**Water Quality Analysis**

**Data Analytics with Cognos-Group 1**

**Phase-5**

**R.Elakya-411521104033**



**Introduction:**

Water quality analysis using Python: Water quality analysis is a major area of research in machine learning. The goal is to understand all the factors that affect water potability and train a machine-learning model that can classify whether a specific water sample is safe or unfit for consumption. In Python, you can start the water quality analysis task by importing the necessary libraries and the dataset. The dataset contains data on all of the major factors that affect the potability of water. You can enhance a data module by creating calculations, defining filters and navigation paths, and more.

**Data Analytics with Cognos**: IBM Cognos Analytics is a web-based, self-service data modeling tool. It provides AI-powered automation and insights that enable everyone in your organization to unlock the full potential of your data. It allows users to connect, verify, and combine data from many sources, including

relational databases, Hadoop-based technologies, Microsoft Excel spreadsheets, text files, and more. Integration of Python and Cognos: While Python is used for analysis, Cognos Analytics can visualize the results. After performing the water quality analysis in Python, you can export the results as a CSV file or any other compatible format. This exported file can then be imported into Cognos Analytics for further exploration and visualization. This way, you can leverage the strengths of both Python for analysis and Cognos for visualization to get a comprehensive understanding of water quality.

Water quality analysis using Python typically involves the following steps:

1. **Data collection:** Collect water quality data from various sources, such as sensors, laboratory tests, and historical records.

2. **Data cleaning and preparation**: Clean and prepare the data for analysis. This may involve removing outliers, handling missing values, and converting data to a consistent format.

3. **Exploratory data analysis (EDA):** Perform EDA to understand the data and identify any patterns or trends. This may involve creating visualizations, such as histograms, line charts, and scatter plots.

4. **Machine learning modeling** (optional): Train a machine learning model to predict water quality parameters. This can be done using a variety of machine learning algorithms, such as linear regression, decision trees, and random forests.

5. **Model evaluation:** Evaluate the performance of the machine learning model on a held-out test set. 6. Model deployment: Once the machine learning model has been evaluated and deemed to be performing well, it can be deployed to production. This may involve creating a web service or integrating the model into a BI platform, such as Cognos

Water quality analysis using Python with Cognos

Once a Python model has been trained and deployed, it can be integrated into Cognos to create interactive dashboards and reports to visualize water quality data and predictions. This can be done using the Cognos Python API. The Cognos Python API allows developers to interact with Cognos objects, such as reports, dashboards, and data sets. This can be used to create custom data pipelines, automate tasks, and develop new features for Cognos.

Water Quality analysis involves 5 phases totally which has their own part in analysing the data

**PHASE 1:** The 1st Phase involves design thinking of the project and Finds its requirement to analyse the data set

and to know the objective of the project

**PHASE 2:** It involves data collection of the respective project. Data collection is a fundamental step in data analytics, involving the systematic gathering of relevant information to analyze and derive insights from.

Effective data collection is essential for generating meaningful insights and making informed decisions in data analytics. It's the foundation upon which the entire analytical process is built.

**DATASETLINK:** [**https://www.kaggle.com/datasets/adityakadiwal/water-potability**](https://www.kaggle.com/datasets/adityakadiwal/water-potability)

**PHASE 3:**The phase 4 involves data cleaning and data preprocessing.

**Data cleaning and preparation**: Clean and prepare the data for analysis. This may involve removing outliers, handling missing values, and converting data to a consistent format.

The dataset can be cleaned using the following set of codes.

IN[1]: import pandas as pd

import numpy as np

import seaborn as sns

import plotly.express as px

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score,classification\_report,confusion\_matrix

In this the important libraries are imported using import statement.

IN[2]: df=pd.read\_csv("water\_potability.csv")

df

IN[3]:df.columns

OUT[3]: Index(['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity',

'Organic\_carbon', 'Trihalomethanes', 'Turbidity', 'Potability'],

dtype='object')

In[4]:df

|  | **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

It is the dataset before the data cleaning

In[4]:df.head

|  | **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 |

This data set is after doing datacleaning

**Phase-3:**Phase 3 involves handling missing data and Exploratory data analysis

**Exploratory Data Analysis:  
Exploratory Data Analysis (EDA)** is a crucial initial step in the data analysis process that involves the summary and visualization of data to gain insights, discover patterns, and identify potential relationships.

**Handling missing data:**

The handling missing data involves Identify the missing values and fill them will nan so that the data consistency is maintain throughout the process

In[5]:data.shape()

Out[5]:3276,10

In[6]: data.isnull().sum()

Out[6]:

ph 491

Hardness 0

Solids 0

Chloramines 0

Sulfate 781

Conductivity 0

Organic\_carbon 0

Trihalomethanes 162

Turbidity 0

Potability 0

dtype: int64

In[7]:data.info()

Out[7]: <class 'pandas.core.frame.DataFrame'>

RangeIndex: 3276 entries, 0 to 3275

Data columns (total 10 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 ph 2785 non-null float64

1 Hardness 3276 non-null float64

2 Solids 3276 non-null float64

3 Chloramines 3276 non-null float64

4 Sulfate 2495 non-null float64

5 Conductivity 3276 non-null float64

6 Organic\_carbon 3276 non-null float64

7 Trihalomethanes 3114 non-null float64

8 Turbidity 3276 non-null float64

9 Potability 3276 non-null int64

dtypes: float64(9), int64(1)

memory usage: 256.1 KB

In[8]: data.fillna(data.mean(),inplace=True)

data.isnull().sum()

