Machine learning lab–Review II

Water Quality Classification

1) Scope:

Objective: The Main objective of this project is to develop a Machine Learning Model which accurately predict the water samples whether they are potable or non-potable based on various water quality parameters.

Target values: In this project the accuracy classification is will be based on the attributes such as Ph, Hardness, Solids, Chloramines, Sulfate, Conductivity, Organic_carbon, Trihalomethanes, Turbidity and Potability. So based on these parameters we are going to classify the water samples as potable or non-potable.

Data Sources: In this Project we are going to use the dataset which is publicly available which is on the Kaggle and these datasets provide the huge amount of water samples.

Geographical Scope: In this project the water samples are considered from different locations of India depending on the availability of the data on the time period of 2022. So, this analysis Is based on this specific time period.

Limitations: There are some limitations in the dataset as it has some null values and some of the data quality issues. So, to avoid these limitations the random will be placed at the empty spaces by taking mean of values of that particular attribute.

2) Summary of the work:

We all know that water is very essential need for every human in this world and it's a basic right. But now-a-days water pollution is happing in different ways which affects the life a human being. And due to the industrial growth, many waste chemicals are merging into the water which polluted that water or lakes and through lakes rivers and oceans. Meanwhile to provide the good quality of water to the peoples, we need to check whether the water is potable or non-potable for drinking purpose. So, to know whether it is potable or non-potable there is only a way which Is nothing, by examining the water quality of water at particular area. So, by taking some previous data of the water classifications with help of some attributes which affects the qualification of water we can predict the quality of that water, whether the water is good for drinking or not (potable or non-potable). By this way we can provide the good quality of water for every human being which helps in reducing the scarcity of water. So, the main purpose of this project is to build a Machine Learning Model which can predict the quality of the water based of the previous dataset and some water quality measures. So, for the better accuracy rate the collected data is going to be defined and pre-processed and then by separating the data into training and testing, then by applying some Machine Learning Algorithms finding the accuracy of the water by those algorithms and predicting the best algorithm on the basis of high accuracy. So, for this project the following algorithms are going to be used Logestic Regression, KNN Algorithm, SVM algorithm, Decision Tree algorithm and Random Forest Algorithm and finding the best Algorithm which gives the more accuracy of the water samples.

3) Existing Works:

There are many researches have been conducted in the field of water quality classification using the machine learning algorithms. The following papers will provide the valuable information regarding the methodologies and approaches to the problem.

1. Title: Classification of water potability using Machine Learning Algorithms.
   Author: Husain Yusuf.
   Year: 2022.
   Scope: This paper aims to evaluate the performance of different types of Machine Learning algorithms for the water quality classification. So, this paper focussed on various parameters such as Ph, Hardness, TDS, Chloramines, Sulfate, Conductivity, Organic Carbon, Trihalomethanes, turbidity and potability. So, by using these parameters the author researched on the following algorithms KNN Algorithm, Decision Tree Algorithm, Random Forest, ANN Algorithm, Logestic Regression Algorithm and SVM Algorithm.
   Merits: This study highlights the effectiveness of Random Forest in predicting the potability of water with the high accuracy based on the attributes.
   Demerits: This study consists of the Null values and lacks the proper consideration of that null values and not properly handled those values.

2. Title: Classification of water potability using Machine Learning Algorithms.
   Author: Muhammad I'tikafi Khoirul Haq.
   Year: 2021.
   Scope: This paper aims to increase the quality of water by checking the potability of water by using the various types of Machine Learning Algorithms. So, for this, the author focussed on the Decision Tree Algorithm, Navie Bayes Algorithm and K-fold Cross-Validation. And this author took a small public dataset from Kaggle, which can work effectively with a large data set.
   Merits: This study achieves the high accuracy rate with the Machine Learning Algorithms. It also highlights the effectiveness of Decision Tree which gives the high accuracy rate compatible to the dataset.
   Demerits: In this study the Gaussian Navie Bayes model is far better than the other Navie models but has the poor results with an small accuracy, where most of them fail to classify water potability.

3. Title: Comparison of water quality classification models using Machine Learning.
   Author: Neha Radhakrishnan.
   Year: 2020
   Scope: This paper provides a comparison of water quality classification models with the help of Machine Learning Algorithms they are SVM Algorithm, Decision Tree Algorithm and Naive Bayes Algorithm. The attributes that are used for determining the water quality are: pH, DO, BOD and electrical conductivity. Among all algorithms decision tree algorithm gives an high accuracy.
   Merits: The study achieves high accuracy rates for water quality classification using machine learning algorithms. It highlights the effectiveness of Decision Tree algorithm in predicting potability based on a comprehensive set of water quality attributes.
   Demerits: In the study there is some incorrectly classified instances in the confusion matrix.

4. Title: River Water Quality Classification using a Hybrid Machine Learning Technique
   Author: Sakshi Khullar
   Year: 2022
   Scope: This paper mainly focused on Yamuna River water quality classification; they presented a novel hybrid machine learning technique by taking the advantage of an ensemble learning mechanism. The performance of proposed water is classified with the classification techniques like Naive Bayes Algorithm, SVM Algorithm, bagged tree and boosted tree for the attributes pH, DO, TC, BOD, Conductivity.
   Merits: This study achieves high accuracy rates for water quality classification using machine learning algorithms. It highlights the effectiveness of SVM in predicting potability based on a comprehensive set of water quality attributes.
   Demerits: In the above study there are some missing values which leads to the inaccurate result.

5. Title: Water Quality Classification Using SVM And XGBoost Method.
   Author: Hasriq Izzuan Hasnol Yusri
   Year: 2022
   Scope: This paper is aims to evaluate How various pollutants have been endangering water quality over the past decades. It is named as water quality classification (WQC). The WQC is classified by using the algorithms Support Vector Machine (SVM) and Extreme Gradient Boosting (XGBoost). The paper is classified based on the water quality index from using 7 parameters in a dataset and they are Dissolved Oxygen, pH, Coductivity, Biological Oxygen demand, Nitrate, Fecal Coliform, Total coliform.
   Merits: This study achieves the high accuracy rate with the Machine Learning Algorithms. It also highlights the effectiveness of SVM Algorithm which gives the high accuracy compared to the other Algorithms.
   Demerits: In this study there are some null values which were not handled properly and some external factors that a ffect the effectiveness of the accuracy.

4) Tabulate with all content:

Title	Author	Year	Scope	Problem addressed	Methodology	Merits	Demerits
Classification of water potability using Machine Learning Algorithms	Husain Yusuf	2022	Evaluating the performance of different types of Machine Learning algorithms for the water quality classification	The Classification is done on the basis of parameters Ph, Hardness, TDS, Chloramines, Sulfate, Conductivity, Organic Carbon, Trihalomethanes, turbidity and potability	KNN Algorithm, Decision Tree Algorithm, Random Forest, ANN Algorithm, Logestic Regression Algorithm and SVM Algorithm.	This study achieved high accuracy with Random Forest.	Not proper consideration of null values and not properly handled those values.
Classification of water potability using Machine Learning Algorithms	Muhammad I'tikafi Khoirul Haq	2021	To increase the quality of water by checking the potability of water by using the various types of Machine Learning Algorithms	The Classification is done on the basis of a small public dataset from Kaggle, which can work effectively with a large data set.	Decision Tree Algorithm, Navie Bayes Algorithm and K-fold Cross-Validation	This study achieved high accuracy with Decision Tree.	Gaussian Navie Bayes model is better than the other Navie models but has the poor results with a small accuracy, where most of them fail to classify water potability.
Comparison of water quality classification models using Machine Learning.	Neha Radhakrishnan.	2020	Comparison of water quality classification models with the help of Machine Learning Algorithms	The Classification is done on the basis of parameters pH, DO, BOD and electrical conductivity	SVM Algorithm, Decision Tree Algorithm and Naive Bayes Algorithm	This study achieved high accuracy with Decision Tree.	There are some incorrectly classified instances in the confusion matrix.
River Water Quality Classification using a Hybrid Machine Learning Technique	Sakshi Khullar	2022	Focussed on Yamuna River water quality classification.	The Classification is done on the basis of parameters pH, DO, TC, BOD, Conductivity.	Naive Bayes Algorithm, SVM Algorithm, bagged tree and boosted tree	This study achieved high accuracy with SVM.	There are some missing values which leads to the inaccurate result.
Water Quality Classification Using SVM And XGBoost Method.	Hasriq Izzuan Hasnol Yusri	2022	evaluate How various pollutants have been endangering water quality over the past decades	The Classification is done on the basis of parameters Dissolved Oxygen, pH, Coductivity, Biological Oxygen demand, Nitrate, Fecal Coliform, Total coliform.	Support Vector Machine (SVM) and Extreme Gradient Boosting (XGBoost)	This study achieved high accuracy with SVM.	There are some null values which were not handled properly and some external factors that effects the effectiveness of the accuracy.

5) Overall merits and Limitations of the Existing Work:

Merits:

- Most of them achieved the high accuracy rates in the classification of water.
- The Random Forest and SVM Algorithms has gave the high accuracy more probably in all cases.
- Demonstrated the potential of the Machine Learning Algorithms for potability prediction.
- The algorithms are very helpful to predict the water potability.
- Finally, because of this water quality classification, we find the potability of water with high accuracy rates and found the best algorithms.

Limitations:

- Most of the cases the dataset having the null values.
- Sometimes there are some external factors that affected the accuracy of that particular Algorithm.
- Lack of consideration for temporal variations in water quality.
- Some of the algorithms were not very efficient. Every time they giving the different accuracy rates.

6) Description about Data Set:

The dataset we used have various water quality parameters, including characteristics such as pH, hardness, solids concentration, chloramines, sulfate, conductivity, organic carbon, trihalomethanes, turbidity, and a binary variable "Potability." Each row in the dataset represents a sample or observation, likely related to water samples from different sources or locations.

