AMERICAN INTERNATIONAL UNIVERSITY- BANGLADESH
(AIUB)
Faculty of Science and Technology
Department of Computer Science and Engineering

FINAL TERM PROJECT REPORT

PROGRAMMING IN PYTHON

Project Name: Water Quality Prediction by using Classification Models.

Submitted to:	Semester: Spring 2022-23
Dr. Akinul Islam Jony Associate Professor, Head UG Faculty of Science & Technology Department of CSE	Section: B Date of Submission: 3/05/2023

Submitted By

No	Name	ID
1	Anika Saba Ibte Sum	20-43242-1
2	Oishi Singh	20-43067-1
3	Mehadi Hasan Shojib	20-42713-1
4	Anik Debnath	20-42780-1

Table of Contents

Section 1: Project Overview

Water quality refers to the characteristics of water that determine its suitability for specific uses, such as drinking, irrigation, industrial processes, and aquatic life. It is determined by the physical, chemical, biological, and radiological properties of water, and the presence or absence of contaminants, such as bacteria, viruses, organic and inorganic compounds, and toxic substances. Access to safe drinking-water is essential to health, a basic human right and a component of effective policy for health protection. This is important as a health and development issue at a national, regional, and local level. In some regions, it has been shown that investments in water supply and sanitation can yield a net economic benefit, since the reductions in adverse health effects and health care costs outweigh the costs of undertaking the interventions.

Recently, machine learning techniques have been used for predicting water quality, as it is more accurate than models based on physical principles. To address various problems, varieties of machine learning algorithms are applied in different fields. In this project, to examine how accurately these models predict water quality. It is carried out to compare these models so that it can be realized which model works better. In this project, a set of the most common machine learning techniques are explored to generate robust water potability model for long periods of time. Moreover, the combinations of all the model parameters are considered for simulations. The experimental results of the classifiers show which classifier model gives better classification accuracy. In this project NumPy, pandas, matplotlib, seaborn, scikit learn libraries are used. Data preprocessing and Exploratory data analysis is presented to justify the project as a more accurate one. After data analysis unnecessary variables were removed. Then the dataset is splitted into training and test set. After that a model is built for each model and test our data. Finally, it is compared to the accuracy score.

Section 2: Dataset Overview

Data Source: This data set is taken from Kaggle.com. The name of the dataset is Water Quality Prediction. The art of water quality prediction has been a difficult task for many of the researchers and analysts. It's very important to predict water for drinking safely. So, we decided to predict water quality using data set which is downloaded from Kaggle to train our model and evaluate by testing using 10 attributes. We are going to predict 2 types of water potability conditions (potable or not potable). This is a copy of Water Quality prediction dataset: Water Quality | Kaggle Dataset Link:

url: https://www.kaggle.com/datasets/adityakadiwal/water-potability

Description of Dataset:

We have used a dataset about weather information from Kaggle to train our model and evaluate by testing.
Number of Instances: 3276
Number of Attributes: 10 numeric predictive.

Attribute Information:

1. pH value: PH is an important parameter in evaluating the acid–base balance of water. It is also an indicator of acidic or alkaline condition of water status. WHO has recommended maximum permissible limit of pH from 6.5 to 8.5. The current investigation ranges were 6.52–6.83 which are in the range of WHO standards.

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. These minerals produced un-wanted taste and diluted color in appearance of water. This is an important parameter for the use of water. The water with high TDS value indicates that water is highly mineralized. The desirable limit for TDS is 500 mg/l and maximum limit is 1000 mg/l which prescribed for drinking purposes.

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. Chlorine levels up to 4 milligrams per liter (mg/L or 4 parts per million (ppm)) are considered safe in drinking water.

5. Sulfate: 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. Sulfate concentration in seawater is about 2,700 milligrams per liter (mg/L). It ranges from 3 to 30 mg/L in most freshwater supplies, although much higher concentrations (1000 mg/L) are found in some geographic locations.

6. Conductivity: Pure water is not a good conductor of electric current rather's a good insulator. Increase in ions concentration enhances the electrical conductivity of water. Generally, the number of dissolved solids in water determines the electrical conductivity. Electrical conductivity (EC) actually measures the ionic process of a solution that enables it to transmit current. According to WHO standards, EC value should not exceed 400 μS/cm.

7. Organic_carbon: Total Organic Carbon (TOC) in source waters comes from decaying natural organic matter (NOM) as well as synthetic sources. TOC is a measure of the total amount of carbon in organic compounds in pure water. According to US EPA < 2 mg/L as TOC in treated / drinking water, and < 4 mg/Lit in source water which is use for treatment.

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. THM levels up to 80 ppm are considered safe in drinking water.

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 and the test is used to indicate the quality of waste discharge with respect to colloidal matter. The mean turbidity value obtained for Wondo Genet Campus (0.98 NTU) is lower than the WHO recommended value of 5.00 NTU.

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

Section 3: Data Preprocessing and Exploratory Data Analysis

Load dataset:

We upload the dataset into google colab and then we load the dataset from the local drive.

```
from google.colab import files

uploaded = files.upload()
```

```
Choose Files water_potability.csv
• water_potability.csv(text/csv) - 525187 bytes, last modified: 4/14/2023 - 100% done
Saving water_potability.csv to water_potability.csv
```

```
[5] # importing necessary libaries
import pandas as pd
import io
#importing dataset from a csv file
dataset = pd.read_csv(io.BytesIO(uploaded['water_potability.csv']))
dataset
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	NaN	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	NaN	592.885359	15.180013	56.329076	4.500656	0
2	8.099124	224.236259	19909.541732	9.275884	NaN	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
...	...	...	...	...	...	...	...	...	...	...
3271	4.668102	193.681735	47580.991603	7.166639	359.948574	526.424171	13.894419	66.687695	4.435821	1
3272	7.808856	193.553212	17329.802160	8.061362	NaN	392.449580	19.903225	NaN	2.798243	1
3273	9.419510	175.762646	33155.578218	7.350233	NaN	432.044783	11.039070	69.845400	3.298875	1
3274	5.126763	230.603758	11983.869376	6.303357	NaN	402.883113	11.168946	77.488213	4.708658	1
3275	7.874671	195.102299	17404.177061	7.509306	NaN	327.459760	16.140368	78.698446	2.309149	1

3276 rows × 10 columns

We check the over all information like how many rows and columns, data type, memory usage and so on.

```
#information about our data
dataset.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
```

Data Preprocessing:

Here, we can see that data is missing. For column ph, sulfate and trihalomethanes, they hold missing values.

```
#Checking 0 value arries or not for the all attributes
dataset.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
```

We are calculating mean values for ph, sulfate, trhalomethanes.

```
cols = ['ph', 'Sulfate', 'Trihalomethanes']
dataset[cols].mean()
```

```
ph                    7.080795
Sulfate             333.775777
Trihalomethanes      66.396293
dtype: float64
```

We are replaceing the null values by using the mean values for three columns.

```
dataset['ph'].fillna((dataset['ph'].mean()), inplace=True)
dataset['Sulfate'].fillna((dataset['Sulfate'].mean()), inplace=True)
dataset['Trihalomethanes'].fillna((dataset['Trihalomethanes'].mean()), inplace=True)
dataset
```

	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
...	...	...	...	...	...	...	...	...	...	...
3271	4.668102	193.681735	47580.991603	7.166639	359.948574	526.424171	13.894419	66.687695	4.435821	1
3272	7.808856	193.553212	17329.802160	8.061362	333.775777	392.449580	19.903225	66.396293	2.798243	1
3273	9.419510	175.762646	33155.578218	7.350233	333.775777	432.044783	11.039070	69.845400	3.298875	1
3274	5.126763	230.603758	11983.869376	6.303357	333.775777	402.883113	11.168946	77.488213	4.708658	1
3275	7.874671	195.102299	17404.177061	7.509306	333.775777	327.459760	16.140368	78.698446	2.309149	1

3276 rows × 10 columns

After replacing the missing values, we check again that thre is any null values are here.

```
dataset.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
```

Then, we describe the full dataset where shows that the count, mean, standard deviation, minimum, quartile values and maximum values.