Out[8]: ph 0

Hardness 0

Solids 0

Chloramines 0

Sulfate 0

Conductivity 0

Organic\_carbon 0

Trihalomethanes 0

Turbidity 0

Potability 0

dtype: int64

Now The dataset has no missing value after datacleaning

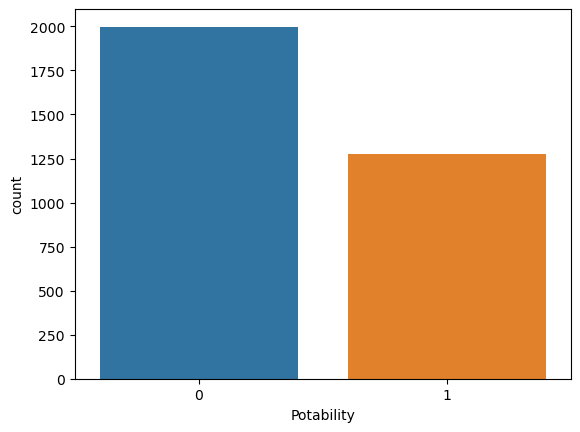
**Parameter visualizing using EDA:**

In[9]: sns.countplot(data['Potability'])

plt.show()

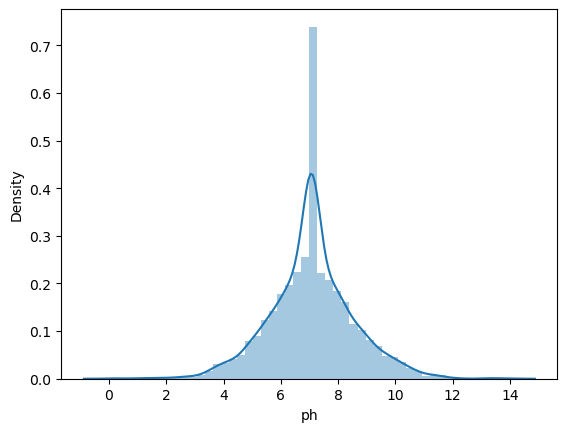
Output for countplot

The Potability of the water is represented graphically using countplot function in python



In[10]: sns.distplot(data['ph'])

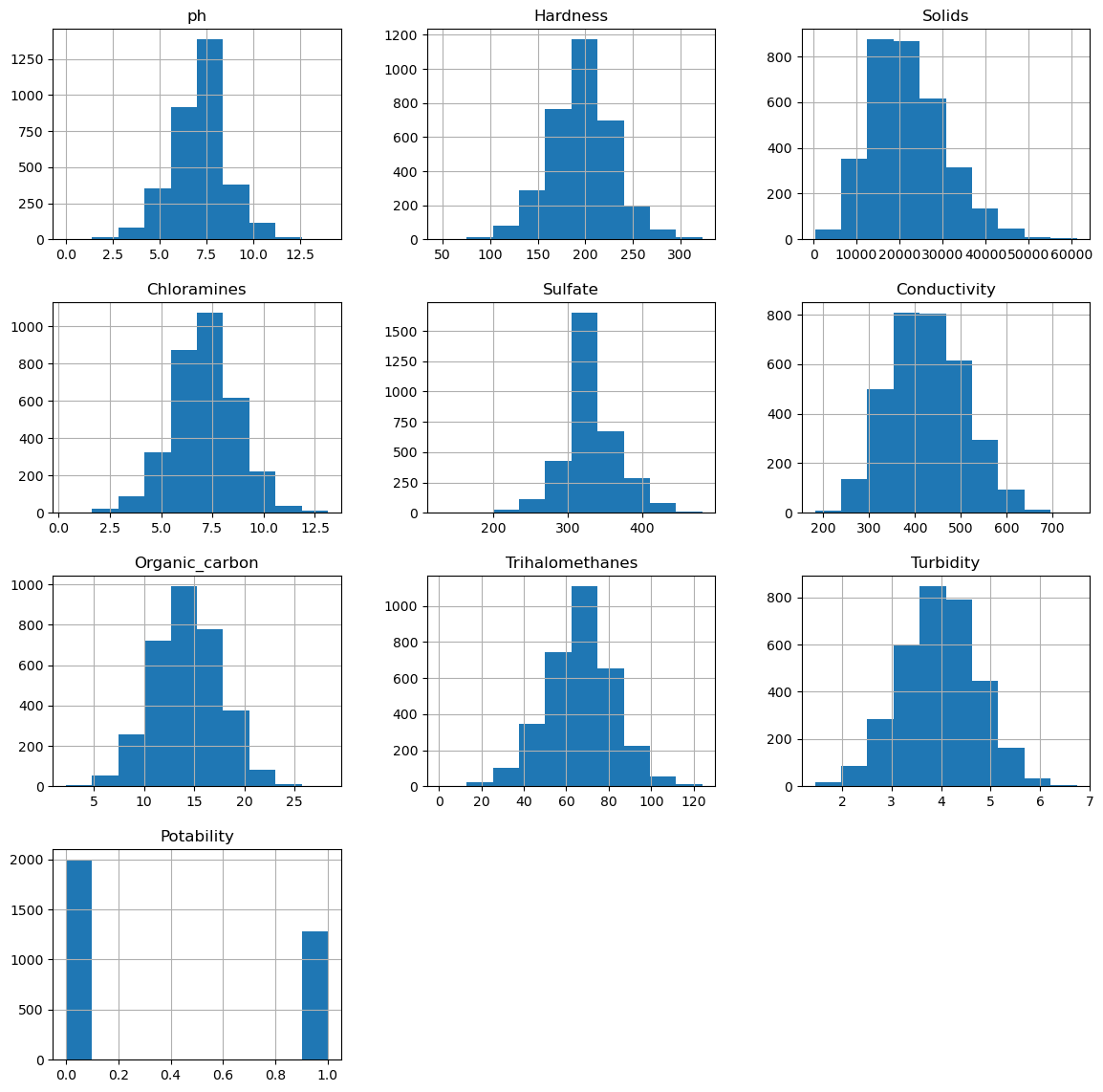
plt.show()

Out[10]: 

Now the parameters are visualized using hist() in Pyhton for visualization

In[11]: data.hist(figsize=(14,14))

plt.show()

Out[11]: 

