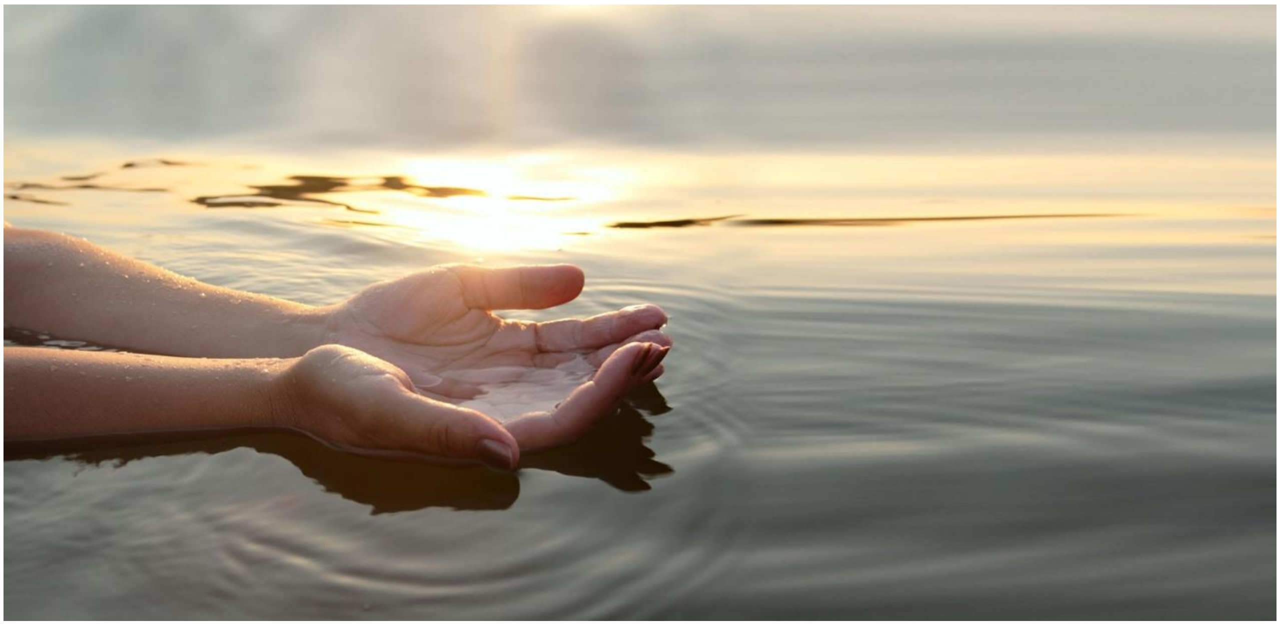
Water Quality Prediction System using Machine Learning for Sustainable Resource Management

: Rakesh Singh



Abstract

The aim of this project is to develop a machine learning model for predicting water quality. Water quality is a critical factor in ensuring the health and safety of communities. By leveraging machine learning algorithms, we seek to create a predictive model that can anticipate water quality parameters and alert relevant authorities in case of potential issues. This project employs a dataset containing various water quality indicators, and through the application of machine learning, aims to contribute to the proactive management of water resources. The main aim of this project to make an analysis of water quality predication using machine learning algorithms with these kind of parameters such as Temperature (Temp), Dissolved Oxygen (DO) (% sat), pH, conductivity, Biochemical oxygen demand (BOD), nitrates (NO3) and total coli forms (TC).

Problem Statement:

Water quality is subject to various natural and anthropogenic factors. Monitoring water quality manually can be resource-intensive and may not provide real-time insights. This project addresses the need for an automated system that can predict water quality based on historical data. By identifying patterns and correlations in the data, the model aims to forecast potential issues, allowing for timely interventions and ensuring the delivery of safe and clean water.

Market/Customer/Business Need Assessment:

The water quality prediction project caters to a growing market demand for proactive water management solutions, targeting municipalities, water treatment facilities, and environmental agencies. With a focus on cost-effectiveness, regulatory compliance, and operational efficiency, the project addresses critical business needs by providing a cutting-edge, machine learning-based system. By offering customization options and aligning with long-term sustainability goals, theproject aims to differentiate itself in a competitive landscape, positioning as an essential tool for future-proof water resource management strategies.

Target Specification:

The water quality prediction project is designed to meet the needs of municipalities, water treatment facilities, and environmental agencies, providing a proactive solution to address water quality challenges. With a focus on cost-effectiveness, regulatory compliance, and operational efficiency, the project aims to offer a customizable and sustainable tool for long-term water resource management.