Here's a simple description of the columns in the dataset:

ph: The pH level of the water sample.

Hardness: Measure of the water's hardness, likely indicating mineral content.

Solids: Concentration of solids in the water.

Chloramines: Concentration of chloramines in the water, which are often used as disinfectants.

Sulfate: Sulfate concentration in the water.

Conductivity: Electrical conductivity of the water, often correlated with dissolved minerals.

Organic_carbon: Concentration of organic carbon in the water.

Trihalomethanes: Concentration of trihalomethanes, which can form as byproducts of water disinfection.

Turbidity: Turbidity of the water, a measure of clarity or cloudiness.

Potability: A binary variable indicating whether the water is deemed potable (1) or not (0).

The dataset seems to be related to water quality assessment, with the "Potability" variable indicating whether the water is considered safe for consumption (potable) or not.

To gain deeper insights and draw meaningful conclusions from this dataset, further analysis, data preprocessing, and potentially applying machine learning or statistical techniques could be considered, depending on the specific goals and objectives.

7) CODE:

Importing Necessary libraries

```
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import plotly.figure_factory as ff
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

from collections import Counter
plt.style.use('fivethirtyeight')
colors_blue = ["#132C33", "#264D58", '#17869E', '#51C4D3', '#B4DBE9']
colors_dark = ["#1F1F1F", "#313131", '#636363', '#AEAEAE', '#DADADA']
colors_green = ['#01411C','#4B6F44','#4F7942','#74C365','#D0F0C0']
```

Importing Data

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

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	NaN	204.890456	20791.31898	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	0
1	3.716080	129.422921	18630.05786	6.635246	NaN	592.885359	15.180013	56.329076	4.500656	0
2	8.099124	224.236259	19909.54173	9.275884	NaN	418.606213	16.868637	66.420093	3.055934	0
3	8.316766	214.373394	22018.41744	8.059332	356.886136	363.266516	18.436525	100.341674	4.628771	0
4	9.092223	181.101509	17978.98634	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	0
...	...	...	...	...	...	...	...	...	...	...
3271	4.668102	193.681736	47580.99160	7.166639	359.948574	526.424171	13.894419	66.687695	4.435821	1
3272	7.808856	193.553212	17329.80216	8.061362	NaN	392.449580	19.903225	NaN	2.798243	1
3273	9.419510	175.762646	33155.57822	7.350233	NaN	432.044783	11.039070	69.845400	3.298875	1
3274	5.126763	230.603758	11983.86938	6.303357	NaN	402.883113	11.168946	77.488213	4.708658	1
3275	7.874671	195.102299	17404.17706	7.509306	NaN	327.459761	16.140368	78.698446	2.309149	1

3276 rows × 10 columns

First 5 Rows of the Data

```
df.head(5)
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	NaN	204.890456	20791.31898	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	0
1	3.716080	129.422921	18630.05786	6.635246	NaN	592.885359	15.180013	56.329076	4.500656	0
2	8.099124	224.236259	19909.54173	9.275884	NaN	418.606213	16.868637	66.420093	3.055934	0
3	8.316766	214.373394	22018.41744	8.059332	356.886136	363.266516	18.436525	100.341674	4.628771	0
4	9.092223	181.101509	17978.98634	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	0

Last 5 Rows of the Data

```
df.tail(5)
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
3271	4.668102	193.681736	47580.99160	7.166639	359.948574	526.424171	13.894419	66.687695	4.435821	1
3272	7.808856	193.553212	17329.80216	8.061362	NaN	392.449580	19.903225	NaN	2.798243	1
3273	9.419510	175.762646	33155.57822	7.350233	NaN	432.044783	11.039070	69.845400	3.298875	1
3274	5.126763	230.603758	11983.86938	6.303357	NaN	402.883113	11.168946	77.488213	4.708658	1
3275	7.874671	195.102299	17404.17706	7.509306	NaN	327.459761	16.140368	78.698446	2.309149	1

Data Info

```
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
```

Describing Data

```
df.describe()
```

	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	15666.690300	6.127421	307.699498	365.734414	12.065801	55.844536	3.439711	0.000000
50%	7.036752	196.967627	20927.833605	7.130299	333.073546	421.884968	14.218338	66.622485	3.955028	0.000000
75%	8.062066	216.667456	27332.762125	8.114887	359.950170	481.792305	16.557652	77.337473	4.500320	1.000000
max	14.000000	323.124000	61227.196010	13.127000	481.030642	753.342620	28.300000	124.000000	6.739000	1.000000

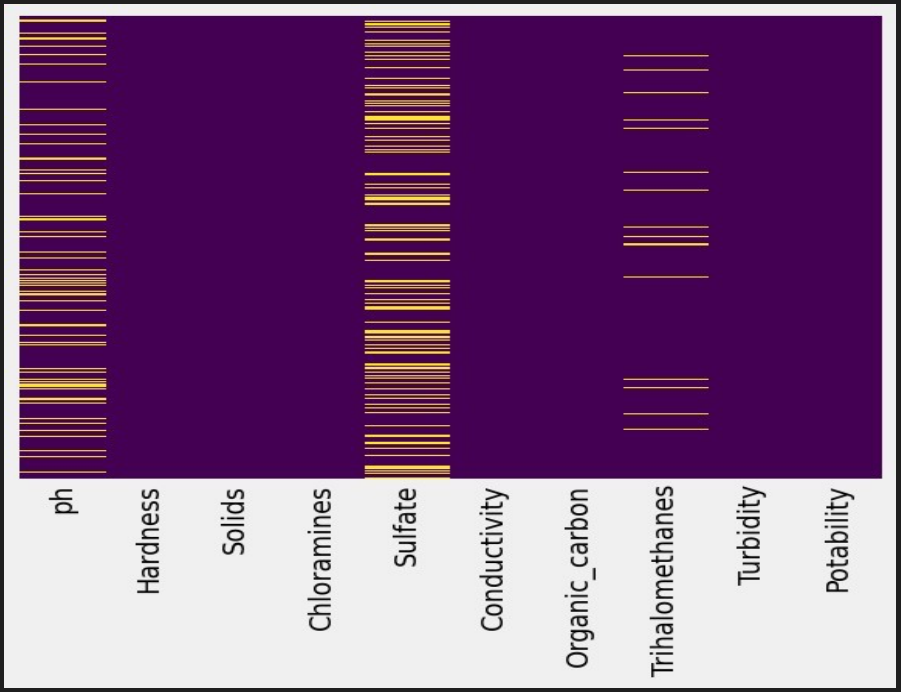
Total Null values in each column of Data

```
df.isnull().sum()
```

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

Heatmap representation of the Null values.

```
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```



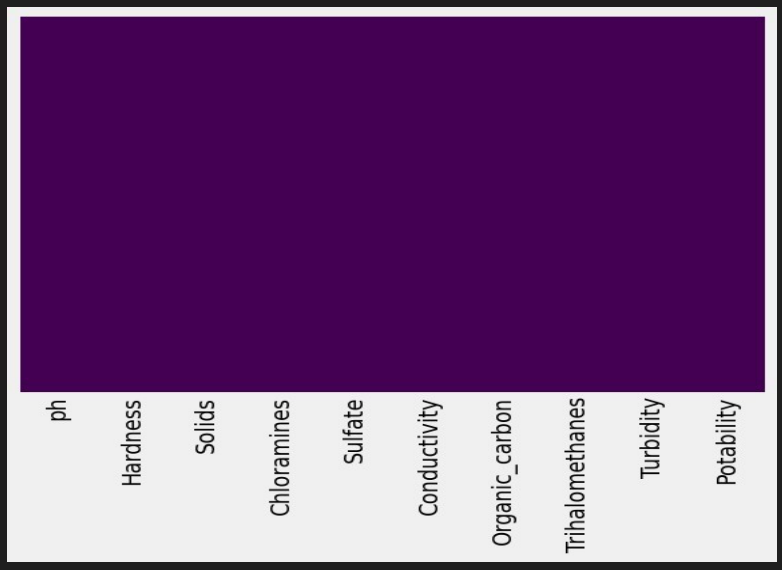
Handling the Null Values

```
df.fillna(df.mean(),inplace=True)
df.isnull().sum()
```

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

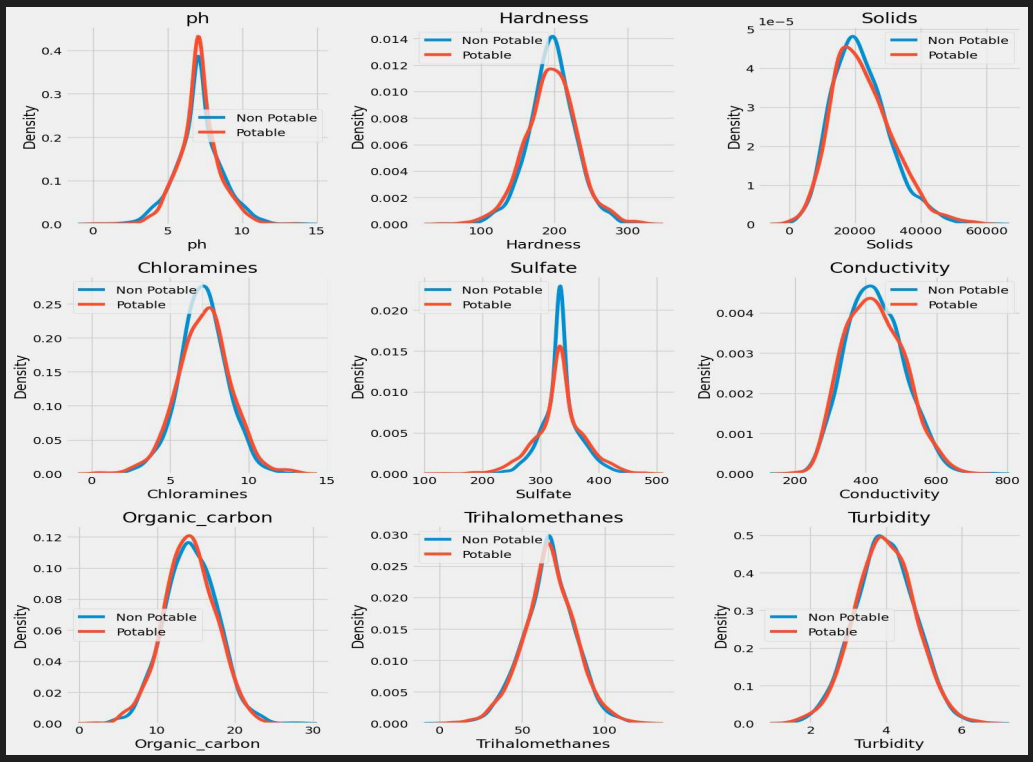
Heatmap representation for the Updated Null Values.