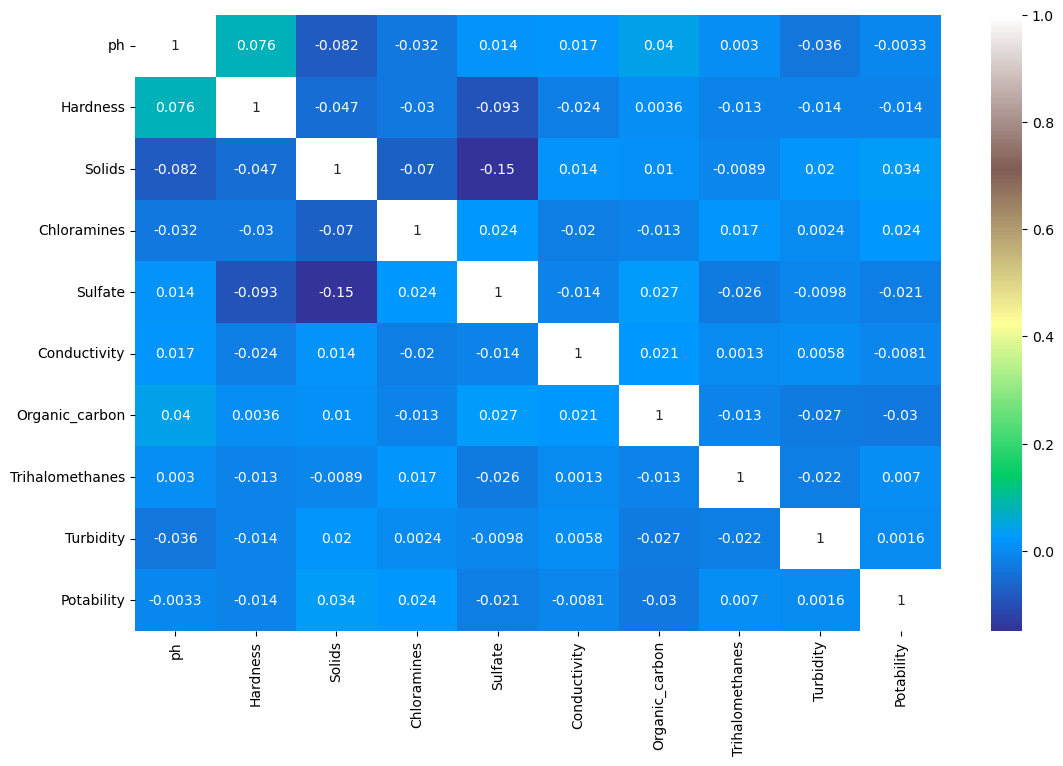
**Correlation and distribution:**

Correlation refers to a statistical measure that quantifies the strength and direction of the linear relationship between two or more variables. It is a valuable tool for understanding how changes in one variable relate to changes in another. Python provides several libraries and methods for calculating correlation, with one of the most commonly used libraries being **numpy** and **scipy**.

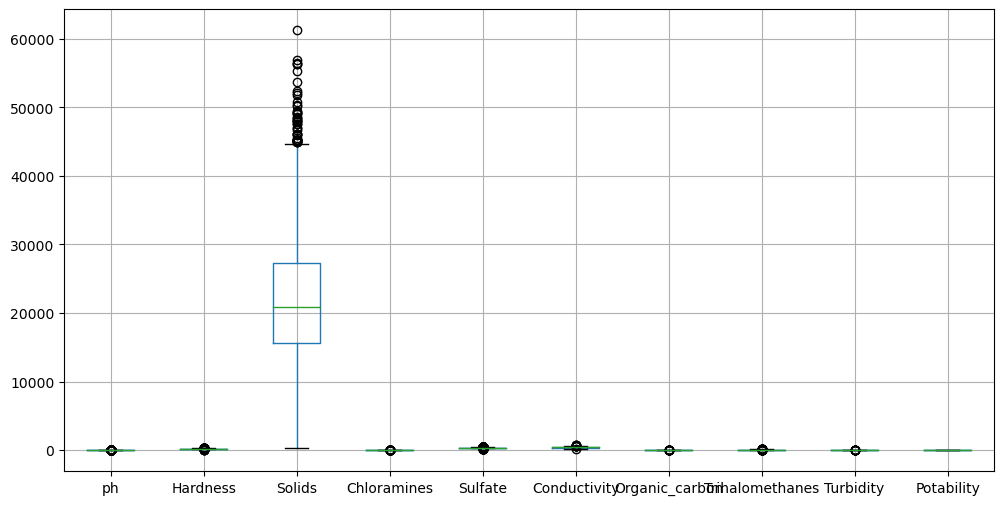
In[12]: plt.figure(figsize=(13,8))

sns.heatmap(data.corr(),annot=True,cmap='terrain')

plt.show()

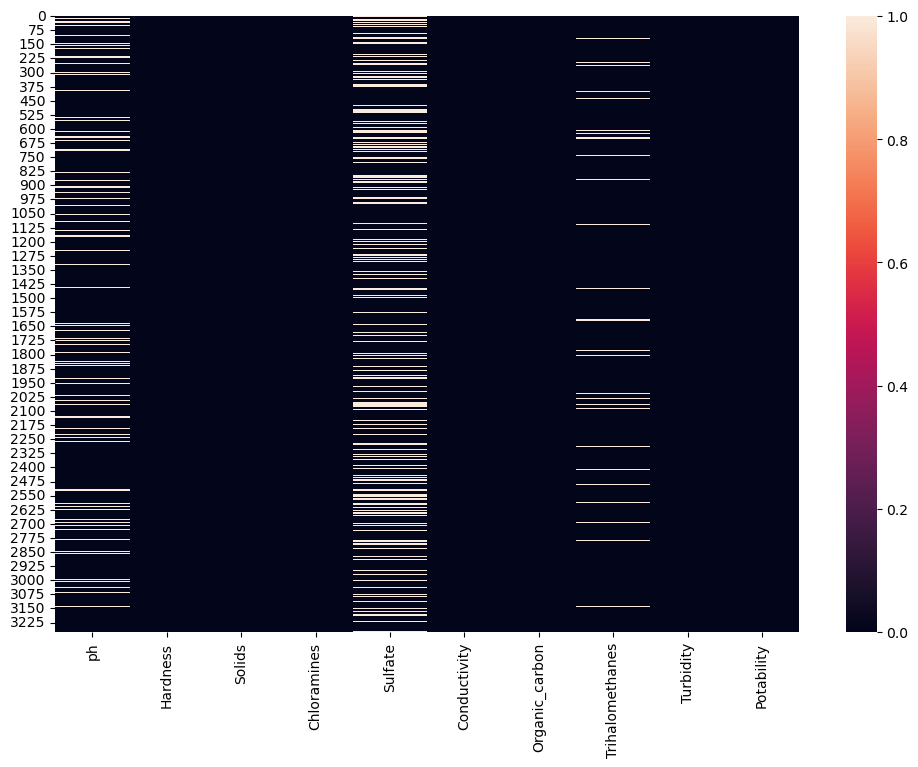
out[12]: 