```
dataset.describe()
```

	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

Data type conversion which is numeric value convert into the catagorical value.

```
[28] dataset.dtypes
     dataset['Potability'] = dataset['Potability'].astype('category')
```

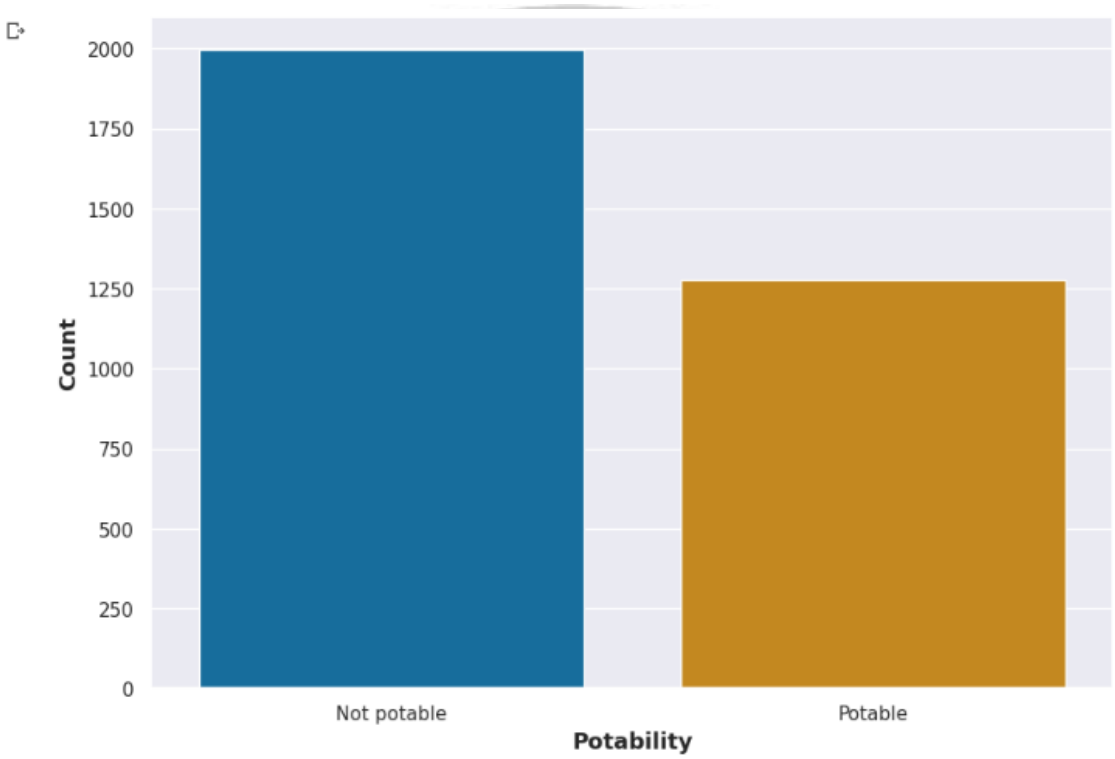
```
[47] import pandas as pd
     import numpy as np
     dataset.loc[:, 'Potability'] = dataset['Potability'].map({0: 'Not potable', 1: 'Potable'})
     dataset
```

	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	Not potable
1	3.716080	129.422921	18630.057858	6.635246	333.775777	592.885359	15.180013	56.329076	4.500656	Not potable
2	8.099124	224.236259	19909.541732	9.275884	333.775777	418.606213	16.868637	66.420093	3.055934	Not potable
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.341674	4.628771	Not potable
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	Not potable
...	...	...	...	...	...	...	...	...	...	...
3271	4.668102	193.681735	47580.991603	7.166639	359.948574	526.424171	13.894419	66.687695	4.435821	Potable
3272	7.808856	193.553212	17329.802160	8.061362	333.775777	392.449580	19.903225	66.396293	2.798243	Potable
3273	9.419510	175.762646	33155.578218	7.350233	333.775777	432.044783	11.039070	69.845400	3.298875	Potable
3274	5.126763	230.603758	11983.869376	6.303357	333.775777	402.883113	11.168946	77.488213	4.708658	Potable
3275	7.874671	195.102299	17404.177061	7.509306	333.775777	327.459760	16.140368	78.698446	2.309149	Potable

3276 rows × 10 columns

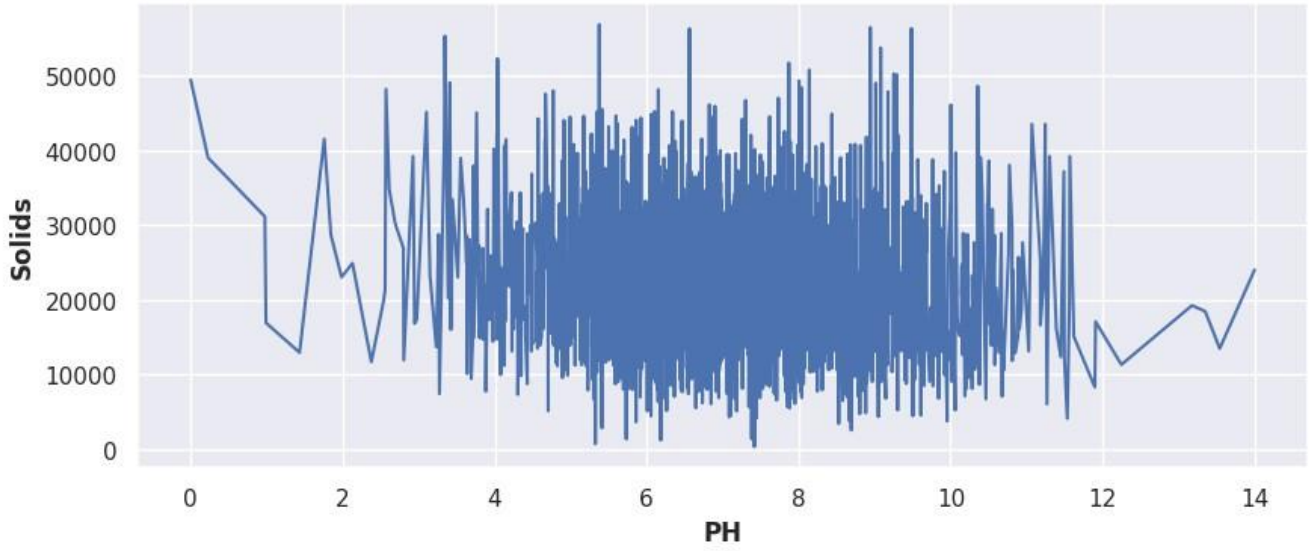
Exploratory Data Analysis:

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10,7))
sns.set_theme()
sns.countplot(x = 'Potability',data = dataset,palette="colorblind")
plt.xlabel("Potability",fontweight='bold',size=13)
plt.ylabel("Count",fontweight='bold',size=13)
plt.show()
```