Business Opportunity:

The water quality prediction project seizes a lucrative business opportunity by providing municipalities, water treatment facilities, and environmental agencies with a proactive and cost-effective solution. Monetization involves licensing the predictive model, offering subscription-based services, and customization for client-specific needs. This business model ensures recurring revenue while addressing critical water quality challenges and positioning the project as a valuable asset in the competitive landscape of water resource management.

Implementation:

This Python code conducts a comprehensive analysis of water potability data. It starts by loading and exploring the dataset, handling missing values, and visualizing distributions and relationships. The code further includes model training using a Random Forest classifier and evaluates its accuracy on a test set, providing insights into the quality of the predictive model.

Importing the dependencies

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
```

Loading Dataset

In [2]:
```
data=pd.read_csv("C:\\Users\\Rakesh\\Downloads\\water_potability.csv")
```

Now we will create a DataFrame named ‘df’ to work with the data.

Creating Dataframe

In [3]:
```
df=pd.DataFrame(data)
```

Exploring Data:

1. df.shape: Prints the dimensions (rows, columns) of the DataFrame.

2. df.info(): Provides information about the DataFrame, including data types and non-null counts.

3. df.describe(): Gives statistical summary (mean, min, max, etc.) of the numerical columns.

4. df.isnull().sum(): Displays the count of missing values in each column.

Exploring Data

In [4]:
```
df.shape
```

Out[4]: (3276, 10)