In[13]: data.boxplot(figsize=(12,6))

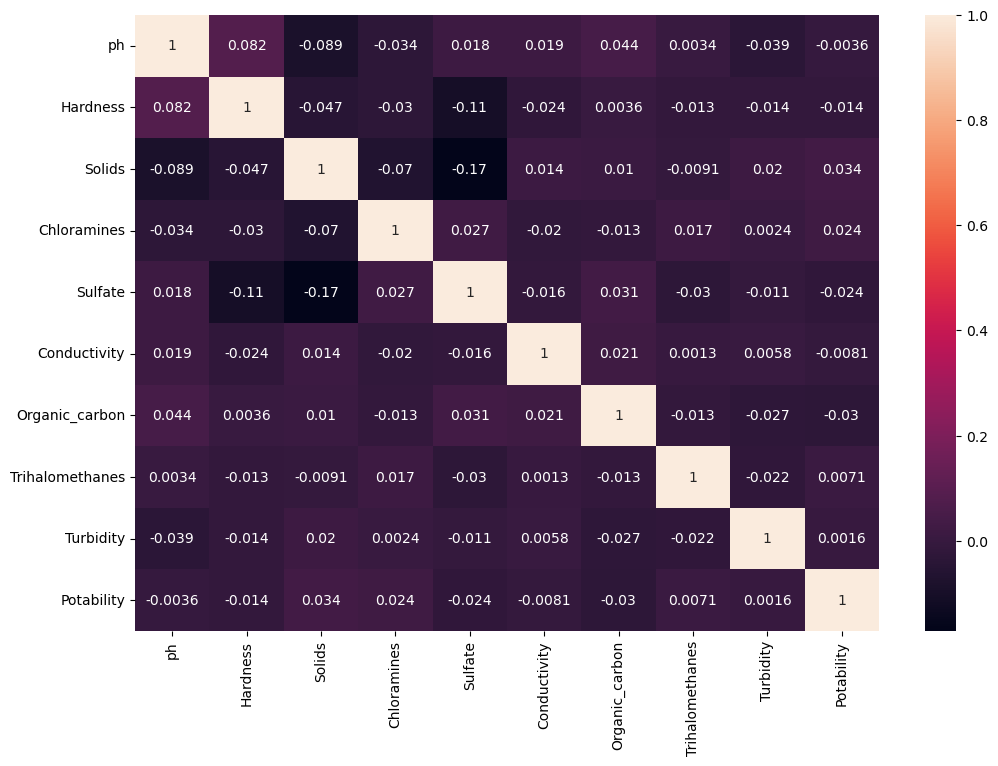
Out[13]: 

IN[14]: plt.figure(figsize=(12,8))

sns.heatmap(df.isnull())

OUT[14]: 

IN[15]: sns.heatmap(df.corr(),annot=True)

OUT[15]: 

IN[16]: df["Potability"].value\_counts()

