In [111]:
```
grid_search_knn.best_params_
```

Out[111]:
```
{'metric': 'manhattan', 'n_neighbors': 26, 'weights': 'uniform'}
```

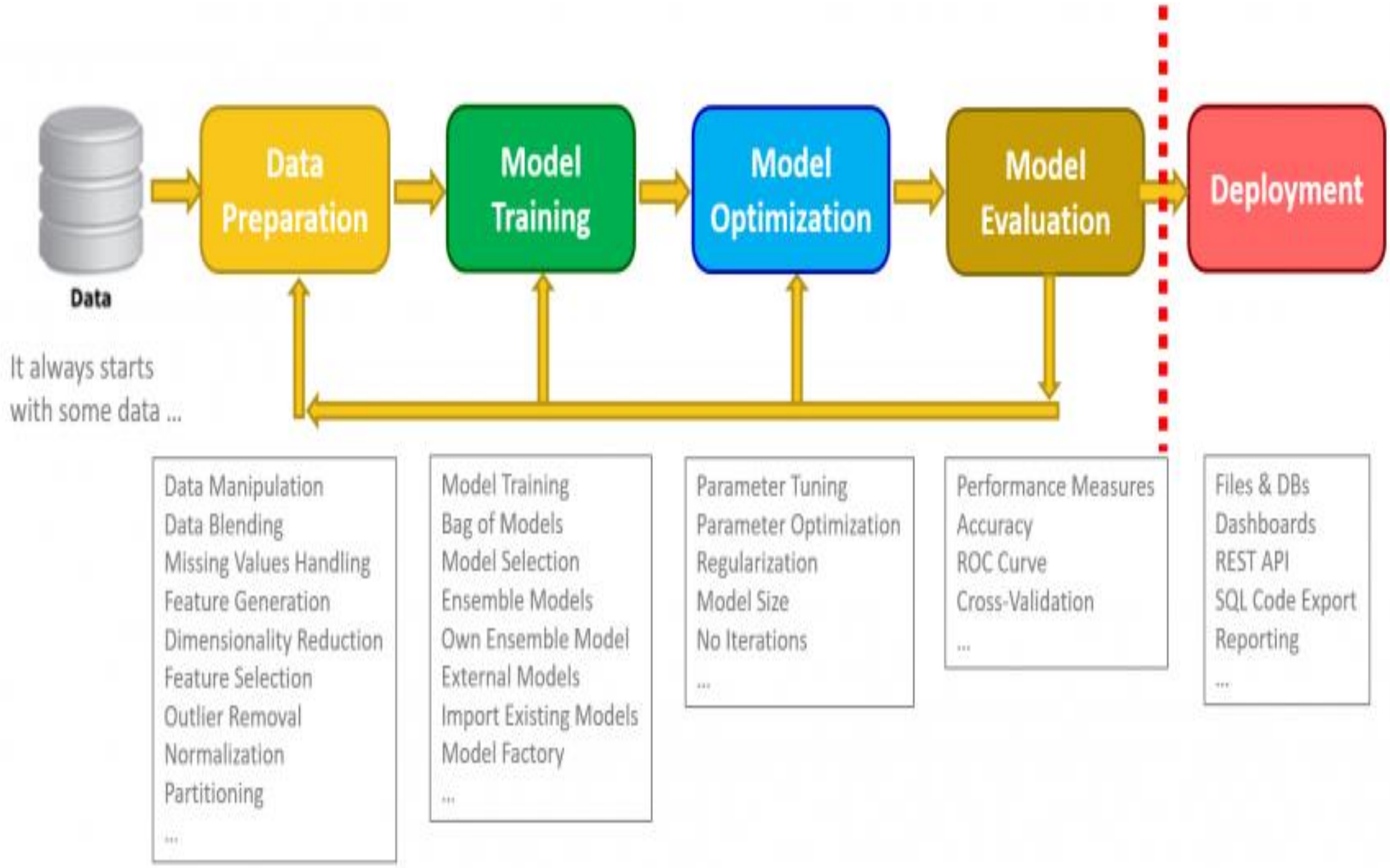
In [112]:
```
confusion_matrix(Y_test, knn_y_predicted)
```

Out[112]:
```
array([[616,  64],
       [356,  46]], dtype=int64)
```

3. Specification of Project

- The Data which is used here is [Dataset]
- In this dataset we first prepare the data for training in which I do various operation like data manipulation , data blending, missing values handling, feature generation ,feature selection ,dimensionality reduction, partioning , label encoding , and Normalization.
- After data preparation we build our model for further process I use 8 models on the given data in which XGBoost classifier Top's the list with 67 % accuracy.
- After data training I optimize the 2 models Decision tree and KNN and by optimizing both I get to accuracy gets increased about 0.1 % .