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               2785 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3114 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

In [6]:
```
df.describe()
```

Out[6]:

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

Filling missing values in the DataFrame with the mean of each column.

Handling the missing values

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

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               3276 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          3276 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3276 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

This step calculates the skewness of numerical columns in the DataFrame, excluding the 'Potability' column. Skewness measures the asymmetry of the distribution. A skewness close to 0 indicates a symmetric distribution.

Skewness

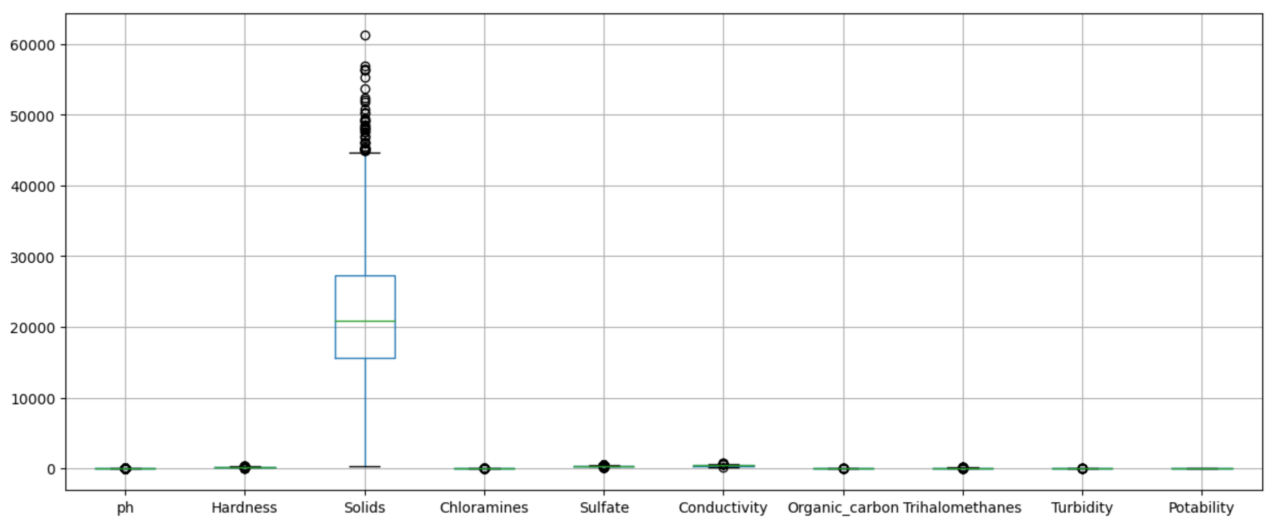
In [10]:
```
#Skewness
df.drop('Potability', axis=1).skew()
```

Out[10]:
```
ph                 0.027796
Hardness          -0.039342
Solids             0.621634
Chloramines       -0.012098
Sulfate           -0.041184
Conductivity       0.264490
Organic_carbon     0.025533
Trihalomethanes   -0.085161
Turbidity         -0.007817
dtype: float64
```

A boxplot is created for the 'Solids' column to visualize the distribution and identify potential outliers. The boxplot displays the median, quartiles, and any outliers in the data.

Box Plot for Outliers

In [13]:
```
#checking the outlier using Box plot
df.boxplot(figsize=(15,6))
plt.show()
```

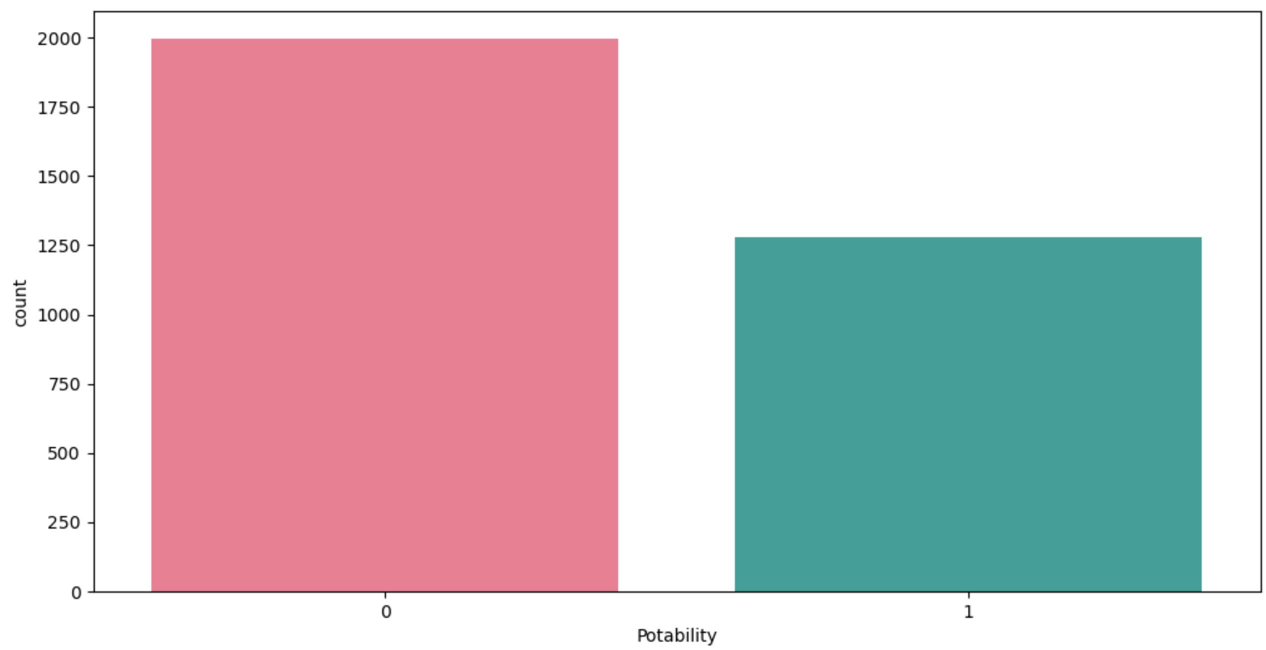


This will provide descriptive statistics for the 'Solids' column.