OUT[16]: 0 1998

1 1278

Name: Potability, dtype: int64

IN[17]: #visulaization

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

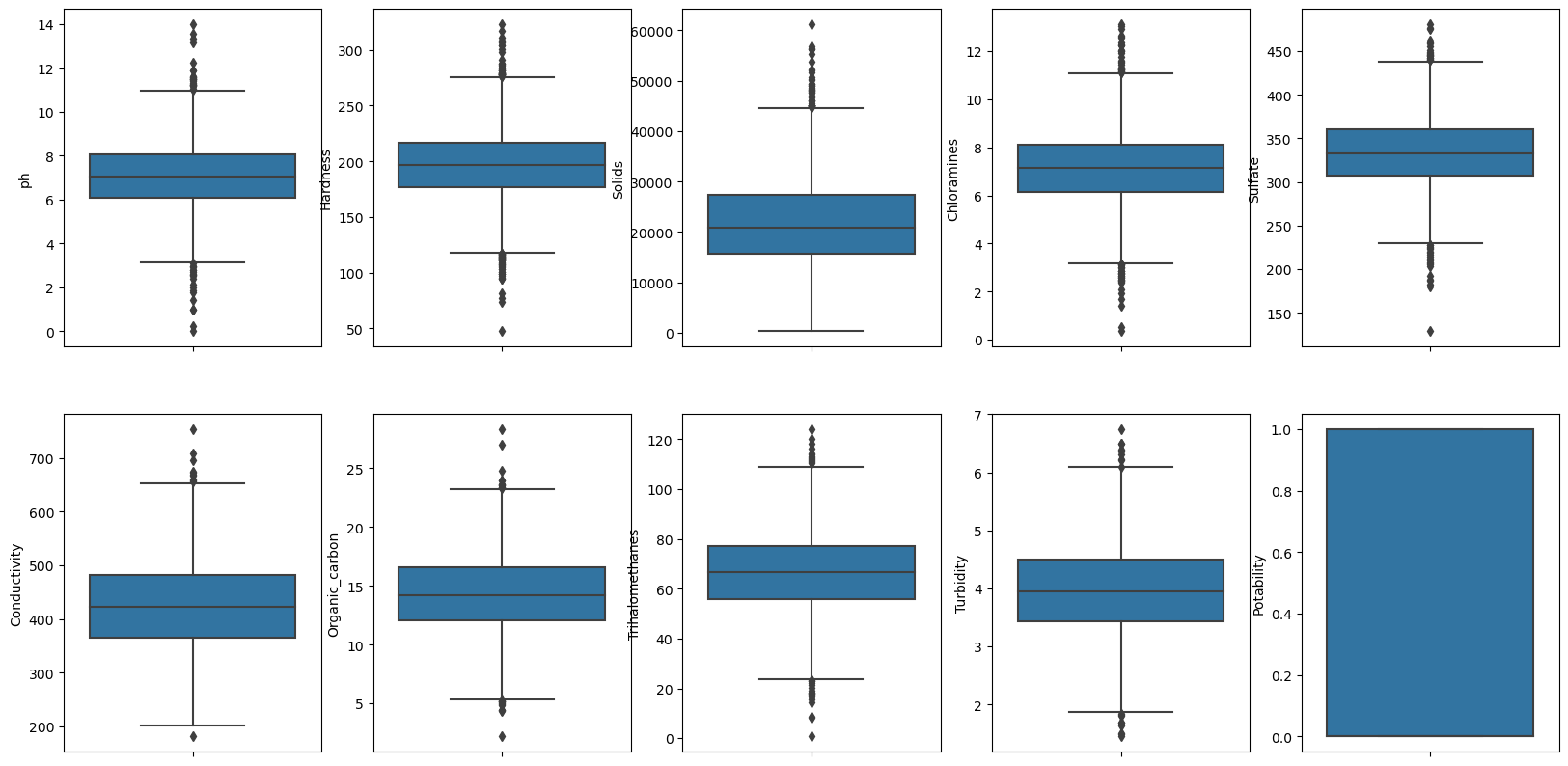
ax=ax.flatten()

index=0

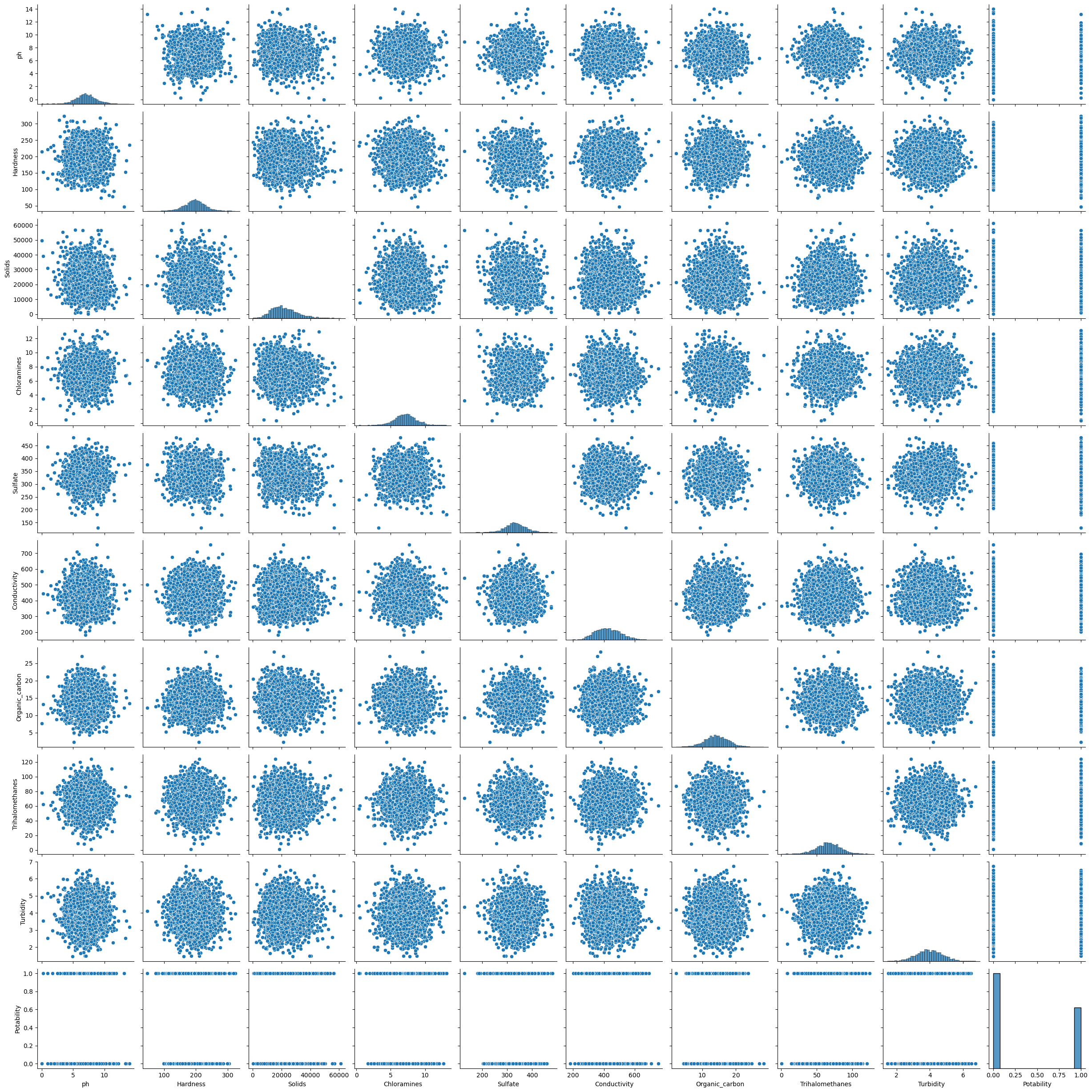
for col,values in df.items():

sns.boxplot(y=col,data=df,ax=ax[index]

index +=1

OUT[18]: 