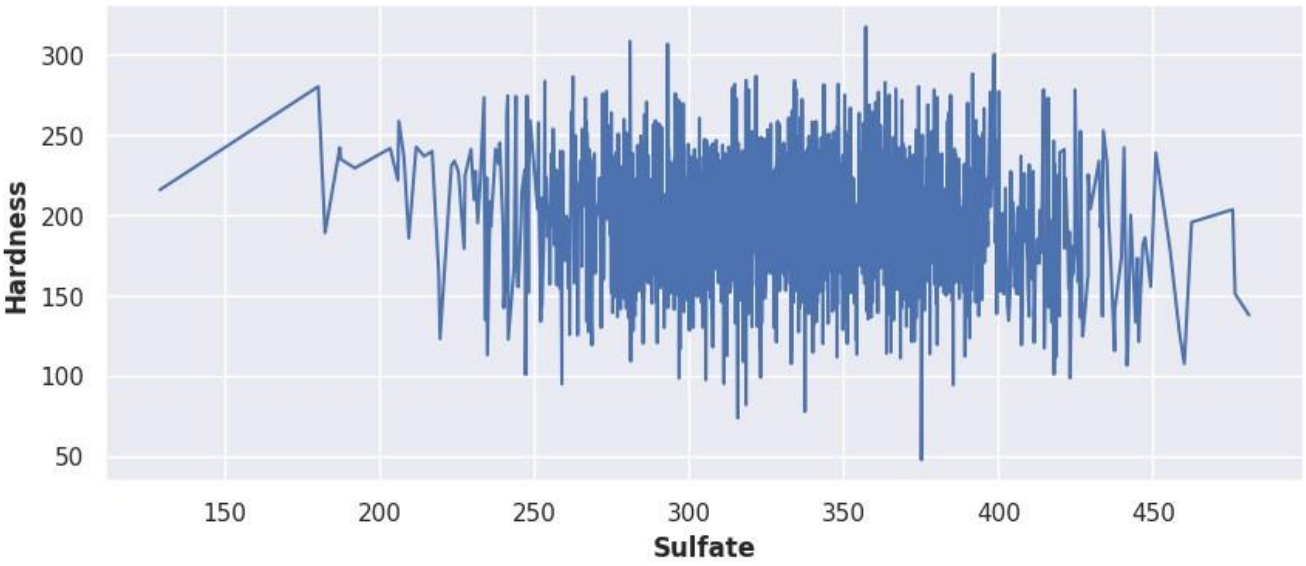
Here we count the value of target attributes which is portability. We can see that most of the value is not potable value above 1500.

```
plt.figure(figsize=(10,4))
sns.set_theme()
sns.lineplot(x = 'ph',y='Solids',data=dataset)
plt.xlabel("PH",fontweight='bold',size=12)
plt.ylabel("Solids",fontweight='bold',size=12)
plt.show()
```



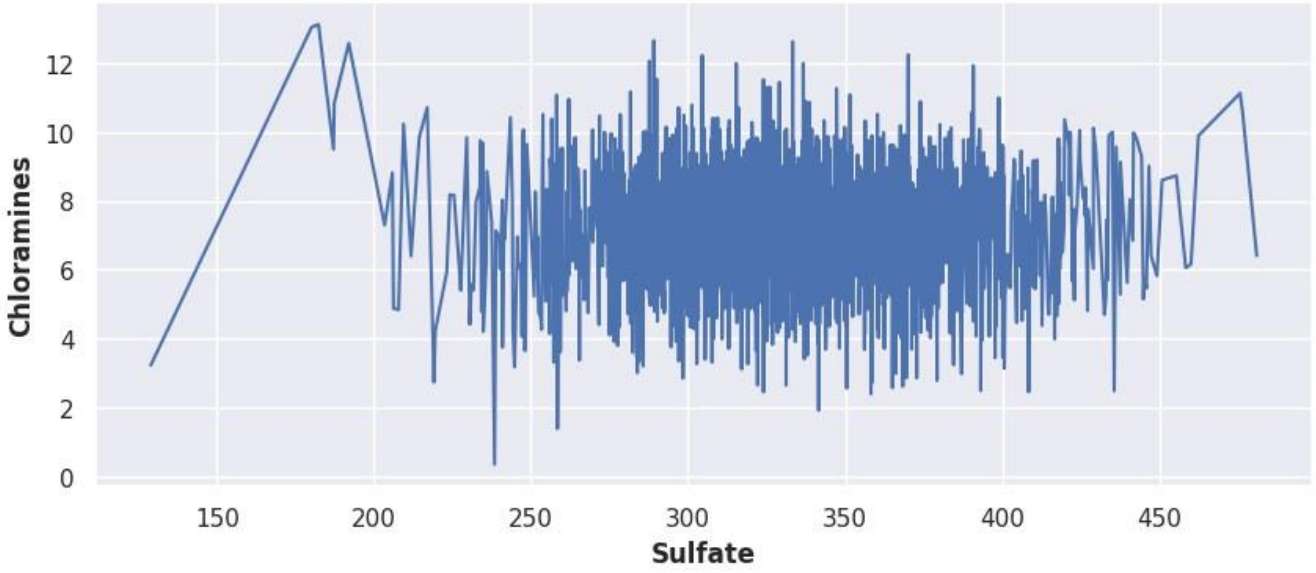
Here the explanation for the relation between pH and Solids is that higher levels of solids in the water may affect the pH level. Solids in water can include dissolved minerals and organic matter, and can impact the water's pH level by affecting the buffering capacity of the water. Therefore, it is possible that higher levels of solids in the water may lead to higher or lower pH levels.

```
plt.figure(figsize=(10,4))
sns.set_theme()
sns.lineplot(x = 'Sulfate',y='Hardness',data=dataset)
plt.xlabel("Sulfate",fontweight='bold',size=12)
plt.ylabel("Hardness",fontweight='bold',size=12)
plt.show()
```



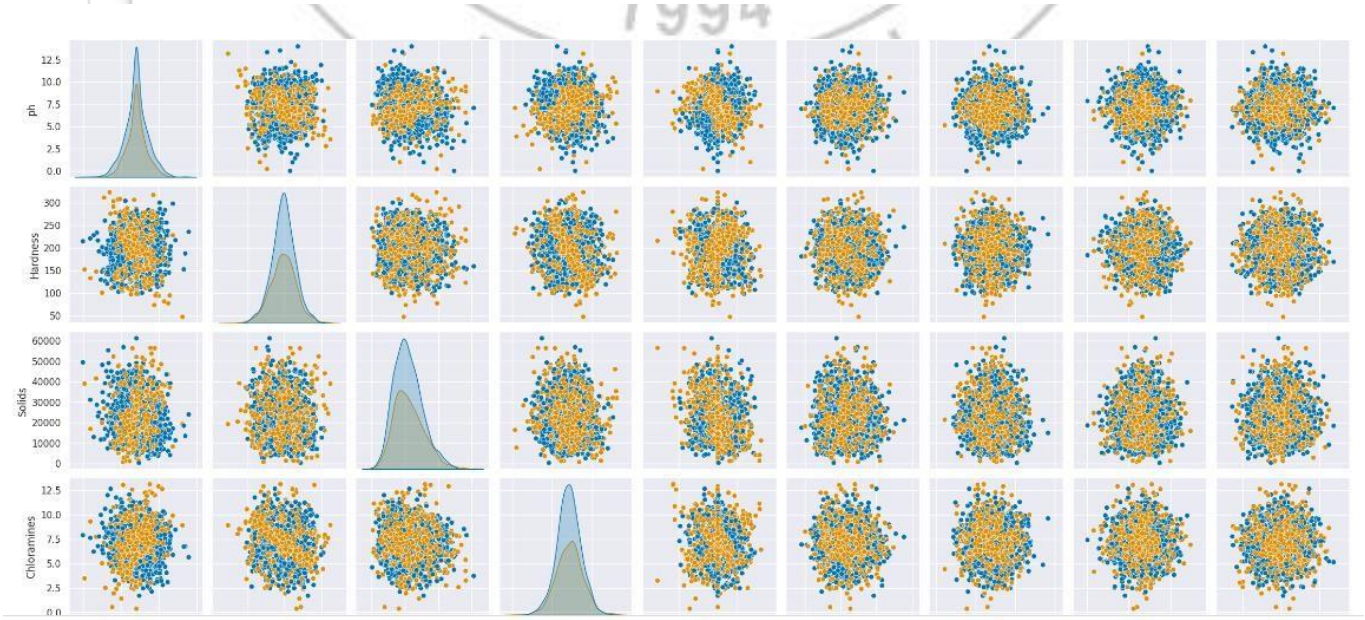
Here we can see that Sulfate and Hardness is that both attributes may be influenced by the geological characteristics of the water source. Hardness is a measure of the concentration of dissolved minerals in the water, particularly calcium and magnesium. Sulfate is also a dissolved mineral that can be present in water sources, and its concentration can be affected by the same geological factors that influence water hardness.