```
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```



Graph for each attribute wrt to Potability attribute(0 and 1).

```
non_potable = df.query("Potability == 0")
potable = df.query("Potability == 1")

plt.figure(figsize=(15,15))

for row, col in enumerate(df.columns[: 9]):
    # df.columns[: 9] because the 10th column is also "Potability"
    plt.subplot(3, 3, row + 1)
    plt.title(col)
    sns.kdeplot(x=non_potable[col], label="Non Potable")
    sns.kdeplot(x=potable[col], label="Potable")
    plt.legend()

plt.tight_layout()
```

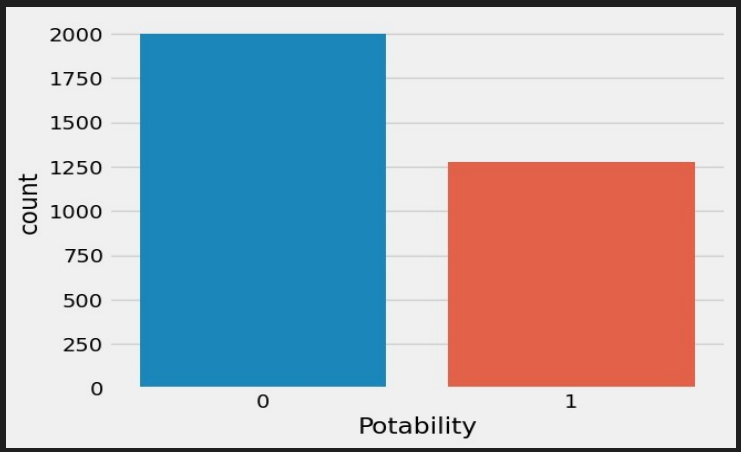


Potability Attribute having a values as 1 and 0 with Sns Graph.

```
df.Potability.value_counts()
```

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

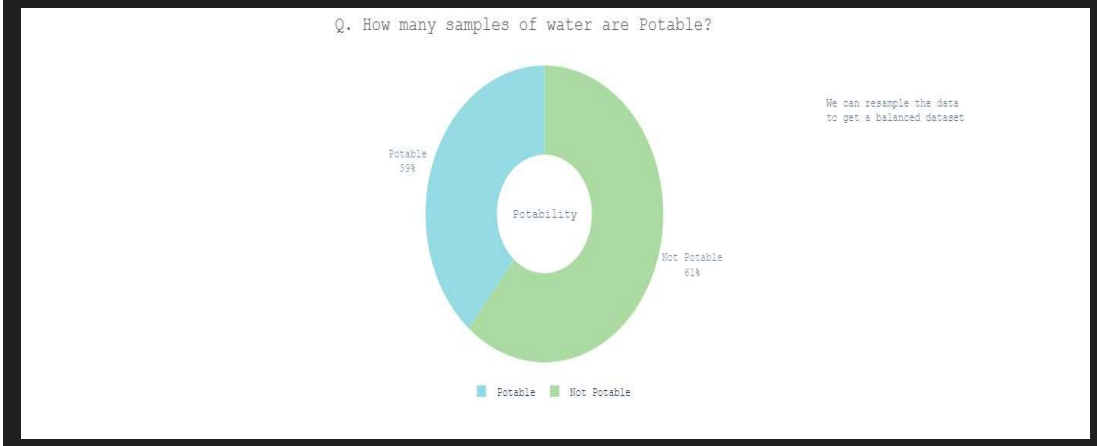
```
sns.countplot(df['Potability'])
plt.show()
```



Percentage Representation of Potablities in Graph

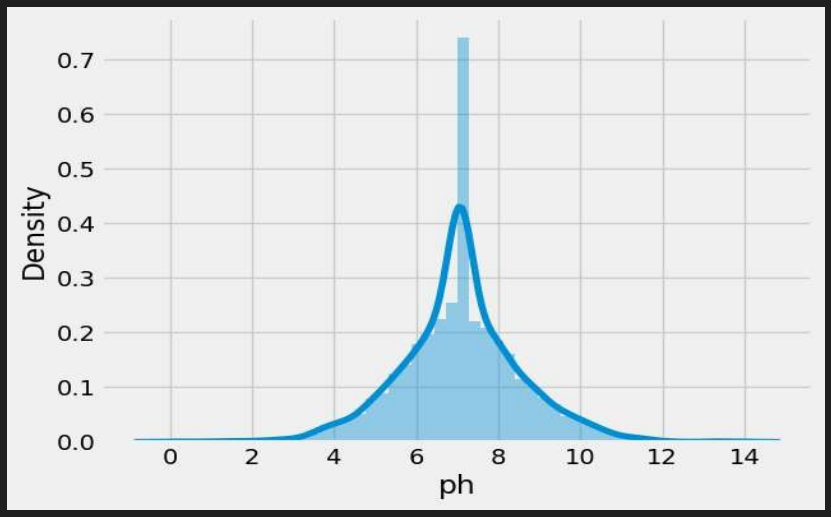
```
d= pd.DataFrame(df['Potability'].value_counts())
```

```
fig = px.pie(d,values='Potability',names=['Not
Potable','Potable'],hole=0.4,opacity=0.6,
             color_discrete_sequence=[colors_green[3],colors_blue[3]],
              labels={'label':'Potability','Potability':'No. Of Samples'})

fig.add_annotation(text='We can resample the data<br> to get a balanced dataset',
                   x=1.2,y=0.9,showarrow=False,font_size=12,opacity=0.7,font_famil
y='monospace')
fig.add_annotation(text='Potability',
                   x=0.5,y=0.5,showarrow=False,font_size=14,opacity=0.7,font_famil
y='monospace')

fig.update_layout(
    font_family='monospace',
    title=dict(text='Q. How many samples of water are Potable?',x=0.47,y=0.98,
               font=dict(color=colors_dark[2],size=20)),
    legend=dict(x=0.37,y=-0.05,orientation='h',traceorder='reversed'),
    hoverlabel=dict(bgcolor='white'))

fig.update_traces(textposition='outside', textinfo='percent+label')
fig.show()
```



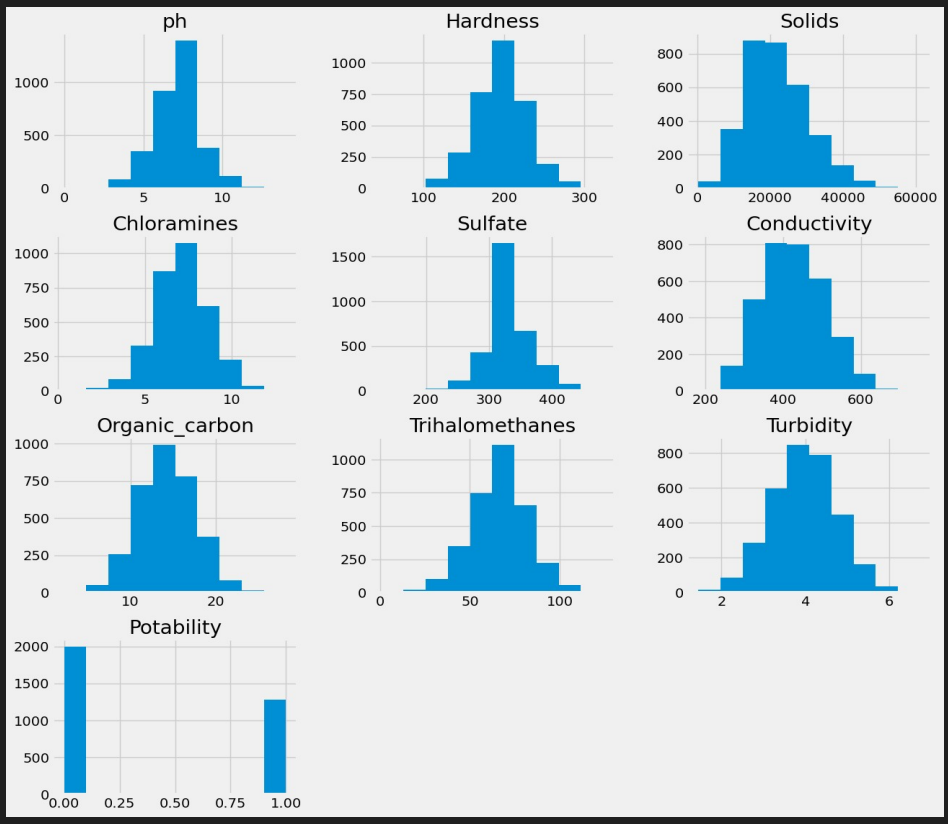
 # potability: the value counts of the data here in the dataset we can see that potability of 0 - the value count is 1998 and 1- the vaue count is 1278. By this we can say that it is an imblanced data

```
sns.distplot(df['ph'])
plt.show()
```




```
# dist plot of ph: here we can see the distribution of the plot goes as normal distribution
```