- **The project pipeline looks like this.**



4. Project Use Interface/Language/Tools/DBMS used in project

- **Language** - Python
- **Editors used** - VsCode , Jupyter Notebook, Python IDLE.
- **Tools** - numpy, pandas ,seaborn ,matplotlib.pyplot , plotly.express etc.
- **Datasets** - Dataset

- **Project use** - Recent Development by Humans degrade our Enviroment which leads to Water Pollution and it is rightly said that water water everywhere but nothing to drink hence this leads to get the knowledge of water quality.
  Drinking bad water leads to many health disease like typhoid, jaundice etc. hence knowledge of water quality is necessary for humans to survive.

5. Outcome/Results of the Project

Results after applying best parameters for **Decision Tree** and **KNN**

	Model	Accuracy_score
4	XGBoost Classifier	67.097967
5	Gaussian Naive Bayes	63.863216
7	AdaBoost	63.401109
2	Logistic Regression	62.846580
3	Random Forest	62.846580
6	SVM	62.846580
1	KNN	60.073937
0	Decision Tree	58.595194

Before Model Optimization

	Model	Accuracy_score
4	XGBoost Classifier	67.097967
5	Gaussian Naive Bayes	63.863216
7	AdaBoost	63.401109
2	Logistic Regression	62.846580
3	Random Forest	62.846580
6	SVM	62.846580
1	KNN	60.813309
0	Decision Tree	60.351201

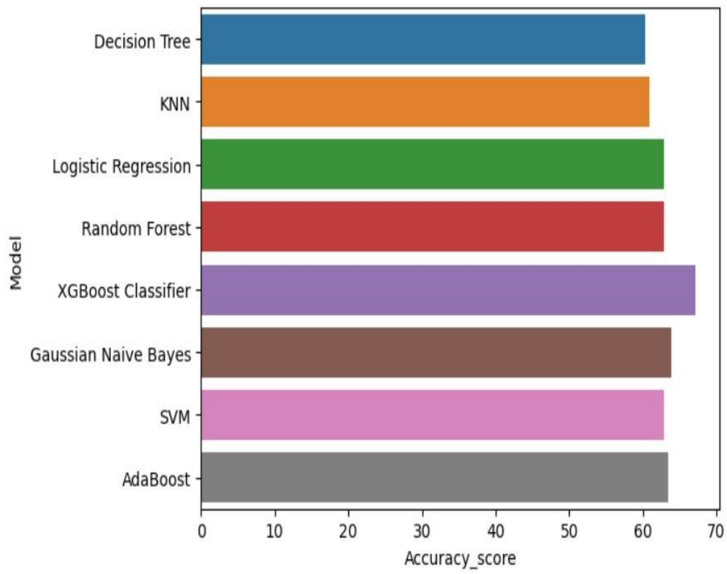
After Model Optimization

6. Contribution Made by you to project

1. Perform Data imputation part.
2. Implement 4 Algorithms.
3. Perform **HyperParameter Tuning/Model optimization** in Decision Tree & KNN Algorithms.
4. Study Research Papers for project ideas suggested by our Mentor. (2021_JECE_Asadollah, 2019_Water_Ahmed, 2022_MESE_Azrour etc)

7. Conclusion

This study investigates the performance of Eight machine learning models to predict the Water Quality Index on a given Dataset. in Recent years many human Development activities led to Enviromental degradation which causes several human Diseases. diseases. Consequently, water quality prediction is essential for sustaining human life. My study performed various experiments and implementing about 8 Models for prediction where the dataset has missing values.
For getting Good results or good accuracies I alone performed Model optimization in Decision tree or KNN models to deal with missing values I use mean value of respective parameters and fill this mean values in place of null space to get better Results after this Extensive experiments were carried out using several machine learning models. Results suggest that the use of the Mean values for filling the missing values plus use of XGBoost Classifier is a better choice and it produces better results in our study.



	Model	Accuracy_score
4	XGBoost Classifier	67.097967
5	Gaussian Naive Bayes	63.863216
7	AdaBoost	63.401109
2	Logistic Regression	62.846580
3	Random Forest	62.846580
6	SVM	62.846580
1	KNN	60.813309
0	Decision Tree	60.351201

- **Future Goal**
  In future, using mixed features and a balanced dataset is intended to obtain generalized results. We also intend to use deep learning with a large dataset for automatic feature extraction and water quality prediction.

8. References

Kaggle. Water Quality Available online: Dataset (accessed on 23rd sept 2022).

https://towardsdatascience.com/understanding-the-confusion-matrix-from-scikit-learn-c51d88929c79

https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

https://www.geeksforgeeks.org/machine-learning-outlier/

https://towardsdatascience.com/seaborn-heatmap-for-visualising-data-correlations-66cbef09c1fe