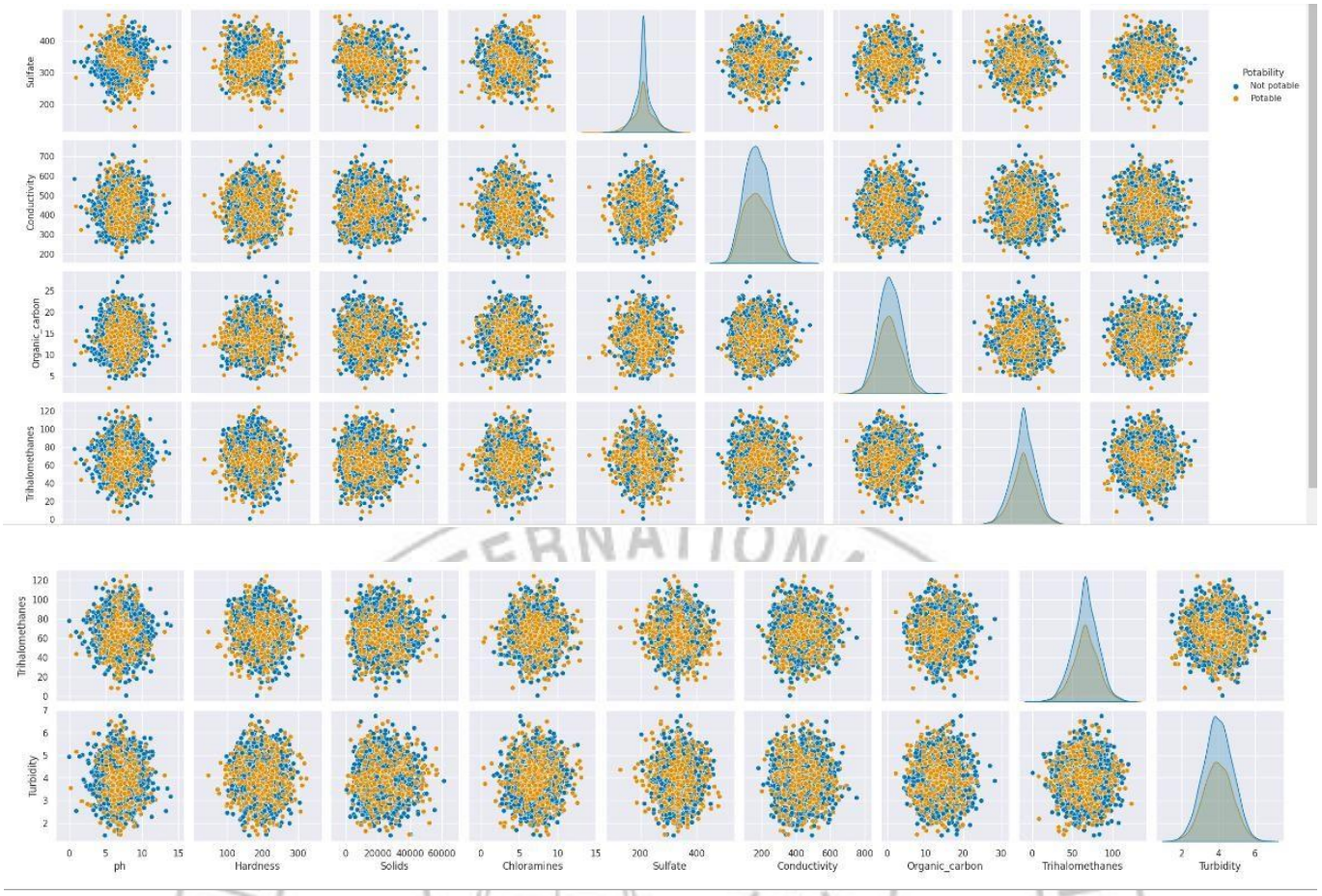
```
plt.figure(figsize=(10,4))
sns.set_theme()
sns.lineplot(x = 'Sulfate',y='Chloramines',data=dataset)
plt.xlabel("Sulfate",fontweight='bold',size=12)
plt.ylabel("Chloramines",fontweight='bold',size=12)
plt.show()
```



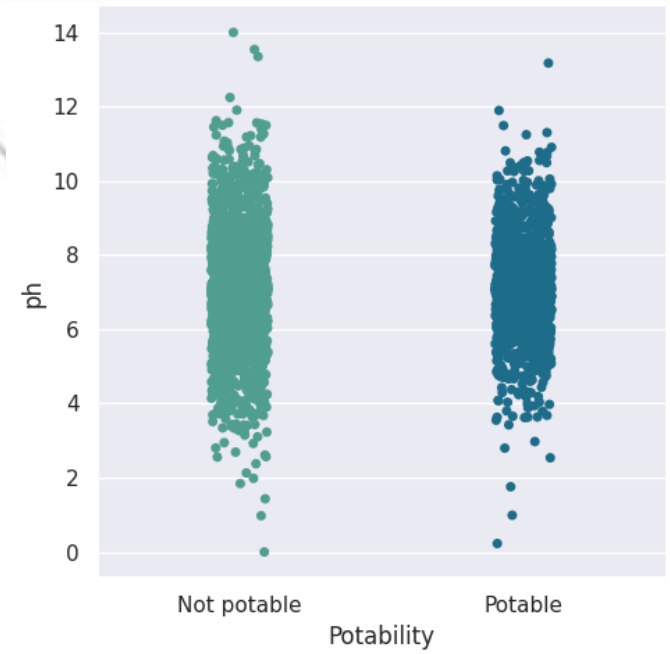
Here, Sulfate and Chloramines is that the levels of these two chemicals are related to the source of the water. Both sulfate and chloramines can occur naturally in water sources, and the concentrations of these chemicals can be influenced by a range of factors, such as the geology of the area, the level of urbanization, and the sources of pollution.

```
plt.figure(figsize=(14,8))
sns.pairplot(dataset,hue='Potability',palette="colorblind")
plt.show()
```

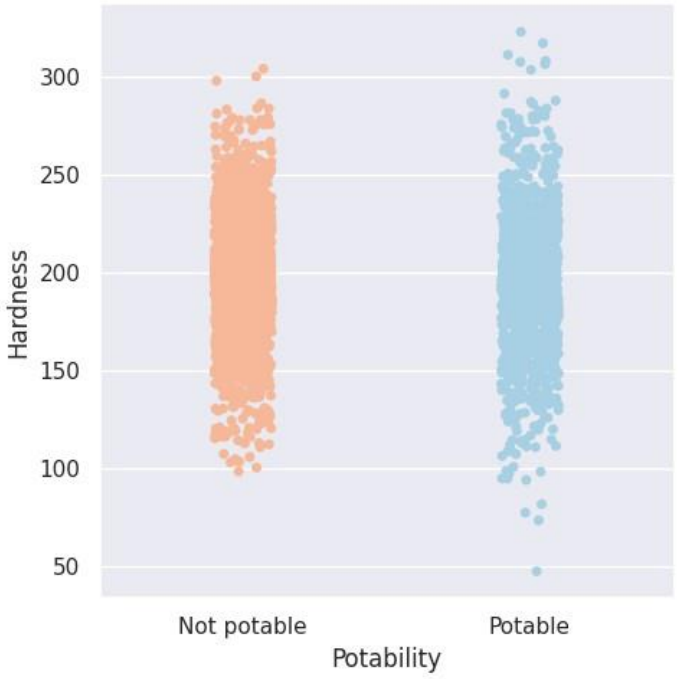




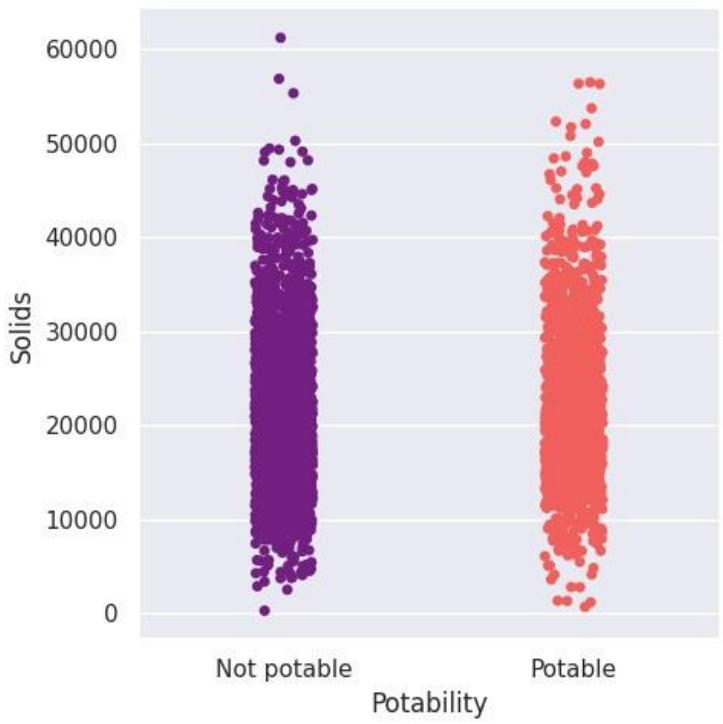

```
plt.figure(figsize=(10,5))
sns.catplot(x='Potability',y ='ph',data=dataset,palette="crest")
plt.show()
```