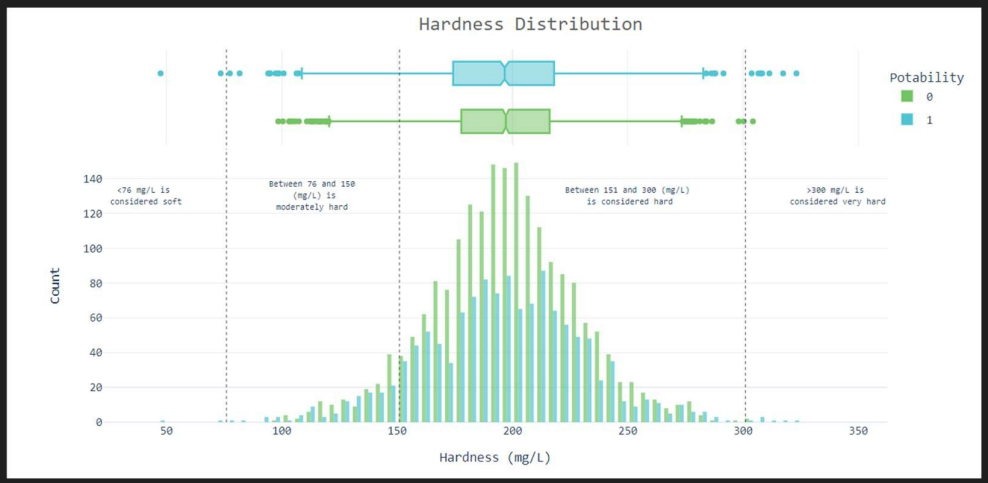
```
df.hist(figsize=(14,14))
plt.show()
```




```
# hist: here the list of all graphs shows the histogram plots of the whole dataset with each features.
```

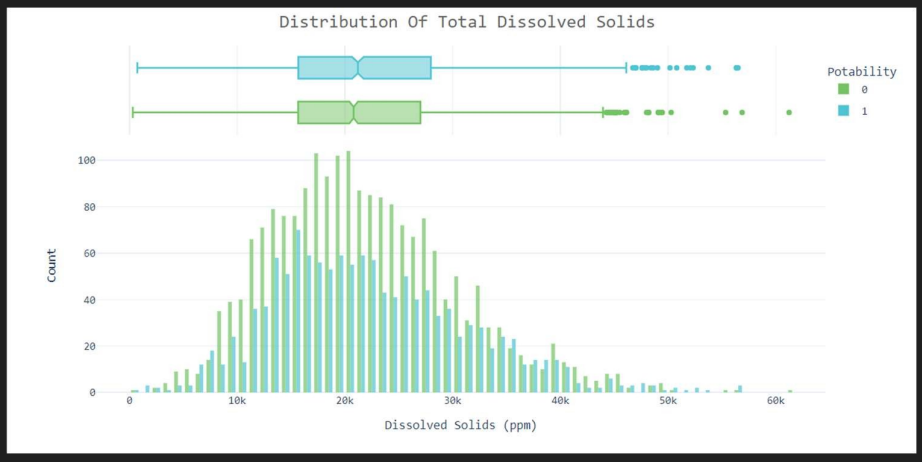
For Hardness:

```
fig =
px.histogram(df,x='Hardness',y=Counter(df['Hardness']),color='Potability',template
='plotly_white',  marginal='box',opacity=0.7,nbins=100,color_discrete_sequence=[co
lors_green[3],colors_blue[3]],
                  barmode='group',histfunc='count')
fig.add_vline(x=151, line_width=1,
line_color=colors_dark[1],line_dash='dot',opacity=0.7)
fig.add_vline(x=301, line_width=1,
line_color=colors_dark[1],line_dash='dot',opacity=0.7)
fig.add_vline(x=76, line_width=1,
line_color=colors_dark[1],line_dash='dot',opacity=0.7)

fig.add_annotation(text='<76 mg/L is<br> considered
soft',x=40,y=130,showarrow=False,font_size=9)
fig.add_annotation(text='Between 76 and 150<br> (mg/L) is<br>moderately
hard',x=113,y=130,showarrow=False,font_size=9)
fig.add_annotation(text='Between 151 and 300 (mg/L)<br> is considered
hard',x=250,y=130,showarrow=False,font_size=9)
fig.add_annotation(text='>300 mg/L is<br> considered very
hard',x=340,y=130,showarrow=False,font_size=9)

fig.update_layout(
    font_family='monospace',
    title=dict(text='Hardness Distribution',x=0.53,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Hardness (mg/L)',
    yaxis_title_text='Count',
    legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,tracegroupgap=
5),bargap=0.3,)
fig.show()
```



For Solids:

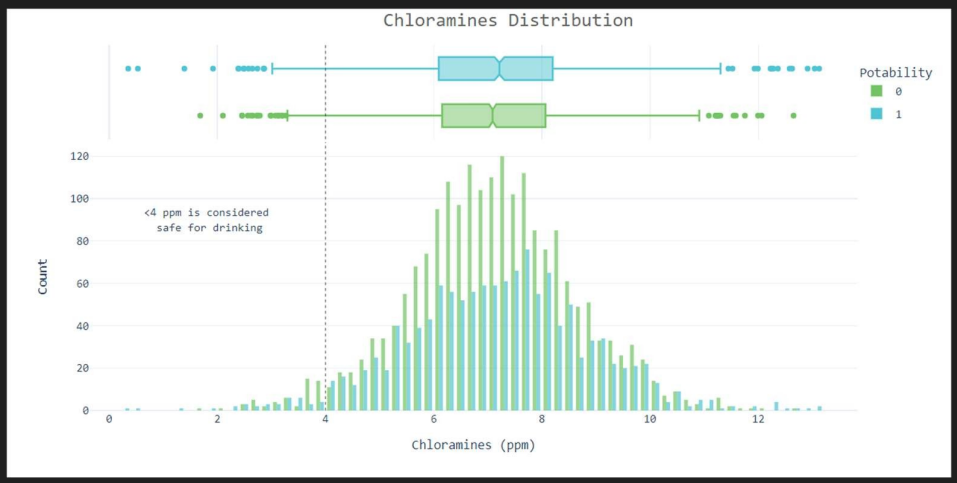
```
fig =
px.histogram(df,x='Solids',y=Counter(df['Solids']),color='Potability',template='pl
otly_white',
                   marginal='box',opacity=0.7,nbins=100,color_discrete_sequence=[co
lors_green[3],colors_blue[3]],
                   barmode='group',histfunc='count')

fig.update_layout(
    font_family='monospace',
    title=dict(text='Distribution Of Total Dissolved Solids',x=0.5,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Dissolved Solids (ppm)',
    yaxis_title_text='Count',
    legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,tracegroupgap=
5),
    bargap=0.3,
)
fig.show()
```



For Chloramines:

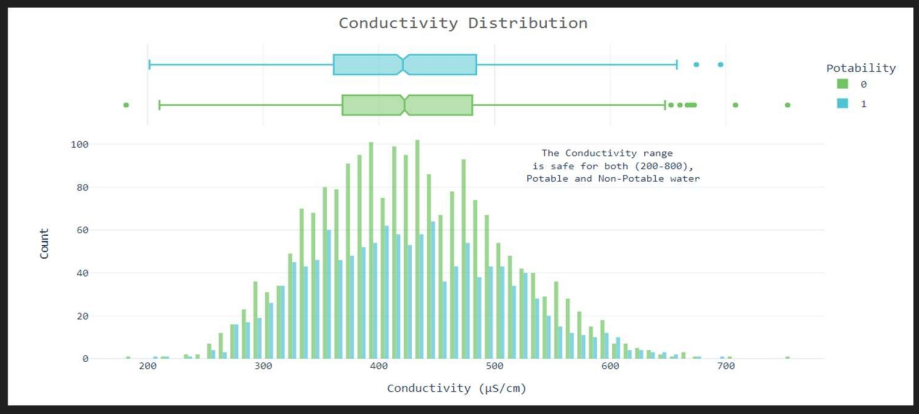
```
fig =
px.histogram(df,x='Chloramines',y=Counter(df['Chloramines']),color='Potability',te
mplate='plotly_white',
                   marginal='box',opacity=0.7,nbins=100,color_discrete_sequence=[co
lors_green[3],colors_blue[3]],
                   barmode='group',histfunc='count')

fig.add_vline(x=4, line_width=1,
line_color=colors_dark[1],line_dash='dot',opacity=0.7)

fig.add_annotation(text='<4 ppm is considered<br> safe for
drinking',x=1.8,y=90,showarrow=False)

fig.update_layout(
```

```
    font_family='monospace',
    title=dict(text='Chloramines Distribution',x=0.53,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Chloramines (ppm)',
    yaxis_title_text='Count',
    legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,tracegroupgap=
5),
    bargap=0.3,
)
fig.show()
```