```
In [14]: df['Solids'].describe()

Out[14]: count     3276.000000
         mean     22014.092526
         std       8768.570828
         min        320.942611
         25%      15666.690297
         50%      20927.833607
         75%      27332.762127
         max      61227.196008
         Name: Solids, dtype: float64

In [15]: df['Solids']  # not removing ouotliers , it can help later

Out[15]: 0       20791.318981
         1       18630.057858
         2       19909.541732
         3       22018.417441
         4       17978.986339
                     ...
         3271    47580.991603
         3272    17329.802160
         3273    33155.578218
         3274    11983.869376
         3275    17404.177061
         Name: Solids, Length: 3276, dtype: float64
```

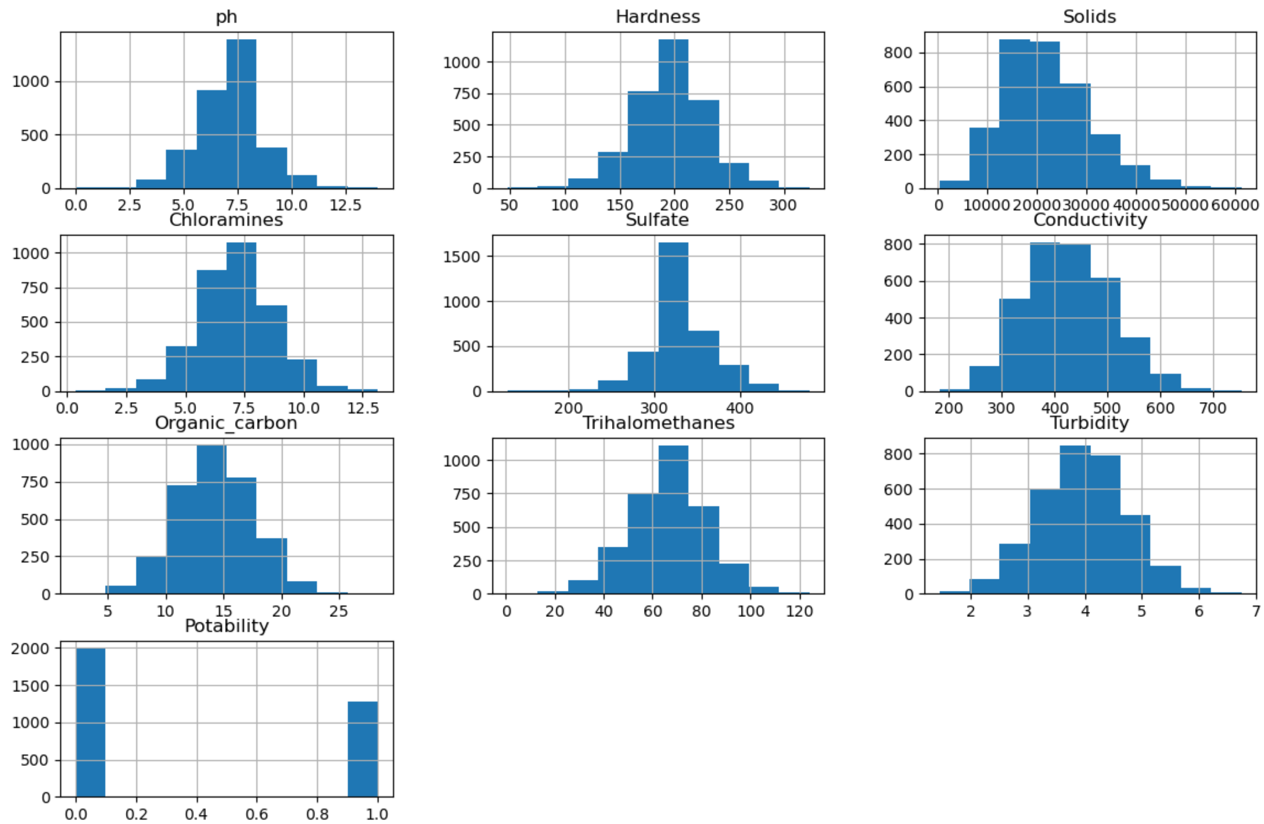
Univariate Analysis – Histograms

Histograms are plotted for each numerical column in the DataFrame. Histograms provide a visual representation of the distribution of data, showing the frequency of values in different bins.



Creating a count plot to visualize the distribution of 'Potability' classes.

In [32]:
```
# Count plot for potability
plt.figure(figsize=(12, 6))
sns.countplot(x="Potability", data=data, palette='husl');
```



Bivariate Analysis - Correlation

This step calculates the correlation matrix for numerical columns in the DataFrame. The correlation matrix shows the relationships between pairs of variables. The heatmap visualizes the correlations, with annotations displaying the correlation coefficients.