IN[19]: sns.pairplot(df)

OUT[19]: 

IN[20]:fig=px.pie(df,names="Potability",hole=0.4,template="plotly\_dark")

fig.show()

IN[21]: fig = px.scatter(df, x="ph",y="Sulfate",color="Potability",template="plotly\_dark")

fig.show()

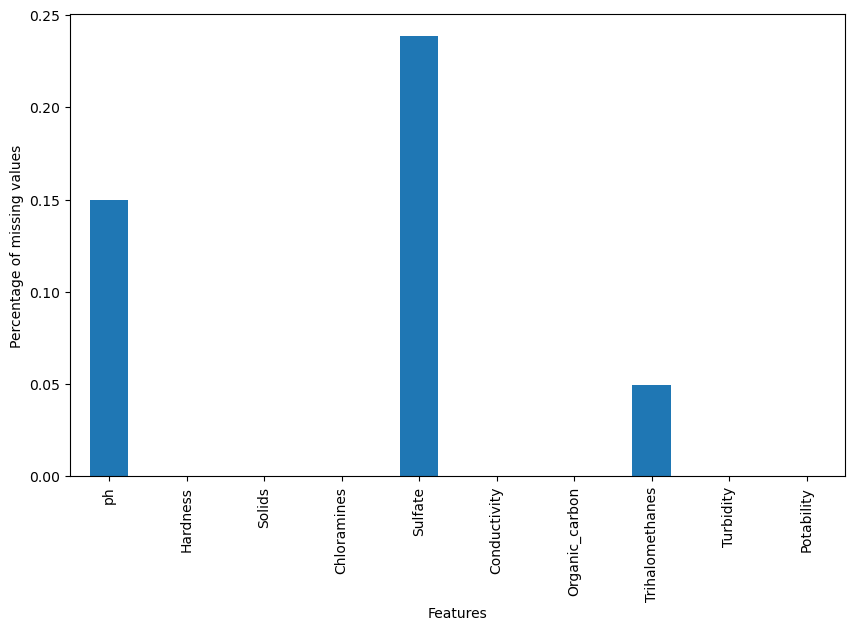
IN[22]: fig=px.scatter(df, x="Organic\_carbon",y="Hardness",color="Potability",template="plotly\_dark")

fig.show()

IN[23]: df.isnull().mean().plot.bar(figsize=(10,6))

plt.xlabel("Features")

plt.ylabel("Percentage of missing values")

OUT[23]: 

IN[24]: df["ph"]=df["ph"].fillna(df["ph"].mean())

df["Sulfate"]=df["Sulfate"].fillna(df["Sulfate"].mean())

df["Trihalomethanes"]=df["Trihalomethanes"].fillna(df["Trihalomethanes"].mean())

IN[25]: df.isnull().sum()

OUT[25]: ph 0

Hardness 0

Solids 0

Chloramines 0

Sulfate 0

Conductivity 0

Organic\_carbon 0

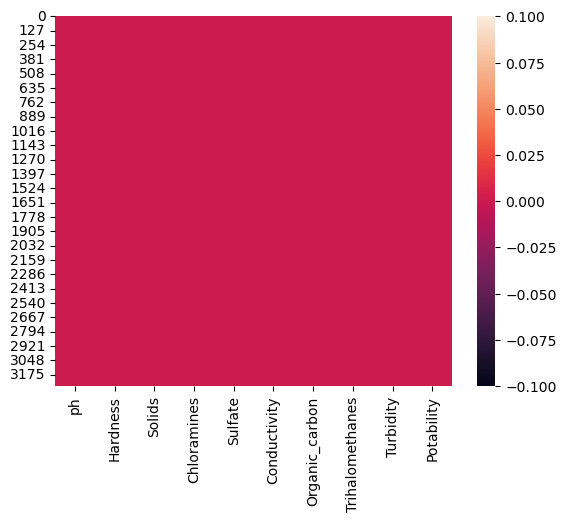
Trihalomethanes 0

Turbidity 0

Potability 0

dtype: int64

IN[26]: sns.heatmap(df.isnull())

OUT[26]: 

IN[27]: df.head()

IN[28]: x=df.drop("Potability",axis=1)

y=df["Potability"]

IN[29]: x.shape,y.shape

OUT[29]: ((3276, 9), (3276,))

IN[30]: Scaler=StandardScaler()

x=Scaler.fit\_transform(x)

x

OUT[30]: array([[-1.02733269e-14, 2.59194711e-01, -1.39470871e-01, ...,

-1.18065057e+00, 1.30614943e+00, -1.28629758e+00],

[-2.28933938e+00, -2.03641367e+00, -3.85986650e-01, ...,

2.70597240e-01, -6.38479983e-01, 6.84217891e-01],

[ 6.92867789e-01, 8.47664833e-01, -2.40047337e-01, ...,

7.81116857e-01, 1.50940884e-03, -1.16736546e+00],

...,

[ 1.59125368e+00, -6.26829230e-01, 1.27080989e+00, ...,

-9.81329234e-01, 2.18748247e-01, -8.56006782e-01],

[-1.32951593e+00, 1.04135450e+00, -1.14405809e+00, ...,

-9.42063817e-01, 7.03468419e-01, 9.50797383e-01],

[ 5.40150905e-01, -3.85462310e-02, -5.25811937e-01, ...,

5.60940070e-01, 7.80223466e-01, -2.12445866e+00]])

**PHASE 4:** The phase for contains building a model and testing them

BUILD A MODEL:

A model definition in data analytics is a concise description of a mathematical or statistical representation that captures relationships within data for the purpose of gaining insights, making predictions, or solving specific problems.