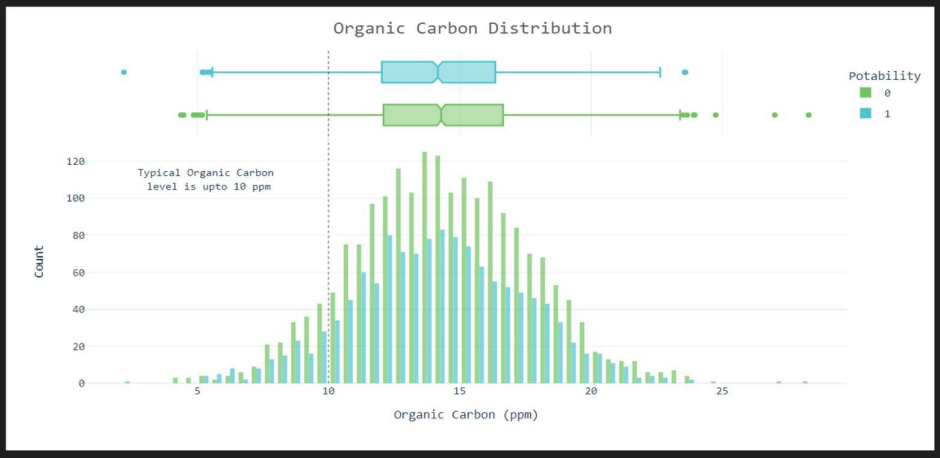
For Conductivity:

```
fig =
px.histogram(df,x='Conductivity',y=Counter(df['Conductivity']),color='Potability',
template='plotly_white',
                  marginal='box',opacity=0.7,nbins=100,color_discrete_sequence=[co
lors_green[3],colors_blue[3]],
                  barmode='group',histfunc='count')

fig.add_annotation(text='The Conductivity range <br> is safe for both (200-
800),<br> Potable and Non-Potable water',
                   x=600,y=90,showarrow=False)

fig.update_layout(
    font_family='monospace',
    title=dict(text='Conductivity Distribution',x=0.5,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Conductivity (µS/cm)',
    yaxis_title_text='Count',
    legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,tracegroupgap=
5),
    bargap=0.3,
)
fig.show()
```



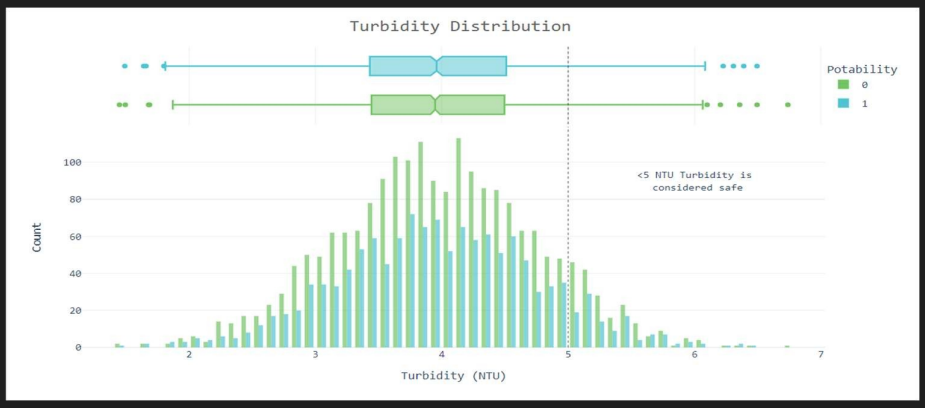
For Organic Carbon:

```
fig =
px.histogram(df,x='Organic_carbon',y=Counter(df['Organic_carbon']),color='Potabili
ty',template='plotly_white',
                  marginal='box',opacity=0.7,nbins=100,color_discrete_sequence=[co
lors_green[3],colors_blue[3]], barmode='group',histfunc='count')
fig.add_vline(x=10, line_width=1,
line_color=colors_dark[1],line_dash='dot',opacity=0.7)
fig.add_annotation(text='Typical Organic Carbon<br> level is upto 10
ppm',x=5.3,y=110,showarrow=False)

fig.update_layout(
    font_family='monospace',
    title=dict(text='Organic Carbon Distribution',x=0.5,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Organic Carbon (ppm)',
    yaxis_title_text='Count',
    legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,tracegroupgap=
5),bargap=0.3,)
fig.show()
```

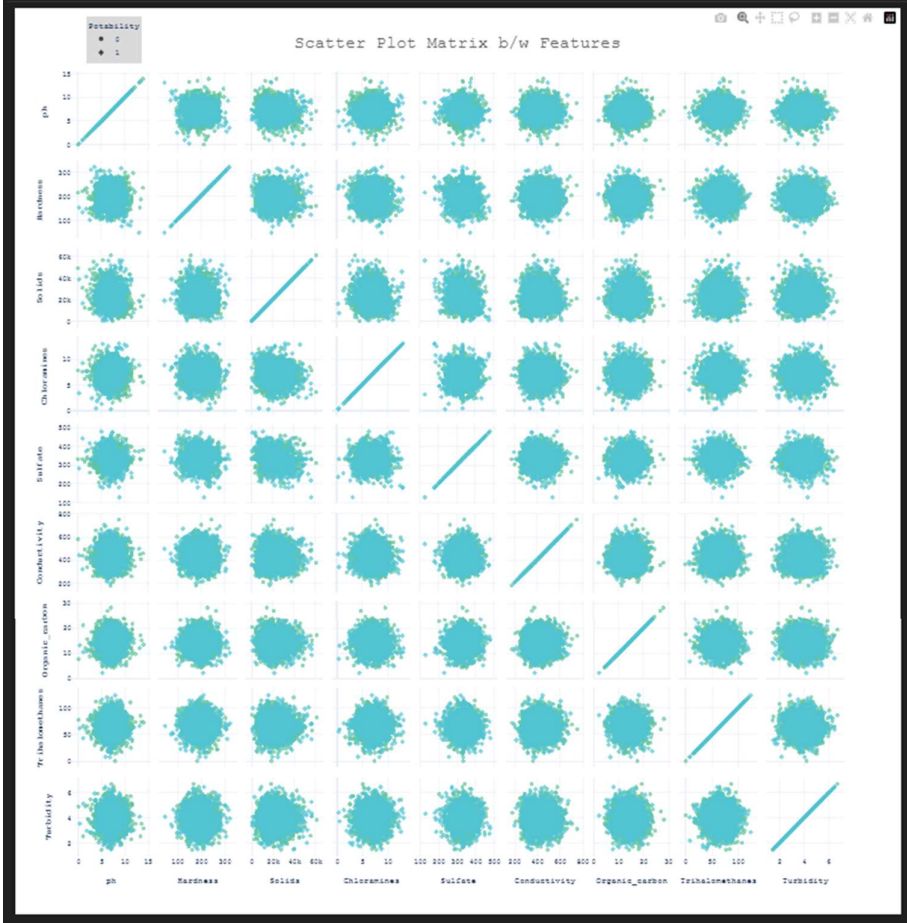


For Turbidity:

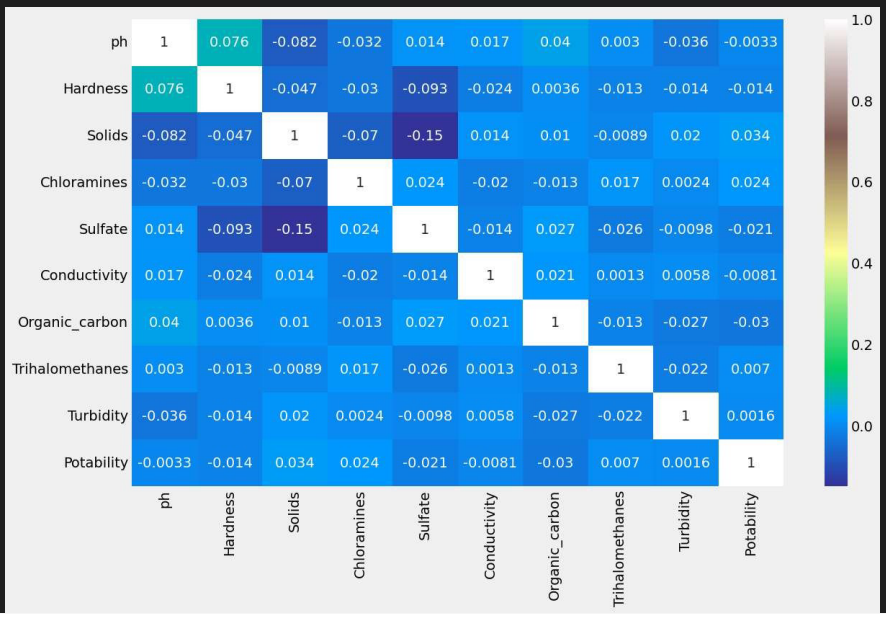
```
fig =
px.histogram(df,x='Turbidity',y=Counter(df['Turbidity']),color='Potability',templa
te='plotly_white',
                  marginal='box',opacity=0.7,nbins=100,color_discrete_sequence=[co
lors_green[3],colors_blue[3]], barmode='group',histfunc='count')

fig.add_vline(x=5, line_width=1,
line_color=colors_dark[1],line_dash='dot',opacity=0.7)

fig.add_annotation(text='<5 NTU Turbidity is<br> considered
safe',x=6,y=90,showarrow=False)

fig.update_layout(
    font_family='monospace',
    title=dict(text='Turbidity Distribution',x=0.5,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Turbidity (NTU)',
    yaxis_title_text='Count',
    legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,tracegroupgap=
5), bargap=0.3,)
fig.show()
```