Bivariate Analysis - Correlation

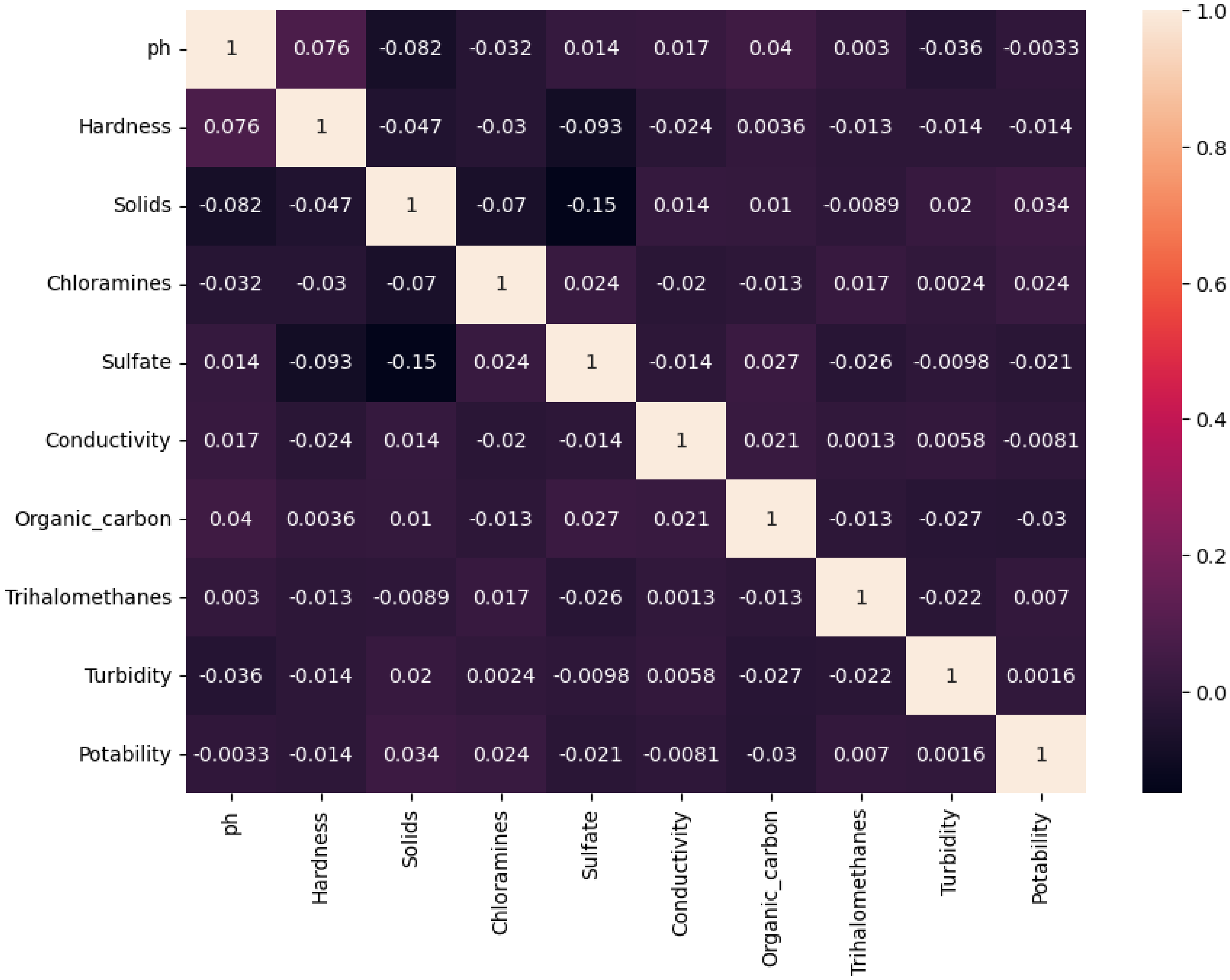
In [20]:
```
# Bivariate Analysis
df.corr()
```

Out[20]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
ph	1.000000	0.075833	-0.081884	-0.031811	0.014403	0.017192	0.040061	0.002994	-0.036222	-0.003287
Hardness	0.075833	1.000000	-0.046899	-0.030054	-0.092766	-0.023915	0.003610	-0.012690	-0.014449	-0.013837
Solids	-0.081884	-0.046899	1.000000	-0.070148	-0.149840	0.013831	0.010242	-0.008875	0.019546	0.033743
Chloramines	-0.031811	-0.030054	-0.070148	1.000000	0.023791	-0.020486	-0.012653	0.016627	0.002363	0.023779
Sulfate	0.014403	-0.092766	-0.149840	0.023791	1.000000	-0.014059	0.026909	-0.025605	-0.009790	-0.020619
Conductivity	0.017192	-0.023915	0.013831	-0.020486	-0.014059	1.000000	0.020966	0.001255	0.005798	-0.008128
Organic_carbon	0.040061	0.003610	0.010242	-0.012653	0.026909	0.020966	1.000000	-0.012976	-0.027308	-0.030001
Trihalomethanes	0.002994	-0.012690	-0.008875	0.016627	-0.025605	0.001255	-0.012976	1.000000	-0.021502	0.006960
Turbidity	-0.036222	-0.014449	0.019546	0.002363	-0.009790	0.005798	-0.027308	-0.021502	1.000000	0.001581
Potability	-0.003287	-0.013837	0.033743	0.023779	-0.020619	-0.008128	-0.030001	0.006960	0.001581	1.000000

This line calculates the correlation matrix for the DataFrame **df**. It is a square matrix where each entry represents the correlation coefficient between two variables.