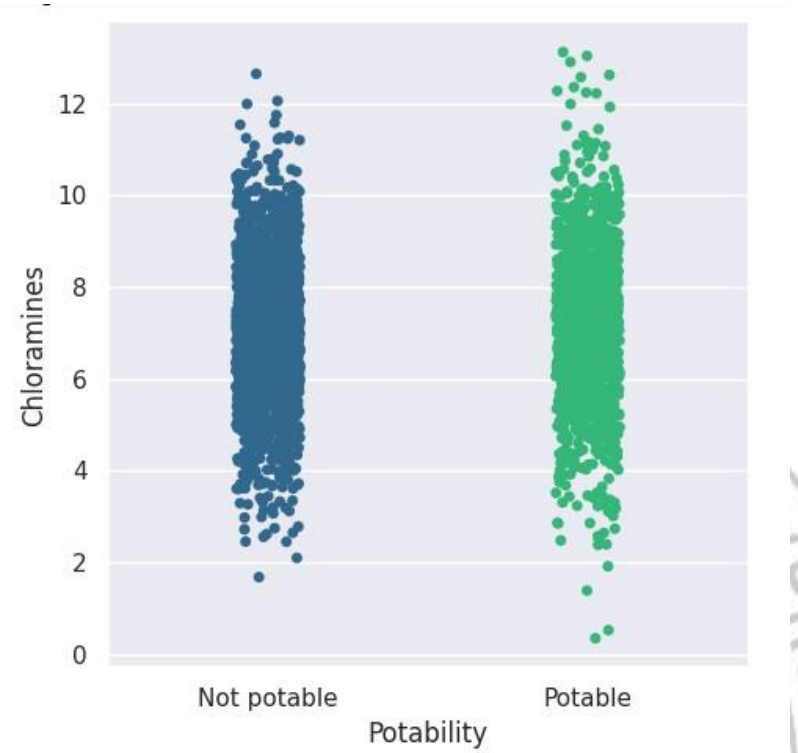
```
plt.figure(figsize=(10,5))
sns.catplot(x='Potability',y ='Hardness',data=dataset,palette = "RdBu")
plt.show()
```



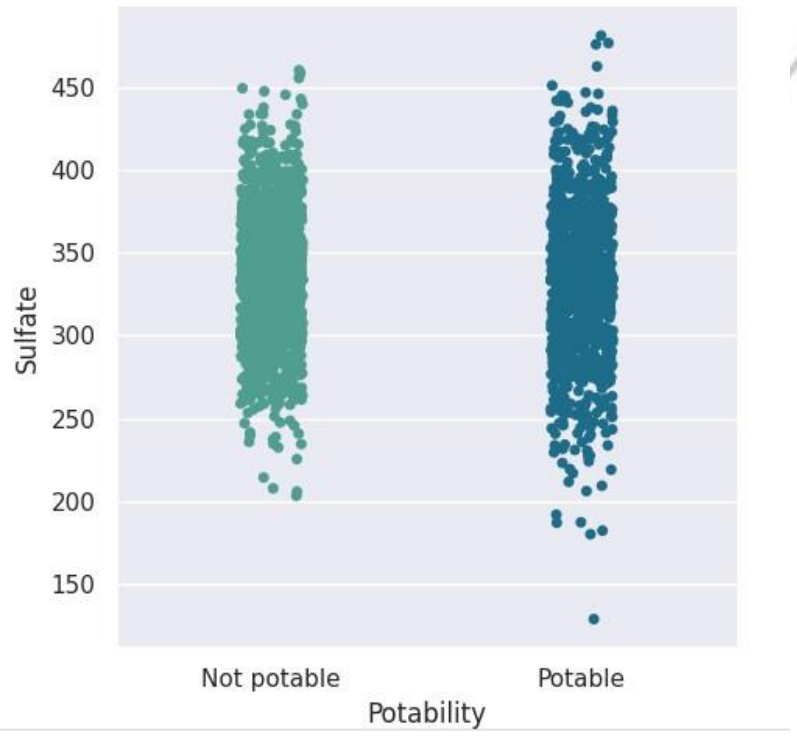

```
plt.figure(figsize=(10,5))
sns.catplot(x='Potability',y ='Solids',data=dataset,palette = "magma")
plt.show()
```



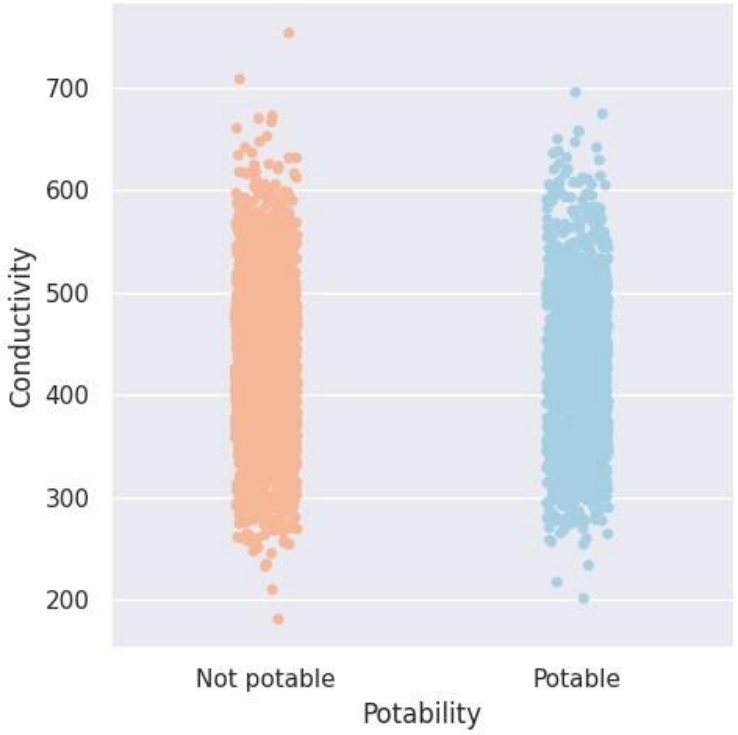
```
plt.figure(figsize=(10,5))
sns.catplot(x='Potability',y ='Chloramines',data=dataset,palette = "viridis")
plt.show()
```