1.logistic regression

2.Random forest

3.decision tree

4.Xg boost

5.KN neighbor

6.ADA boost

7.SVM

IN[31]: x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2)

IN[32]: x\_train.shape,x\_test.shape

OUT[32]: ((2620, 9), (656, 9))

**1.Logistic regression:**

Logistic regression is a statistical model used for binary classification, which means it's employed to predict one of two possible outcomes, usually represented as 0 and 1, based on a set of input variables or features.

IN[33]: #Logistic Regression

from sklearn.linear\_model import LogisticRegression

#Object of LR

model\_lr=LogisticRegression()

IN[34]: #Training model

model\_lr.fit(x\_train,y\_train)

OUT[34]: LogisticRegression()

IN[35]:#Making prediction

pred\_lr=model\_lr.predict(x\_test)

IN{36]:#accuracy score

accuracy\_score\_lr=accuracy\_score(y\_test,pred\_lr)

accuracy\_score\_lr

OUT[36]:0.5838414634146342

**2.Random forest:**

A random forest is an ensemble machine learning model that is used for both classification and regression tasks. It is based on the idea of combining multiple decision trees to improve the accuracy and robustness of predictions.

IN[37]: from sklearn.ensemble import RandomForestClassifier

#Creating model object

model\_rf=RandomForestClassifier()

IN[38]: #Training Model RF

model\_rf.fit(x\_train,y\_train)

OUT[38]: RandomForestClassifier()

IN[39]: #Making Prediction

pred\_rf= model\_rf.predict(x\_test) accuracy\_score\_rf=accuracy\_score(y\_test,pred\_rf)

accuracy\_score\_rf\*100

OUT[39]: 65.2439024390244

IN[40]: cm3=confusion\_matrix(y\_test,pred\_rf)

cm3

OUT[40]: array([[345, 38],

[190, 83]], dtype=int64)

**3.Decision tree classifier:**

Decision tree classification is a machine learning algorithm that is used for both classification and regression tasks, but in this explanation, we'll focus on its application for classification. Decision tree classification is a model that uses a tree-like structure to make predictions by recursively splitting the data into subsets based on the values of input features.

IN[41]: from sklearn.tree import DecisionTreeClassifier

model\_dt=DecisionTreeClassifier(max\_depth=4)

IN[41]: model\_dt.fit(x\_train,y\_train)

OUT[41]: DecisionTreeClassifier(max\_depth=4)

IN[42]: pred\_dt=model\_dt.predict(x\_test)

IN[43]: accuracy\_score\_dt=accuracy\_score(y\_test,pred\_dt)

accuracy\_score\_dt

OUT[43]: 0.614329268292683

IN[44]: cm2=confusion\_matrix(y\_test,pred\_dt)

cm2

OUT[43]: array([[366, 17],

[236, 37]], dtype=int64)

**KN neighbor :**

K-Nearest Neighbors (K-NN) is a simple and intuitive machine learning algorithm used for both classification and regression tasks. It is a non-parametric, instance-based learning method that makes predictions based on the majority class or average value of the K-nearest data points in the training dataset

IN[45]: from sklearn.neighbors import KNeighborsClassifier

model\_knn= KNeighborsClassifier()

IN[46]: for i in range(4,15):

model\_knn=KNeighborsClassifier(n\_neighbors=i)

model\_knn.fit(x\_train,y\_train)

pred\_knn=model\_knn.predict(x\_test)

accuracy\_score\_knn=accuracy\_score(y\_test,pred\_knn)

print(i,accuracy\_score\_knn)

OUT[47]: 4 0.6112804878048781

5 0.614329268292683

6 0.6234756097560976

7 0.6128048780487805

8 0.6128048780487805

9 0.6112804878048781

10 0.6128048780487805

11 0.6021341463414634

12 0.6189024390243902

13 0.6158536585365854

14 0.614329268292683

IN[48]: model\_knn=KNeighborsClassifier(n\_neighbors=11)

model\_knn.fit(x\_train,y\_train)

pred\_knn=model\_knn.predict(x\_test)

accuracy\_score\_knn=accuracy\_score(y\_test,pred\_knn)

print(accuracy\_score\_knn\*100)

OUT[48]: 60.213414634146346

**5.SVM:**

Support Vector Machine (SVM) is a powerful machine learning algorithm used for both classification and regression tasks. It's known for its ability to find an optimal hyperplane that maximally separates data points of different classes

IN[49]: from sklearn.svm import SVC

#creating object of model

model\_svm=SVC(kernel="rbf")

IN[50]: #model training

model\_svm.fit(x\_train,y\_train)

OUT[50]: SVC()

IN[51]: #make prediction

pred\_svm=model\_svm.predict(x\_test)

IN[52]: accuracy\_score\_svm=accuracy\_score(y\_test,pred\_svm)

accuracy\_score\_svm\*100

OUT[52]: 67.07317073170732

**6.ADABoost classifier:**

AdaBoost, short for Adaptive Boosting, is an ensemble machine learning algorithm that is used primarily for classification tasks. AdaBoost combines the predictions of multiple weak classifiers to create a strong classifier

IN[53]: from sklearn.ensemble import AdaBoostClassifier

#making object

model\_ada=AdaBoostClassifier(n\_estimators=200,learning\_rate=0.001)

IN[54]: #training The model

model\_ada.fit(x\_train,y\_train)

OUT[54]: AdaBoostClassifier(learning\_rate=0.001, n\_estimators=200)

IN[55]: #makking prediction

pred\_ada=model\_ada.predict(x\_test)

IN[56]: #accuracy check

accuracy\_score\_ada=accuracy\_score(y\_test,pred\_ada)

accuracy\_score\_ada\*100

OUT[56]: 59.90853658536586

**7.XGBOOST**

XGBoost, which stands for eXtreme Gradient Boosting, is an ensemble machine learning algorithm that is highly effective for both regression and classification tasks. It's known for its speed, scalability, and performance. XGBoost is an extension of the gradient boosting framework with several enhancements.