```
In [21]:
#Correlation Plot
sns.heatmap(data.corr(),annot=True)
fig=plt.gcf()
fig.set_size_inches(10,7)
plt.show()
```



The sns.heatmap() function from the Seaborn library is used to create the heatmap. It takes the correlation matrix (data.corr()) as input. The annot=True parameter adds numerical annotations to the heatmap, displaying the actual correlation coefficients in each cell. Next lines create a **figure** object using **plt.gcf()** (get current figure) and set its size to 10 inches by 7 inches. This controls the overall size of the heatmap when it's displayed.

The data is split into input data (**x**) and the target variable (**y**). It is further divided into training and testing sets using the **train_test_split** function from scikit-learn.

Model Training ¶

In [23]:
```python
#Model Training
#Partitioning
x=df.drop('Potability',axis=1) #Input Data
y=df['Potability'] #Target variable
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,shuffle=True,random_state=2)

```

In [24]:
```python
x_test
```

Out[24]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity
2218	5.574756	189.700665	11512.465842	8.720972	329.939631	331.330352	18.016991	66.188642	4.371806
1626	6.505066	226.419609	16982.131974	6.938467	318.245710	484.092285	18.527105	80.462810	2.890999
2701	7.822257	200.002793	22411.006864	5.283594	343.080377	593.483602	12.273073	75.256212	3.080866
914	5.791154	190.431679	20288.235569	6.298366	347.041020	544.857481	17.706838	84.388191	4.104835
2362	7.859490	211.311029	31725.487766	7.519691	313.344198	433.444665	15.909471	66.396293	4.498069
...	...	...	...	...	...	...	...	...	...
1811	7.080795	247.164018	18939.616461	7.086375	305.040026	419.453088	12.758114	60.319203	4.287441
1123	5.803497	193.200991	19451.767603	4.146601	255.976746	365.477618	14.920616	8.577013	2.181714
515	5.701155	233.515043	41411.601707	5.895464	310.160545	509.767888	22.686837	73.751883	3.403136

In [25]:
```python
x_train
```

Out[25]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity
3216	5.609235	216.122673	14825.934502	7.001788	355.211678	330.092674	9.779518	41.249977	3.224676
3050	8.479593	200.765679	14901.808144	7.379506	333.775777	485.093167	16.806347	53.134170	4.221266
739	6.772157	196.900098	13790.296202	9.575053	333.775777	347.588375	17.705727	68.379815	3.827256
2100	4.894278	184.552715	10922.541994	7.461703	352.830222	338.681069	21.624718	91.007934	3.594991
2035	5.763773	183.073629	22025.696606	8.952896	333.775777	376.581389	10.547398	63.299757	3.253200
...	...	...	...	...	...	...	...	...	...
2347	5.429335	183.439383	15265.407564	5.714731	394.001195	446.879149	17.581557	50.266951	3.081736
1608	6.919726	194.859000	35558.731647	6.371978	299.786711	394.829326	11.983961	69.853135	4.120828
2541	5.735724	158.318741	25363.016594	7.728601	377.543291	568.304671	13.626624	75.952337	4.732954
2575	6.914868	206.249937	10343.378848	7.771206	324.383170	521.320673	16.173730	68.246945	2.988611
3240	6.102307	205.999038	23406.859160	8.061864	333.775777	380.305005	15.492397	69.110815	4.280766

2620 rows × 9 columns

A Random Forest classifier is instantiated, and the model is trained using the training data (x_train and y_train). This step prepares the model for making predictions

Random Forest Model

In [28]:
```python
#Random Forest
#creating model object
model_rf=RandomForestClassifier()

```

In [29]:
```python
#training model RF
model_rf.fit(x_train,y_train)
```

Out[29]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

The trained Random Forest model is used to make predictions on the test set (x_test). The accuracy of the predictions is then calculated using the accuracy_score function, comparing the predicted labels (pred_rf) with the true labels (y_test). The accuracy score is printed as a percentage.

Prediction and Accuracy