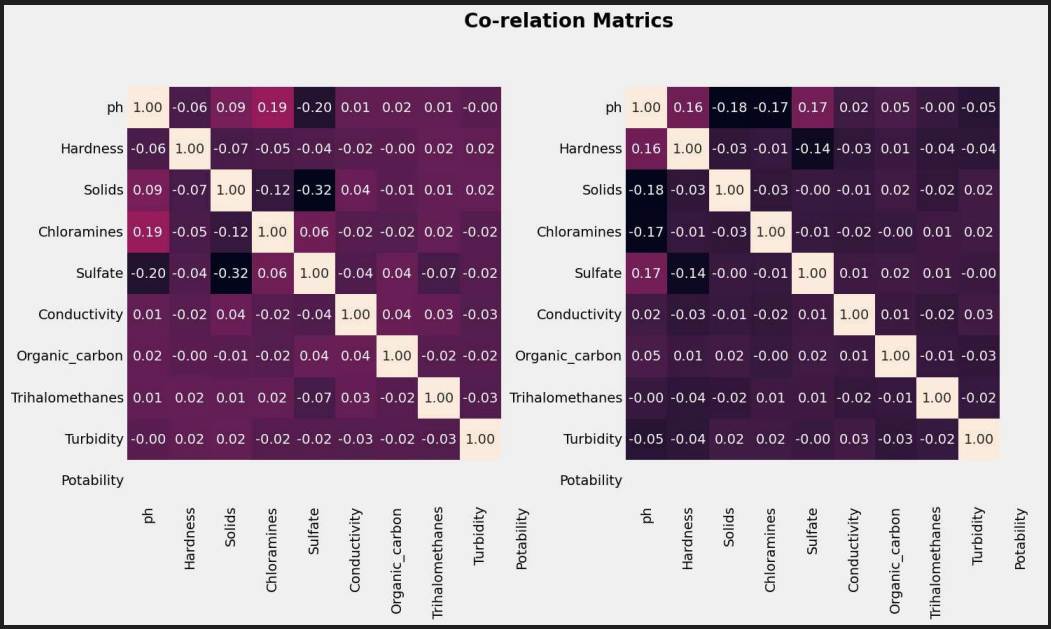

```
fig =
px.scatter_matrix(df,df.drop('Potability',axis=1),height=1250,width=1250,template=
'plotly_white',opacity=0.7,
                        color_discrete_sequence=[colors_blue[3],colors_green[3]],c
olor='Potability',
                       symbol='Potability',color_continuous_scale=[colors_green[3]
,colors_blue[3]])

fig.update_layout(font_family='monospace',font_size=10,
                   coloraxis_showscale=False,
                  legend=dict(x=0.02,y=1.07,bgcolor=colors_dark[4]),
                  title=dict(text='Scatter Plot Matrix b/w Features',x=0.5,y=0.97,
                    font=dict(color=colors_dark[2],size=24)))
fig.show()
```




```
plt.figure(figsize=(13,8))
sns.heatmap(df.corr(),annot=True,cmap='terrain')
plt.show()
```

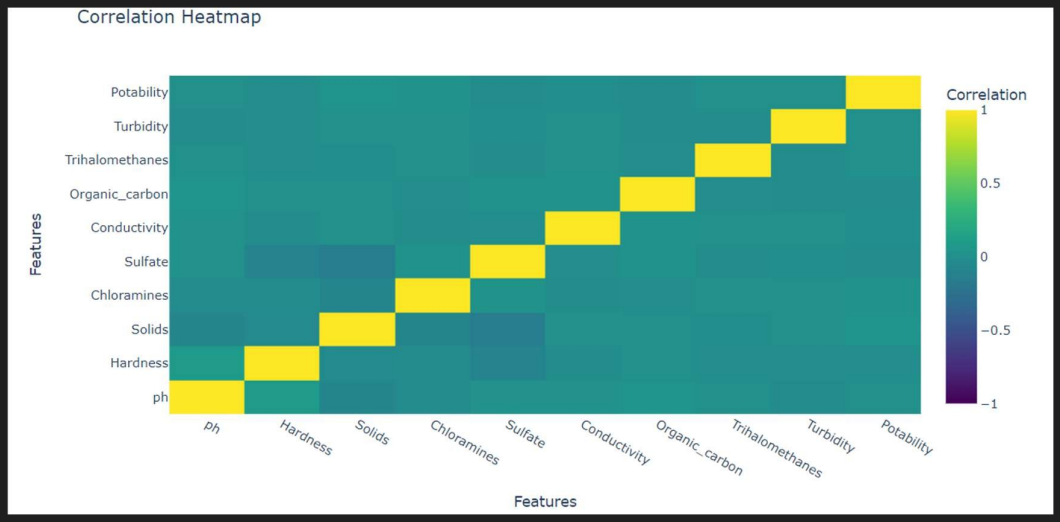


```
# correlation:visualizing correlation using heat map function by
using seaborn. we can see that there is no correlation between
features it means that we cant't reduce the dimension

fig, ax=plt.subplots(nrows=1, ncols=2, figsize=(15,8))
plt.suptitle("Co-relation Matrics", size=20, weight='bold')
ax=ax.flatten()
sns.heatmap(df[df['Potability']==1].corr(), annot=True, square=True, fmt='.2f',
ax=ax[0], cbar=False)
sns.heatmap(df[df['Potability']==0].corr(), annot=True, square=True, fmt='.2f',
ax=ax[1], cbar=False)
```

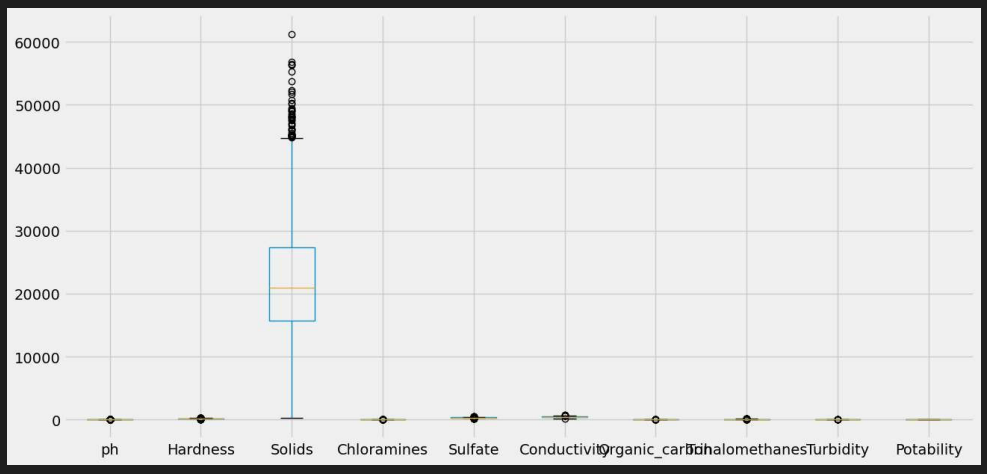



```
# Creating the correlation matrix
correlation_matrix = df.corr()

heatmap = go.Heatmap(
    z=correlation_matrix,
    x=correlation_matrix.columns,
    y=correlation_matrix.index,
    colorscale='viridis',
    zmin=-1,
    zmax=1,
    colorbar=dict(title='Correlation')
)

# Creating the layout
layout = go.Layout(
    title="Correlation Heatmap",
    xaxis=dict(title="Features"),
```

```
    yaxis=dict(title="Features"),
)

# Creating the figure
fig = go.Figure(data=heatmap, layout=layout)

# Displaying the figure
fig.show()
```




```
df.boxplot(figsize=(14,7))
```




```
# box plot: here in the solid feature contains outliers but we can't remove the outliers bcz if we remove the outliers from the solid feature water will be safe to drink every time, but it contains the outlier to make thge water impure means it will tell us whether the water is safe to drink or not. so, we can't remove the outlier to train the model.

X = df.drop("Potability", axis=1).values
Y = df["Potability"].values
```

```
# Confusion Matrix is a table used to evaluate the performance of a classification model in classification problems. It represents the relationship between the actual class labels and the predicted class labels by the model.The Confusion Matrix is typically a 2x2 table and includes four different values:True Positive (TP): Instances where the true positive examples are correctly predicted as positive. False Positive (FP): Instances where the true negative examples are incorrectly predicted as positive. True Negative (TN): Instances where the true negative examples are correctly predicted as negative. False Negative (FN): Instances where the true positive examples are incorrectly predicted as negative. The Confusion Matrix helps in evaluating the performance of a classification model by calculating different metrics such as: Accuracy: The ratio of TP and TN to the total number of data points. Precision: The ratio of TP to TP + FP. Recall or Sensitivity: The ratio of TP to TP + FN. F1 Score: The harmonic mean of precision and recall values. The Confusion Matrix provides a more detailed understanding of a model's classification performance and shows how well the model correctly or incorrectly predicts different classes. Precision Score : Precision Score is an evaluation metric used in classification problems. Precision is defined as the ratio of true positive predictions (TP - True Positive) to the total positive predictions (TP + FP - True Positive + False Positive). In other words, precision aims to minimize false positive predictions (FP - False Positive). Precision Score measures how well a classification model can limit false positive predictions and make accurate positive predictions. A high precision score indicates that fewer false positive predictions are made and that the majority of positive predictions are actually true positives. Precision Score is an important evaluation metric, particularly in imbalanced classification problems and situations where the cost of false positive predictions is high and minimizing false positive predictions is crucial.

from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test =
train_test_split(X,Y,test_size=0.2,random_state=101,shuffle=True)
from sklearn.preprocessing import StandardScaler
# Train Test Split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
random_state=3)

print("X_train", X_train.shape)
print("X_test", X_test.shape)
print("Y_train", Y_train.shape)
print("Y_test", Y_test.shape)
```

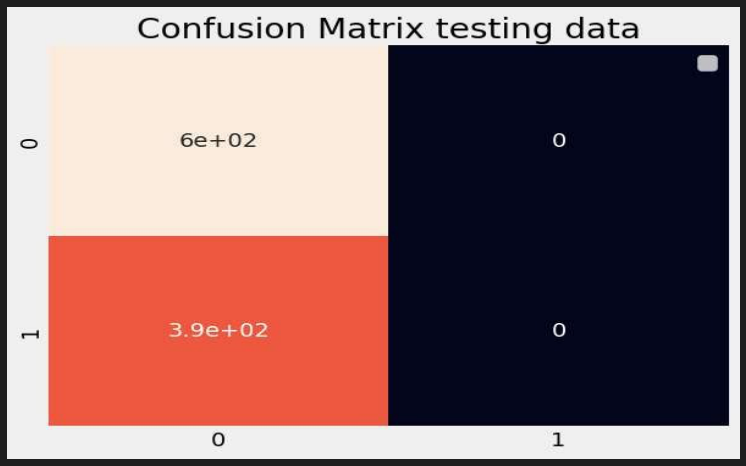
```
X_train (2293, 9)
X_test (983, 9)
Y_train (2293,)
Y_test (983,)
```