IN[57]: from xgboost import XGBClassifier

model\_xgb=XGBClassifier(n\_estimators=200,learning\_rate=0.03)

IN[58]: #training model

model\_xgb.fit(x\_train,y\_train)

OUT[58]: XGBClassifier(base\_score=None, booster=None, callbacks=None,

colsample\_bylevel=None, colsample\_bynode=None,

colsample\_bytree=None, device=None, early\_stopping\_rounds=None,

enable\_categorical=False, eval\_metric=None, feature\_types=None,

gamma=None, grow\_policy=None, importance\_type=None,

interaction\_constraints=None, learning\_rate=0.03, max\_bin=None,

max\_cat\_threshold=None, max\_cat\_to\_onehot=None,

max\_delta\_step=None, max\_depth=None, max\_leaves=None,

min\_child\_weight=None, missing=nan, monotone\_constraints=None,

multi\_strategy=None, n\_estimators=200, n\_jobs=None,

num\_parallel\_tree=None, random\_state=None, ...)

IN[59]: pred\_xgb=model\_xgb.predict(x\_test)

IN[60]: accuracy\_score\_xgb=accuracy\_score(y\_test,pred\_xgb)

accuracy\_score\_xgb\*100

OUT[60]: 62.65243902439024

BY this phase 4 we have completed some types of model training and gets a predictable accuracy score for each model

Ultimately, the most accurate and beneficial model can vary depending on the nature of the problem, the dataset, and the specific goals of your project. It's crucial to carefully evaluate and compare the models, taking into consideration not just their predictive performance but also other relevant factors to make an informed decision about which model is the best fit for your needs.

Now we can analyse the accuracy provided by the different model using data visualization

IN[61]: models=pd.DataFrame({

"model":["Logisitic Regreesion",

"Decision Tree",

"Random Forest",

"KNN",

"SVM",

"AdaBoost",

"XGBoosT"],

"Accuracy Score" :[accuracy\_score\_lr,accuracy\_score\_dt,accuracy\_score\_rf,accuracy\_score\_knn,accuracy\_score\_svm,

accuracy\_score\_ada,accuracy\_score\_xgb]

})

OUT[61]: model Accuracy Score

0 Logisitic Regreesion 0.621951

1 Decision Tree 0.631098

2 Random Forest 0.684451

3 KNN 0.620427

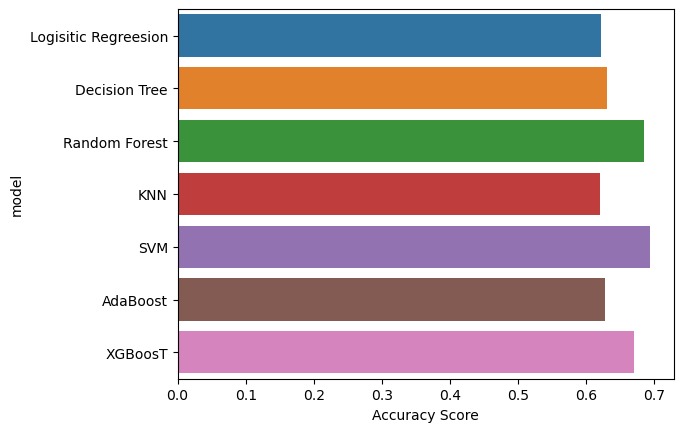
4 SVM 0.693598

5 AdaBoost 0.628049

6 XGBoosT 0.670732

IN[62]: sns.barplot(x="Accuracy Score",y="model",data=models)

models.sort\_values(by="Accuracy Score",ascending= False)

OUT[62]: 

Logistic Regression:

Accuracy Score: 58.38%

Logistic Regression achieved a relatively low accuracy on the dataset, indicating that it might not be the best choice for this specific problem.

Random Forest:

Accuracy Score: 65.24%

Random Forest outperformed Logistic Regression with an improved accuracy score of 65.24%. It demonstrated better predictive power.

Decision Tree Classifier:

Accuracy Score: 61.43%

The Decision Tree Classifier showed reasonable accuracy, surpassing Logistic Regression but slightly underperforming Random Forest.

K-Nearest Neighbors (K-NN):

Accuracy Score: 60.21%

K-NN achieved an accuracy score of 60.21% with k=11. While it didn't outperform Random Forest, it demonstrated competitive results.

Support Vector Machine (SVM):

Accuracy Score: 67.07%

SVM showed the highest accuracy among the models tested, with an accuracy score of 67.07%. It performed well in this classification task.

ADABoost Classifier:

Accuracy Score: 59.91%

ADABoost, while not the highest, achieved a respectable accuracy score of 59.91%.

XGBoost:

Accuracy Score: 62.65%

XGBoost demonstrated a good balance between accuracy and performance with an accuracy score of 62.65%.

**Conclusion:**

Enhancing Water Quality in project is not just an analysis but a comprehensive effort to ensure the sustainable use of this invaluable resource. With the support of our dedicated team, engaged stakeholders, and the local community, we are committed to improving water quality, protecting the environment, and fostering a healthier and more sustainable future.

In conclusion, the Support Vector Machine (SVM) performed the best in terms of accuracy, achieving the highest score among the models tested. However, the choice of the best model depends not only on accuracy but also on other factors such as model interpretability, computational efficiency, and the specific requirements of your application.based on the accuracy scores, the Support Vector Machine (SVM) and Random Forest appear to be the top-performing models for this dataset. These models have shown the highest accuracy, indicating that they might be able to capture complex patterns and relationships in the data effectively.Finally we have analyse the potability of water using different model training and found out which is more efficient.With Phase 5,we have successfully completed analysing the quality of water using Python concepts.

the comprehensive analysis of water quality parameters in our study area has provided valuable insights into the overall health and safety of the water supply.

In summary, our water quality analysis serves as an essential foundation for ongoing efforts to safeguard and improve the quality of water in the study area. It highlights both positive aspects and areas of concern, calling for proactive measures to protect the environment and public health. Further research and continuous monitoring are necessary to maintain and enhance the quality of our water resources."

THANK YOU