```
In [30]:
#Making Prediction
pred_rf=model_rf.predict(x_test)
```

```
In [31]:
accuracy_score_rf=accuracy_score(y_test,pred_rf)
accuracy_score_rf*100

Out[31]: 69.51219512195121
```

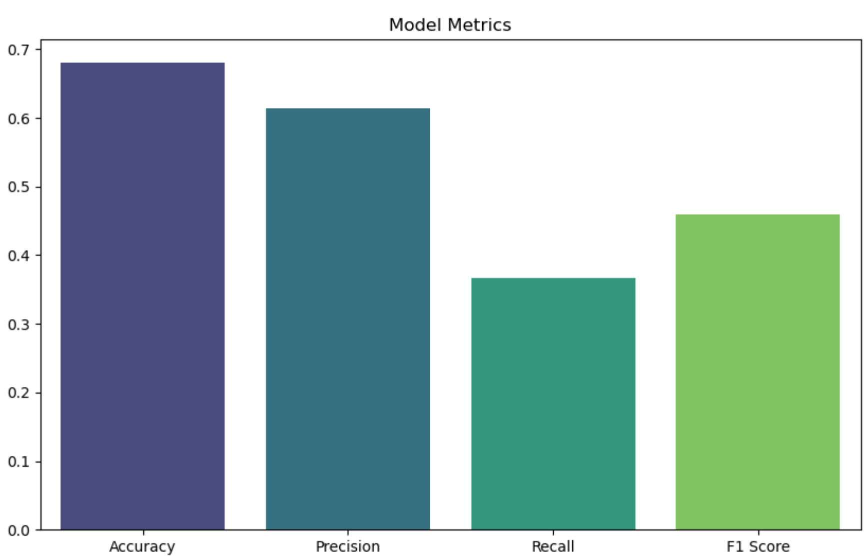
Evaluation:

We can evaluate the model's performance using metrics such as accuracy, precision, recall, and F1 score. This step ensures the reliability and effectiveness of the developed model.

```
In [34]:
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

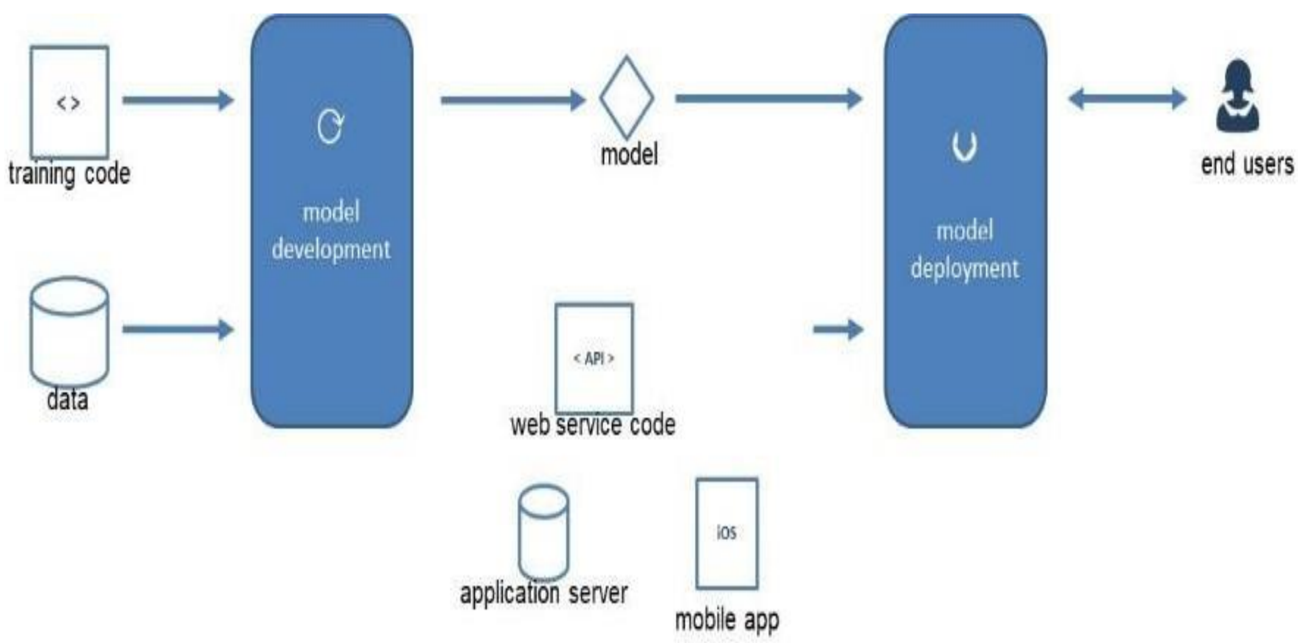
```
In [35]:
# Calculating metrics
accuracy_score_rf = accuracy_score(y_test, pred_rf)
precision_rf = precision_score(y_test, pred_rf)
recall_rf = recall_score(y_test, pred_rf)
f1_rf = f1_score(y_test, pred_rf)
```

```
In [36]:
print(f"Accuracy: {accuracy_score_rf*100:.2f}%")
print(f"Precision: {precision_rf:.2f}")
print(f"Recall: {recall_rf:.2f}")
print(f"F1 Score: {f1_rf:.2f}")

Accuracy: 67.99%
Precision: 0.61
Recall: 0.37
F1 Score: 0.46
```



Final Product Prototype:

The Water Quality Prediction System prototype is a sophisticated solution catering to the needs of municipalities, water treatment facilities, and environmental agencies. This system seamlessly integrates real-time data from water quality sensors with advanced machine learning algorithms to predict crucial parameters. The user interface includes a visually intuitive dashboard, offering insights into both real-time and historical water quality trends.. Notably, an alerting system has been implemented, providing timely notifications to relevant authorities when predicted values surpass predefined thresholds. Cloud integration enhances scalability, making this prototype a robust and adaptable tool for sustainable water resource management.