Logistic Regression:

```
from sklearn.linear_model import LogisticRegression
log = LogisticRegression(random_state=0).fit(X_train, Y_train)
log.score(X_test, Y_test)
```

```
0.6052899287894201
```

```
#  Confusion matrix
from sklearn.metrics import confusion_matrix
# Make Predictions
pred1=log.predict(np.array(X_test))
plt.title("Confusion Matrix testing data")
sns.heatmap(confusion_matrix(Y_test,pred1),annot=True,cbar=False)
plt.legend()
plt.show()
```

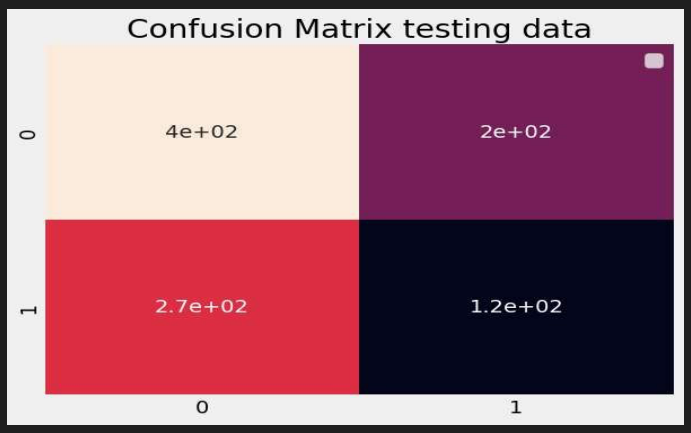


K Nearest Neighbours:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
knn = KNeighborsClassifier(n_neighbors=3)
# Train the model using the training sets
knn.fit(X_train,Y_train)

#Predict Output
predicted= knn.predict(X_test) # 0:Overcast, 2:Mild
print(accuracy_score(Y_test,predicted))
```

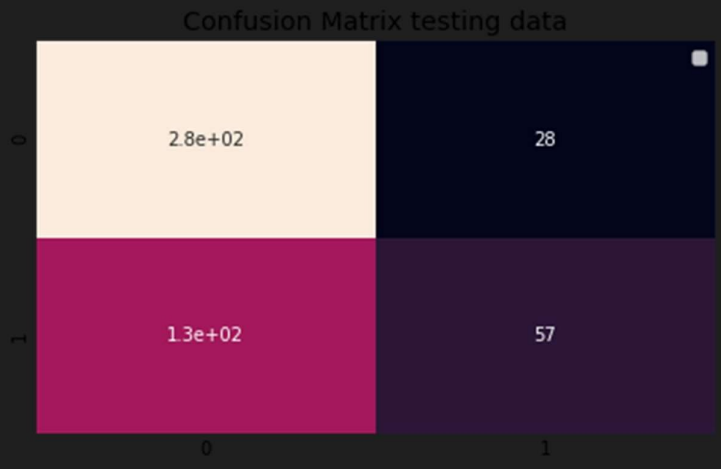
```
0.5198372329603256
```

```
pred1=knn.predict(np.array(X_test))
plt.title("Confusion Matrix testing data")
sns.heatmap(confusion_matrix(Y_test,pred1),annot=True,cbar=False)
plt.legend()
plt.show()
```



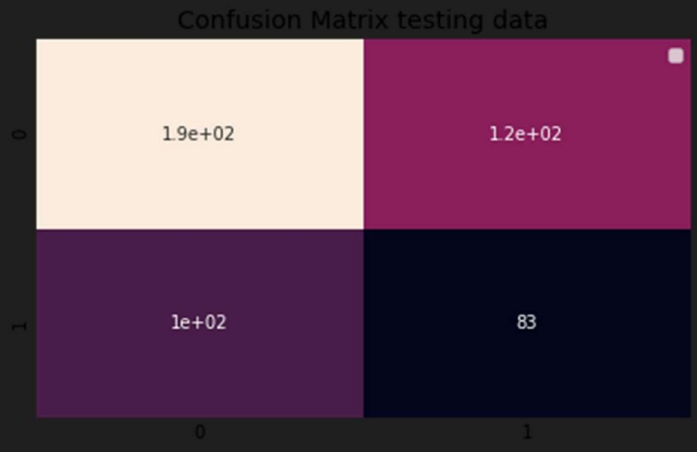
SVM:

```
from sklearn import svm
from sklearn.metrics import accuracy_score
svmc = svm.SVC()
svmc.fit(x_train,  y_train)

y_pred = svmc.predict(x_test)
print(accuracy_score(y_test,y_pred))

0.6808943089430894

#  Confusion matrix
from sklearn.metrics import confusion_matrix
# Make Predictions
pred1=svmc.predict(np.array(x_test))
plt.title("Confusion Matrix testing data")
sns.heatmap(confusion_matrix(y_test,pred1),annot=True,cbar=False)
plt.legend()
plt.show()
```



Decision Tree:

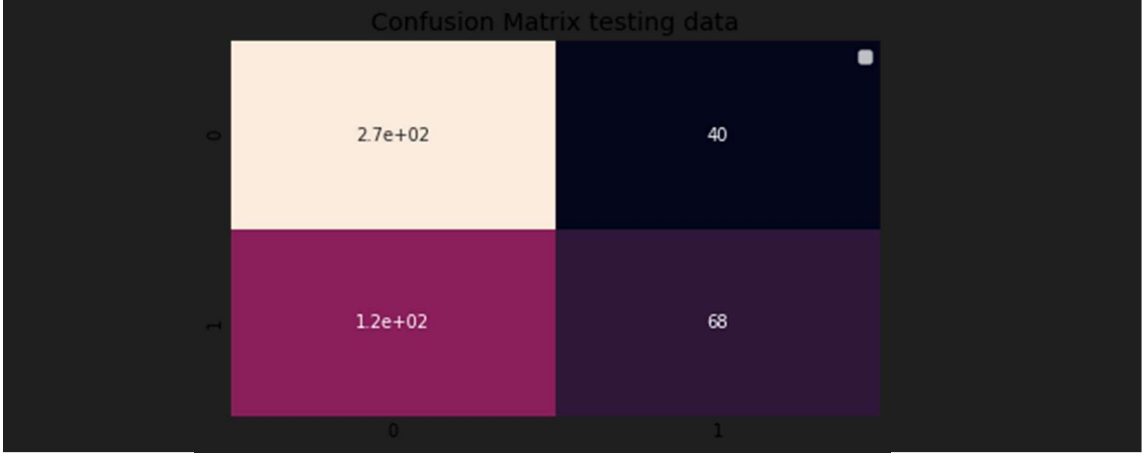
```
from sklearn import tree
from sklearn.metrics import accuracy_score

tre = tree.DecisionTreeClassifier()
tre = tre.fit(x_train, y_train)

y_pred = tre.predict(x_test)
print(accuracy_score(y_test,y_pred))

0.5487804878048781

#  Confusion matrix
from sklearn.metrics import confusion_matrix
# Make Predictions
pred1=tre.predict(np.array(x_test))
plt.title("Confusion Matrix testing data")
sns.heatmap(confusion_matrix(y_test,pred1),annot=True,cbar=False)
plt.legend()
plt.show()
```



Random Forest:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
# create the model
model_rf = RandomForestClassifier(n_estimators=500, oob_score=True,
random_state=100)

# fitting the model
model_rf=model_rf.fit(x_train, y_train)

y_pred = model_rf.predict(x_test)
print(accuracy_score(y_test,y_pred))

0.6788617886178862
```

```
#  Confusion matrix
from sklearn.metrics import confusion_matrix
# Make Predictions
pred1=model_rf.predict(np.array(x_test))
plt.title("Confusion Matrix testing data")
sns.heatmap(confusion_matrix(y_test,pred1),annot=True,cbar=False)
plt.legend()
plt.show()
```



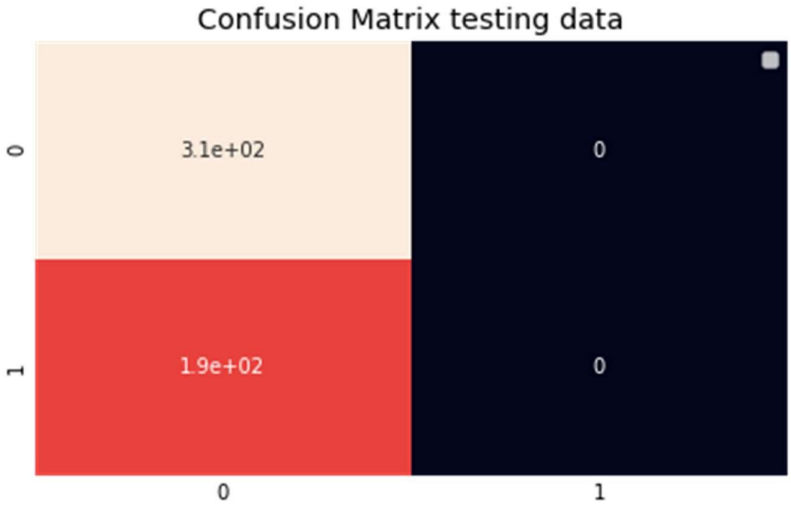
8) Methodology:

Logistic Regression Algorithm:

We used logistic regression algorithm for the training data. Here X_train represents and Y_train represents the corresponding target Vector. The Logistic Regression object is created with random_state = 0 for reproducibility and then fitted to the training data and calculated its accuracy score which is

Accuracy: 0.6052899287894201

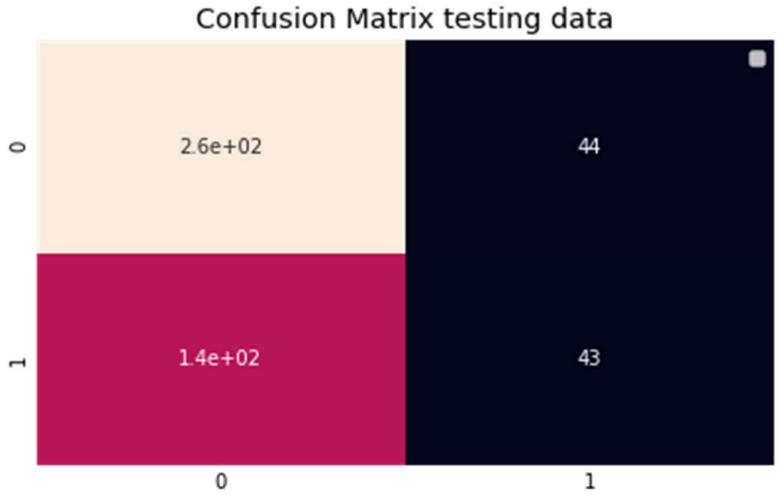
Creating a confusion Matrix for evaluating a classifier's performance in a binary classification problem. A heatmap is used to visualize the confusion matrix.