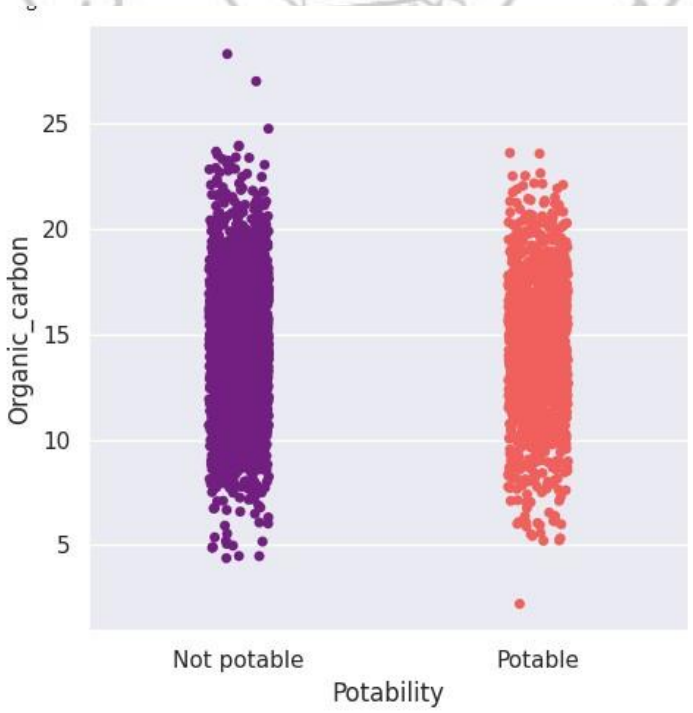

```
plt.figure(figsize=(10,5))
sns.catplot(x='Potability',y ='Sulfate',data=dataset,palette="crest")
plt.show()
```



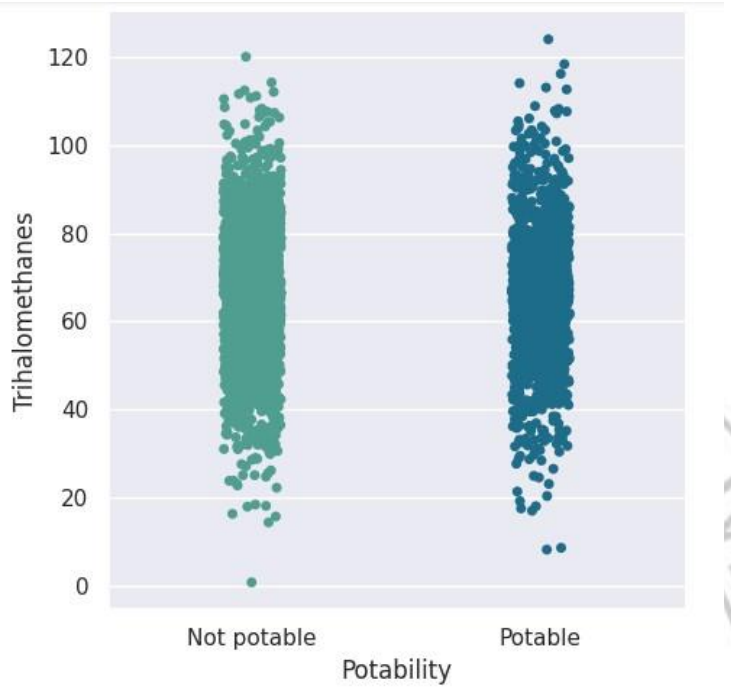
```
plt.figure(figsize=(10,5))
sns.catplot(x='Potability',y ='Conductivity',data=dataset,palette = "RdBu")
plt.show()
```



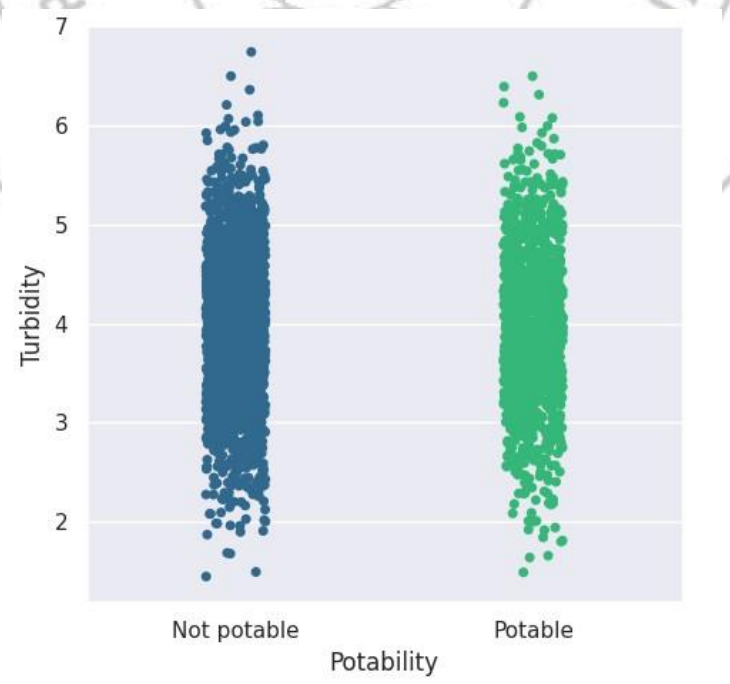

```
plt.figure(figsize=(10,5))
sns.catplot(x='Potability',y ='Organic_carbon',data=dataset,palette = "magma")
plt.show()
```



```
plt.figure(figsize=(10,5))
sns.catplot(x='Potability',y ='Conductivity',data=dataset,palette = "RdBu")
plt.show()
```

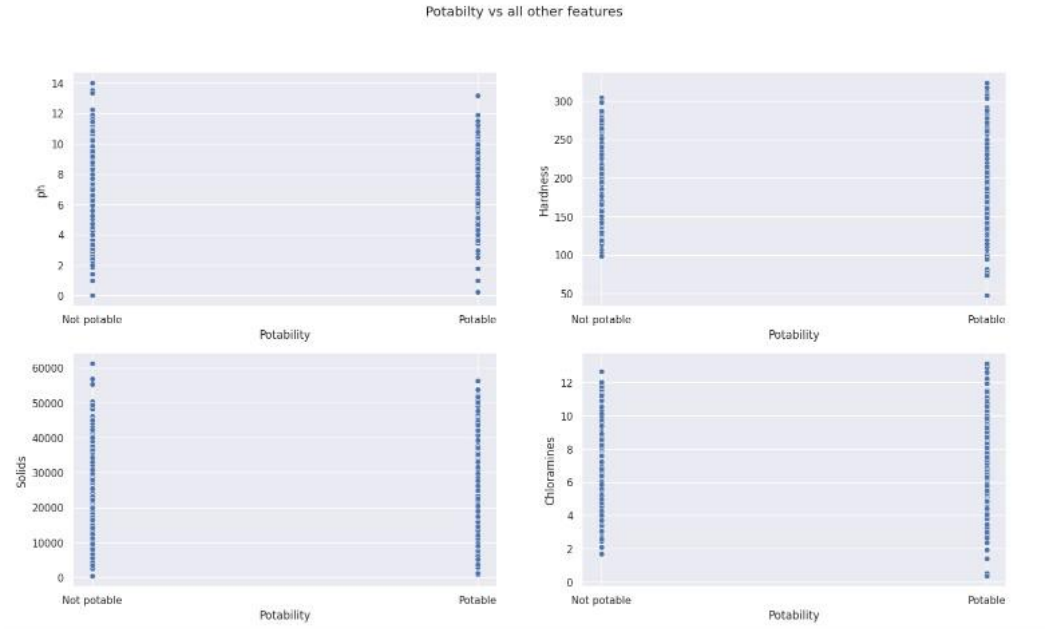



```
plt.figure(figsize=(10,5))
sns.catplot(x='Potability',y ='Turbidity',data=dataset,palette = "viridis")
plt.show()
```



From the catplot we can identify how much potability changes for a particular feature of the dataset.