Business Model:

The Water Quality Prediction System prototype is a comprehensive solution designed for municipalities, water treatment facilities, and environmental agencies. It combines real-time data from water quality sensors with advanced machine learning algorithms to predict crucial parameters. The prototype includes:

1. Seamless Integration:
   - Integrates real-time data from water quality sensors with advanced machine learning algorithms.

2. User-Friendly Interface:
   - A visually intuitive dashboard providing insights into real-time and historical water quality trends.

3. Alerting System:
   - Implements an alerting system to notify relevant authorities promptly when predicted values surpass predefined thresholds.

4. Cloud Integration:
   - Utilizes cloud integration for scalability, ensuring the system is robust and adaptable for sustainable water resource management.

MARKET ANALYSIS:

The market for water quality prediction systems is significant, with municipalities, water treatment facilities, and environmental agencies increasingly recognizing the importance of proactive water resource management. The prototype addresses a gap in the market for a sophisticated, integrated solution.

OPERATING PLAN:

Deployment:
- Collaborate with municipalities, water treatment facilities, and environmental agencies for prototype deployment.

Customization:
- Offer customization based on specific client needs, ensuring the system aligns with their unique requirements.

Training:
- Provide comprehensive training for end-users to maximize the benefits of the Water Quality Prediction System.

Support:
- Establish a robust support system for ongoing assistance and system maintenance.

PRICING STRATEGY:

Subscription Model:
- Implement a subscription-based pricing model, offering tiered plans based on the scale of usage and additional features.

Customization Fees:
- Charge additional fees for customization based on specific client requirements.

MARKETING PLAN:

Target Audience:
- Focus marketing efforts on municipalities, water treatment facilities, and environmental agencies.

Educational Content:
- Create educational content highlighting the benefits of proactive water quality prediction and management.

Demonstrations:
- Conduct live demonstrations and workshops to showcase the capabilities of the Water Quality Prediction System.

Online Presence:
- Establish a strong online presence through a dedicated website, social media, and targeted advertising.

The Water Quality Prediction System prototype offers a sophisticated, integrated solution to address the evolving needs of water resource management. With a focus on customization, training, and ongoing support, the business model aims to establish itself as a key player in the market, providing valuable insights and actionable predictions for sustainable water management.

Financial Equation:

Let's assume that the duration of developing the ML model takes about 1 to 3 weeks, and the cost for producing the model is the sum of the team members' salaries. Consider two ML engineers with a salary denoted as 'ml' and one full-stack web developer with a salary denoted as 'fs'. So, the total cost c is given by the equation c = 2 * ml + fs. The profit or financial equation will look like this:

$$y = 5000*x(t) – (2*ml+fs)$$

- x(t) represent the growth of the customer base (let's say it's 100 for this example).
- ml be the salary of one ML engineer (let's say it's $80,000 per year).
- fs be the salary of one full-stack web developer (let's say it's $90,000 per year).

The total cost (c) for developing and maintaining the ML model is given by:
c = 2 *80,000 + 90,000

Now, let's substitute these values into the profit equation:
y = 5000*100 - (2 * 80,000 + 90,000)

Simplifying the equation:
y = 500,000 - (160,000 + 90,000)

y = 500,000 - 250,000

y = 250,000

So, the financial equation for your water quality check ML model, based on the provided values, is:
y = 250,000

This means that with a customer base growth of 100, the profit for your ML model would be $250,000.

Conclusion:

In conclusion, the water quality prediction project not only taps into a thriving market for innovative water management solutions but also strategically monetizes its offerings. Targeting municipalities and water treatment facilities, the project positions itself as a proactive, cost-effective, and customizable solution. The monetization strategy, including licensing, subscription services, and customization options, ensures a sustainable revenue stream. By addressing crucial needs in water quality management with a focus on regulatory compliance and sustainability, the project emerges as a valuable and financially rewarding asset in the evolving landscape of water resource management.

GitHub link:

“ https://github.com/711Rakesh/Water-quality-prediction-using-machine-learning.git ”