K Nearest Neighbour algorithm: First, we initialized K neighbour Classifier. The n_neighbours parameter is set to 3, indicating that the algorithm will consider the 3 nearest neighbours for making predictions and we used X_train and Y_train as the input features and the corresponding target labels. We trained KNN classifier and predicted labels on test data and compared these predicted labels with actual labels(Y_test) to get the accuracy which is

Accuracy: **0.5198372329603256**

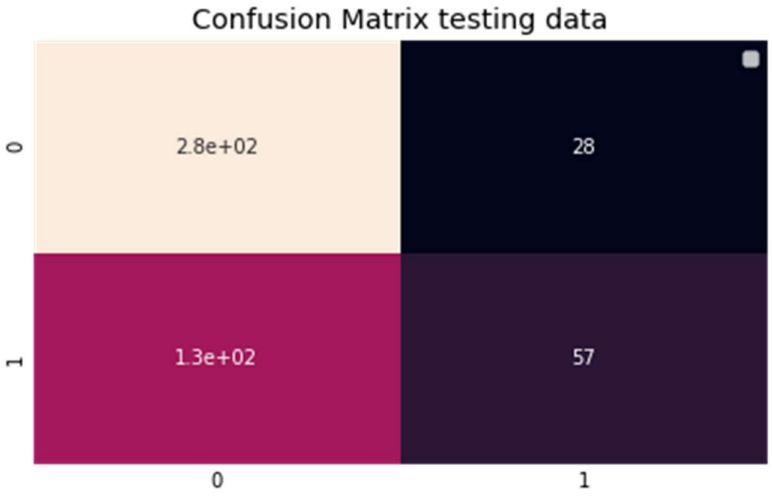
We used KNN to classify the data into different classes based on the input features. It then evaluates the model's performance on the test data by calculating the accuracy and visualizing the confusion matrix to understand how well the model is performing on different classes.



SVM:

By this algorithm we created an SVM classifier from Scikit-learn and initialized without any hyperparameters, we used training data (‘x_train and ‘y_train) to train the classifier. We used the trained SVM classifier to make prediction on the test data(‘x_test’) and then it’s accuracy is calculated by comparing predicted labels(‘y_pred’) with true labels of the test data(‘y_test’). After doing multiple predictions on the test data our code generates a confusion Matrix and is displayed as a heatmap.

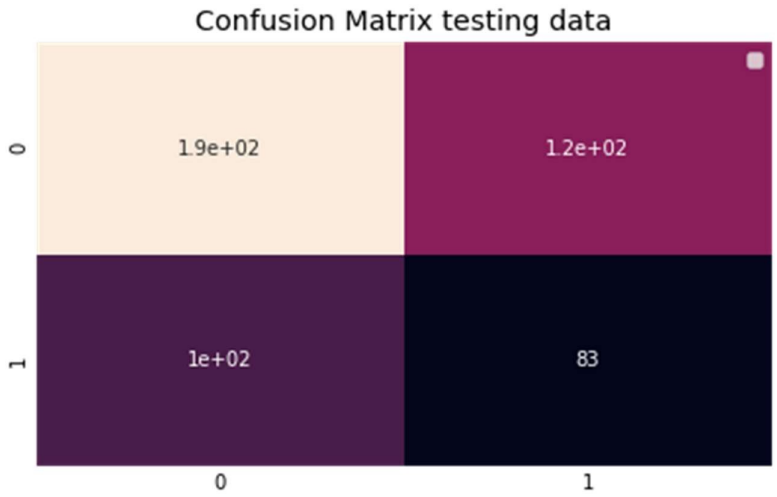
Accuracy: 0.6808943089430894



Decision Tree:

By using this algorithm we created a Decision Tree classifier and we instantiated and assigned to the variable 'tre', and then trained on the training data ('x_train' and 'y_train'). We used the trained Decision Tree classifier to make prediction on the test data('x_test') and then it's accuracy is calculated by comparing predicted labels('y_pred') with true labels of the test data('y_test'). After doing multiple predictions on the test data our code generates a confusion Matrix and is displayed as a heatmap.

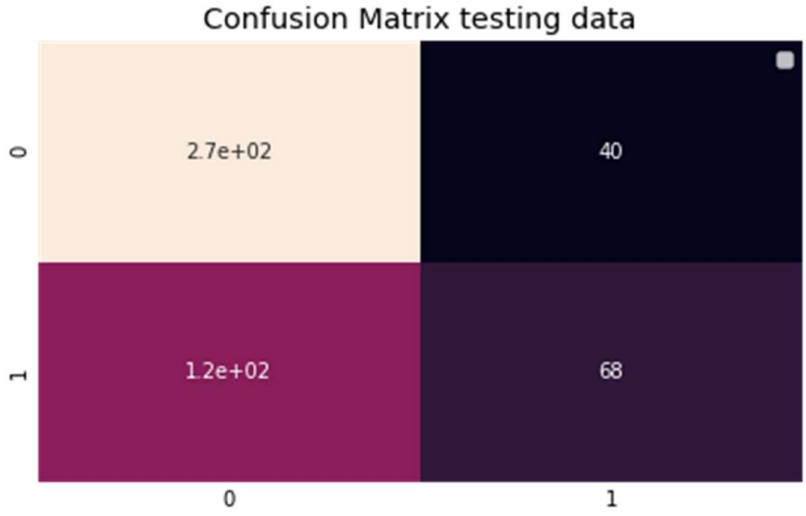
Accuracy: 0.5487804878048781



Random Forest:

First, we created a Random Forest classifier with 500 decision trees and we calculated out-of-bag score during training for validation with the training data ('x_train, 'y_train), where 'x_train' represents the input features and 'y_train' represents the corresponding target variables. After training the model we made our model to predict the target variables ('y_pred') on the test data('x_test') and calculated its accuracy

Accuracy: 0.6788617886178862



9) Differences in Methodology:

After analysing the accuracy results and the differences in methodology for the five machine learning models: Logistic Regression, K-Nearest Neighbours (KNN), Support Vector Machine (SVM), Decision Tree, and the Random Forest. We will present the accuracy scores and highlight the key differences in a tabular form:

Sl. No	Model	Accuracy	Key Differences in Methodology
1.	Logistic Regression	0.6219	Logistic regression, a linear model Used for bina ry classification
2.	K-Nearest Neighbours	0.5198	KNN, a instance based non-linear model
3.	Support Vector Machine (SVM)	0.6809	SVM, a hyperplane-based model
4.	Decision Tree	0.5488	Decision tree, a tree-based model
5.	Random Forest	0.6789	Random Forest, an ensemble of decision trees

Based on the accuracy scores, the **Support Vector Machine (SVM) model achieved the highest accuracy (0.6809)** on the test data. Here's a brief explanation of why SVM might have performed better:

Logistic Regression: Logistic regression is a linear model that might struggle if the relationship between features and the target is non-linear. It's also more sensitive to the scaling and distribution of features.

K-Nearest Neighbours: The accuracy for KNN is not given in the provided code, so it's not possible to directly compare it. KNN's performance can be sensitive to the choice of the number of neighbours (k) and the distance metric.

Support Vector Machine: SVM is a powerful algorithm that can handle both linear and non-linear relationships by finding an optimal separating hyperplane. SVM can be robust in higher-dimensional spaces and is effective when the data is separable with a clear margin.

Decision Tree: Decision trees are prone to overfitting, especially on complex datasets. It might have performed less well due to its tendency to create deep and complex trees.

Random Forest: Random Forest is an ensemble method that combines multiple decision trees. It generally offers good performance and can handle non-linear relationships effectively. However, in this case, it performed slightly worse than SVM.

It's important to note that the choice of the best model depends on the specific problem, dataset, feature engineering, hyperparameter tuning, and other factors. The accuracy alone might not be sufficient to determine the best model, and further analysis, cross-validation, and tuning are essential to make a robust model selection.

10) References:

- https://ieeexplore.ieee.org/Xplore/home.jsp
- https://ieeexplore.ieee.org/search/searchresult.jsp?newsearch=true&queryText=water%20quality%20classification%20using%20machine%20learning
- https://www.elsevier.com/en-in
- https://www.elsevier.com/en-in/search-results?query=water%20quality%20classification%20using%20machine%20learning
- https://mjl.clarivate.com/search-results
- https://www.kaggle.com/datasets/adityakadiwal/water-potability