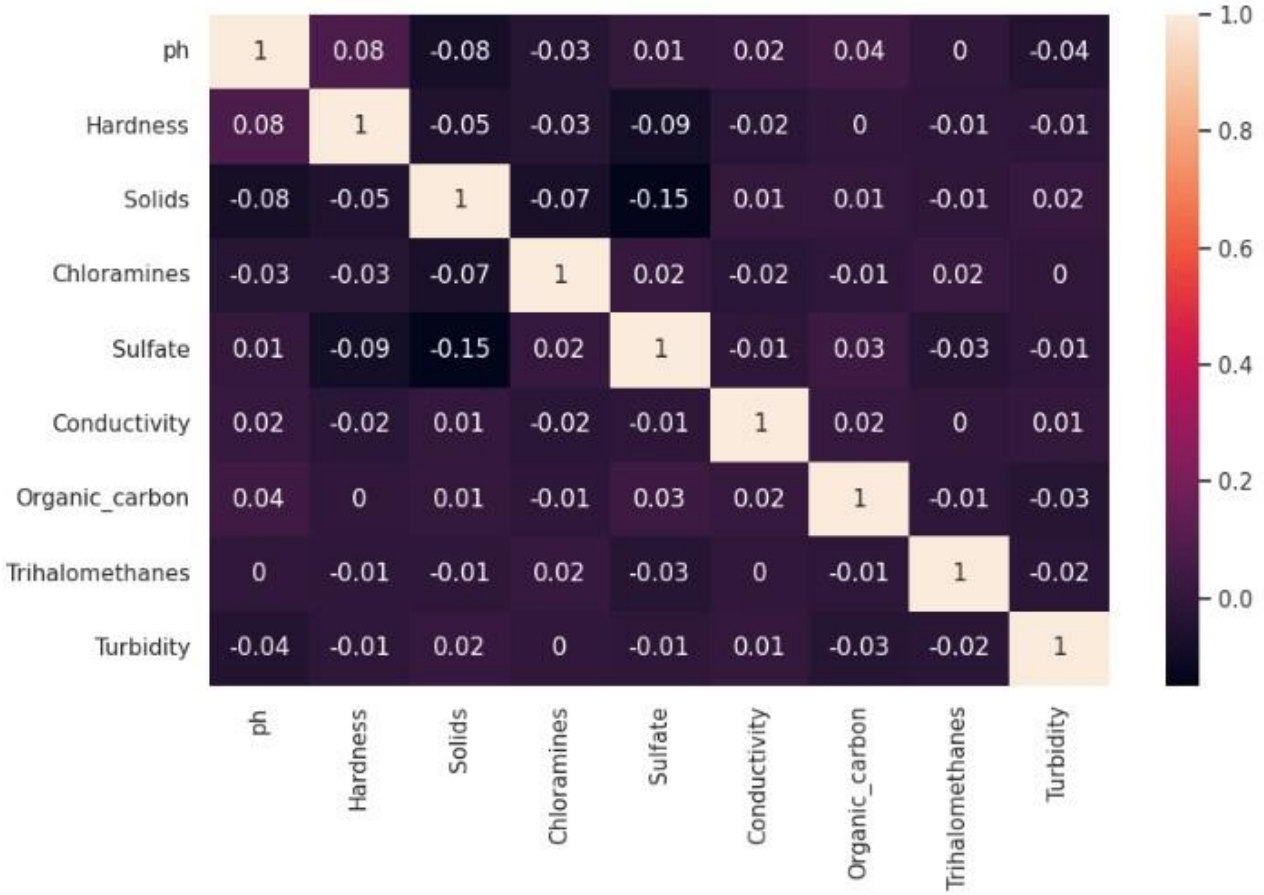
```
fig, axes = plt.subplots(2, 2, figsize=(18, 10))
fig.suptitle('Potabilty vs all other features')
sns.scatterplot(ax=axes[0, 0], data=dataset, x='Potability', y='ph')
sns.scatterplot(ax=axes[0, 1], data=dataset, x='Potability', y='Hardness')
sns.scatterplot(ax=axes[1, 0], data=dataset, x='Potability', y='Solids')
sns.scatterplot(ax=axes[1, 1], data=dataset, x='Potability', y='Chloramines')
plt.show()
```



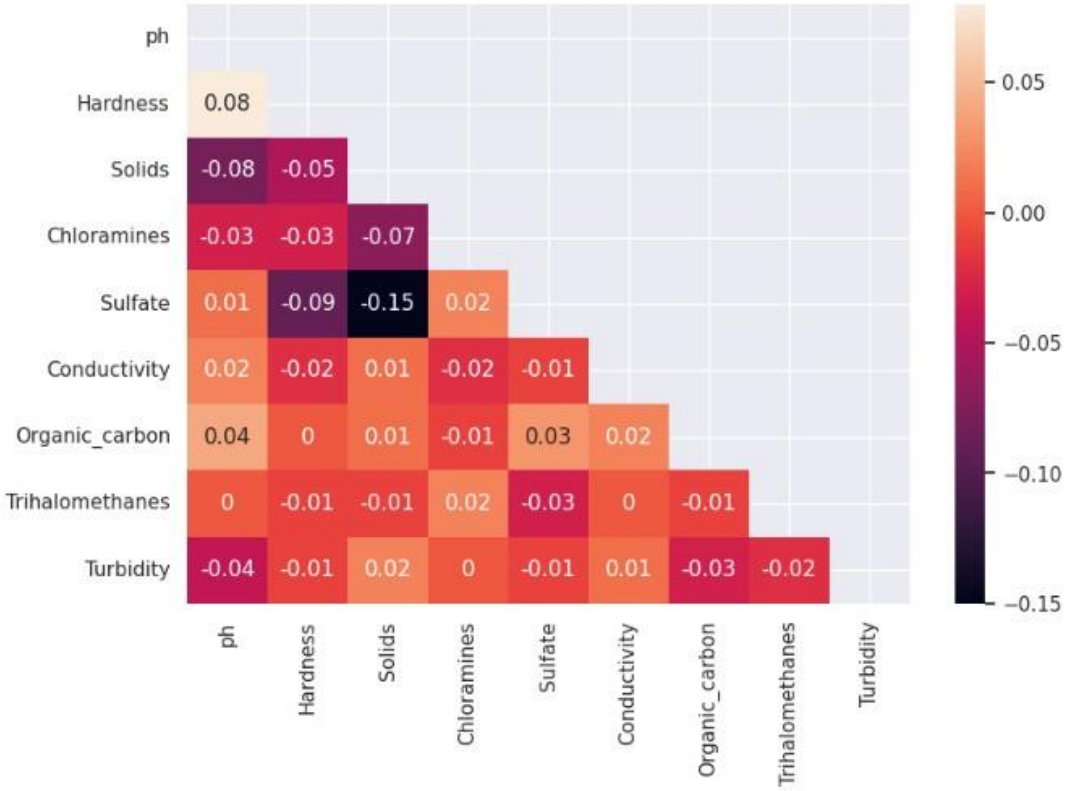

```
fig, axes = plt.subplots(2, 2, figsize=(18, 10))
fig.suptitle('Potabilty vs all other features')
sns.scatterplot(ax=axes[0, 0], data=dataset, x='Potability', y='Sulfate')
sns.scatterplot(ax=axes[0, 1], data=dataset, x='Potability', y='Conductivity')
sns.scatterplot(ax=axes[1, 0], data=dataset, x='Potability', y='Organic_carbon')
sns.scatterplot(ax=axes[1, 1], data=dataset, x='Potability', y='Trihalomethanes')
sns.scatterplot(ax=axes[0, 0], data=dataset, x='Potability', y='Turbidity')
```

From the scatter plot we can identify how much water potability prediction changes for a particular feature of the dataset.

```
correlation_matrix = dataset.corr().round(2)
plt.figure(figsize = (10, 6))
sns.heatmap(data=correlation_matrix, annot=True);
```




```
mask = np.zeros_like(correlation_matrix)
mask[np.triu_indices_from(mask)] = True
plt.figure(figsize = (9, 6))
sns.heatmap(data=correlation_matrix, annot=True, mask=mask)
```



To predict water potability, it is important to select appropriate features that have a strong relationship with the target variable and can help the machine learning model make accurate predictions. Based on the correlation matrix for the water potability dataset, there are no correlated columns in the data.

Create Features Matrix & Target Variable

```
x = dataset.drop(columns=['Potability'])
x
```

	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

```
y = dataset['Potability']
y
```

```
y = dataset['Potability']
y
```

```
0       Not potable
1       Not potable
2       Not potable
3       Not potable
4       Not potable
           ...
3271        Potable
3272        Potable
3273        Potable
3274        Potable
3275        Potable
Name: Potability, Length: 3276, dtype: category
Categories (2, object): ['Not potable', 'Potable']
```

Split the dataset.

```
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 = 1)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(2620, 9)
(656, 9)
(2620,)
(656,)
```

We have split our data set. We take 20% of data for test dataset and 80% of data for training dataset.

Section 4: Model Development

1. Gaussian Naive Bayes (NB)

```
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics # for checking the model accuracy
model_nb = GaussianNB()
model_nb.fit(X_train, y_train) #train the model with the training dataset
y_prediction_nb = model_nb.predict(X_test) #pass the testing data to the trained mo
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_nb = metrics.accuracy_score(y_prediction_nb, y_test).round(4)
print("---------------------------------")
print('The accuracy of the NB is: {}'.format(score_nb))
print("---------------------------------")
# save the accuracy score
score = []
score.append( score_nb)
```

```
---------------------------------
The accuracy of the NB is: 0.6006
---------------------------------
```

2. K Nearest Neighbors (KNN)

```
from sklearn.neighbors import KNeighborsClassifier # for K nearest neighbours
#from sklearn.linear_model import LogisticRegression # for Logistic Regression algo
model_knn = KNeighborsClassifier(n_neighbors=3) # 3 neighbours for putting the new
model_knn.fit(X_train, y_train) #train the model with the training dataset
y_prediction_knn = model_knn.predict(X_test) #pass the testing data to the trained
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_knn = metrics.accuracy_score(y_prediction_knn, y_test).round(4)
print("---------------------------------")
print('The accuracy of the KNN is: {}'.format(score_knn))
print("---------------------------------")
# save the accuracy score
score.append( score_knn)
```

```
---------------------------------
The accuracy of the KNN is: 0.5518
---------------------------------
```

3. Decision Tree

```
from sklearn.tree import DecisionTreeClassifier #for using Decision Tree Algoithm
model_dt = DecisionTreeClassifier(random_state=4)
model_dt.fit(X_train, y_train) #train the model with the training dataset
y_prediction_dt = model_dt.predict(X_test) #pass the testing data to the trained mo
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_dt = metrics.accuracy_score(y_prediction_dt, y_test).round(4)
print("---------------------------------")
print('The accuracy of the DT is: {}'.format(score_dt))
print("---------------------------------")
# save the accuracy score
score.append(score_dt)
```

```
---------------------------------
The accuracy of the DT is: 0.6021
---------------------------------
```

4. Logistic Regression (LR)

```
from sklearn.linear_model import LogisticRegression # for Logistic Regression algor
model_lr = LogisticRegression(solver='lbfgs', max_iter=500)
model_lr.fit(X_train, y_train) #train the model with the training dataset
y_prediction_lr = model_lr.predict(X_test) #pass the testing data to the trained mo
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_lr = metrics.accuracy_score(y_prediction_lr, y_test).round(4)
print("---------------------------------")
print('The accuracy of the LR is: {}'.format(score_lr))
print("---------------------------------")
# save the accuracy score
score.append(score_lr)
```

```
---------------------------------
The accuracy of the LR is: 0.5686
---------------------------------
```

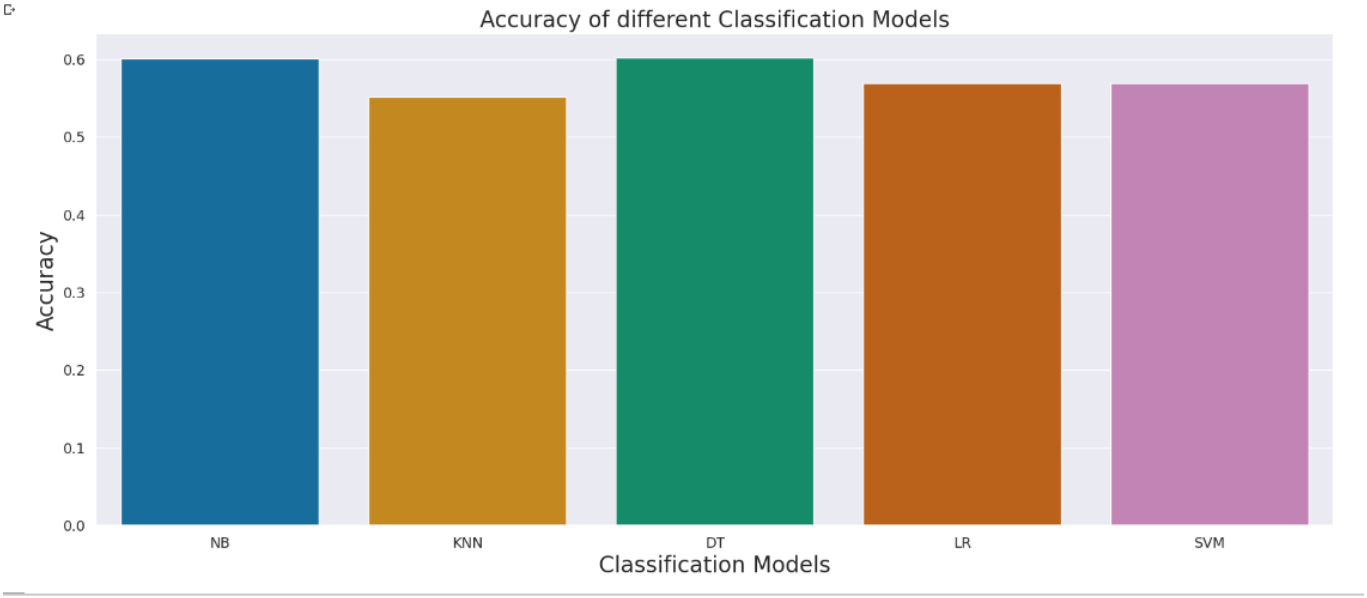
5. Support Vector Machine (SVM)

```
#Training and calculating accuracy for SVM
from sklearn import svm #for Support Vector Machine (SVM) Algorithm
from sklearn import metrics # for checking the model accuracy
model_svm = svm.SVC() #select the algorithm
model_svm.fit(X_train, y_train) #train the model with the training dataset
y_prediction_svm = model_svm.predict(X_test) # pass the testing data to the trained
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_svm = metrics.accuracy_score(y_prediction_svm, y_test).round(4)
print("----------------------------------")
print('The accuracy of the SVM is: {}'.format(score_svm))
print("----------------------------------")
# save the accuracy score
score.append(score_svm)
```

```
----------------------------------
The accuracy of the SVM is: 0.5686
----------------------------------
```

Compare Accuracy Score of Different Models

```
#comparing traning model's accuracy
sns.set_style("darkgrid")
plt.figure(figsize=(22,8))
classifier = ['NB','KNN','DT','LR','SVM']
ax = sns.barplot(x=classifier, y=score, palette = "colorblind")
plt.xlabel("Classification Models", fontsize = 20 )
plt.ylabel("Accuracy", fontsize = 20)
plt.title("Accuracy of different Classification Models", fontsize = 20)
plt.xticks(fontsize = 13, horizontalalignment = 'center')
plt.yticks(fontsize = 13)
plt.show()
```



Section 5: Discussion and Conclusion

In this report, we did an analysis of a data set known as the “Water Potability Prediction”. Here, we developed 5 different classifier models which are Gaussian Naive Bayes, K Nearest Neighbors (KNN), Decision Tree, Logistic Regression and Support Vector Machine. We can see that the lowest accuracy model is which accuracy is 0.5518 and the highest accuracy is 0.6021 and 0.6006. The Decision Tree (DT) and Gaussian Naive Bayes (NB) model gives us the maximum accuracy and K Nearest Neighbors (KNN) model gives us the lowest accuracy. Logistic Regression (LR) and Support Vector Machine (SVM) models are given the same accuracy which are 0.5686. As a result, we can say that the Decision Tree (DT) and Gaussian Naive Bayes (NB) classifier are the best use for this dataset model. The Decision Tree (DT) and Gaussian Naive Bayes classifier model's accuracy is below 70% because of the dataset. It might perform better if we can train this model on a larger dataset.

So, in our opinion, depending on this dataset, the Decision Tree (DT) and Gaussian Naive Bayes classifier is much better for predicting the water potability. Although, for a larger dataset other models may perform better.