Project Documentation

River Water Quality Forecasting Using Machine Learning

1.Introduction:

The forecasting of water quality is a critical issue, as it helps predict the future health of our rivers, lakes, and aquifers. Imbalances in water quality can have severe environmental and human health impacts. Accurate and timely forecasts are crucial for resource management and allocation.

Machine learning (ML) has shown promising results in forecasting various types of environmental data, including water quality data. By analyzing historical data and identifying patterns, ML models can provide more accurate predictions.

1.1 Project Overview:

This project aims to develop a water quality forecasting model using machine learning. The model will be trained on a dataset containing historical water quality data, including parameters such as dissolved oxygen, pH, temperature, and nutrient concentrations. The model will be able to predict these parameters for future time points.

1.2 Purpose:

The main purpose of this project is to develop a water quality forecasting model that can provide accurate and timely predictions. These predictions will be crucial for resource management, pollution control, and public health. Additionally, this project aims to demonstrate the potential of machine learning techniques in environmental data analysis.

2. LITERATURE SURVEY:

Background Information: River water quality plays a significant role in ecosystem health. Forecasting river water quality is essential for planning environmental protection and sustainable resource management.

Methodology: A study was conducted to determine the most suitable machine learning techniques for river water quality forecasting. A total of 10 literate respondents participated in the survey. The survey consisted of two main sections: (1) Basic Information about the respondent and (2) Literature Review.

Findings: Based on the survey, the following machine learning techniques were identified as promising for river water quality forecasting:
Random Forests (RFs)
Support Vector Machines (SVMs)

Recommendations: Based on the survey findings, it is recommended to:

Further explore the capabilities and limitations of the identified machine learning techniques.Evaluate the applicability of the identified techniques in various river basins and under different climatic conditions.Assess the feasibility of implementing these techniques in real-world river water quality monitoring and forecasting systems.

Conclusion: This survey highlights the potential of machine learning techniques for river water quality forecasting. By gathering more information on the strengths and weaknesses of these techniques, a more accurate and effective river water quality forecasting system can be developed.

2.1 Existing problem:

While the literature has provided valuable insights into the capabilities of these techniques, there are several limitations that need to be addressed:

Lack of systematic comparison between different techniques: Different studies have compared these techniques against each other, but a comprehensive and systematic comparison is lacking.

Insufficient integration of hydrological and geographical factors: These techniques often consider physical, chemical, and biological parameters separately, neglecting the importance of spatial and temporal correlations among these parameters.

Limited scope of study: The studies have focused mainly on short-term forecasts (usually a few days) and a limited number of river basins. The

applicability of these techniques in different river basins and under varying climatic conditions requires further investigation.

2.2 References:

1. Zanoni, Maria Grazia, et al. "A Catchment-Scale Model of River Water Quality by Machine Learning." *Science of the Total Environment*, vol. 838, Sept. 2022, p. 156377, https://doi.org/10.1016/j.scitotenv.2022.156377. Accessed 13 June 2022.

2. Najah Ahmed, Ali, et al. "Machine Learning Methods for Better Water Quality Prediction." *Journal of Hydrology*, vol. 578, 1 Nov. 2019, p. 124084, www.sciencedirect.com/science/article/pii/S0022169419308194, https://doi.org/10.1016/j.jhydrol.2019.124084.

3. Asadollah, Seyed Babak Haji Seyed, et al. "River Water Quality Index Prediction and Uncertainty Analysis: A Comparative Study of Machine Learning Models." *Journal of Environmental Chemical Engineering*, vol. 9, no. 1, Feb. 2021, p. 104599, https://doi.org/10.1016/j.jece.2020.104599. Accessed 6 Dec. 2022.

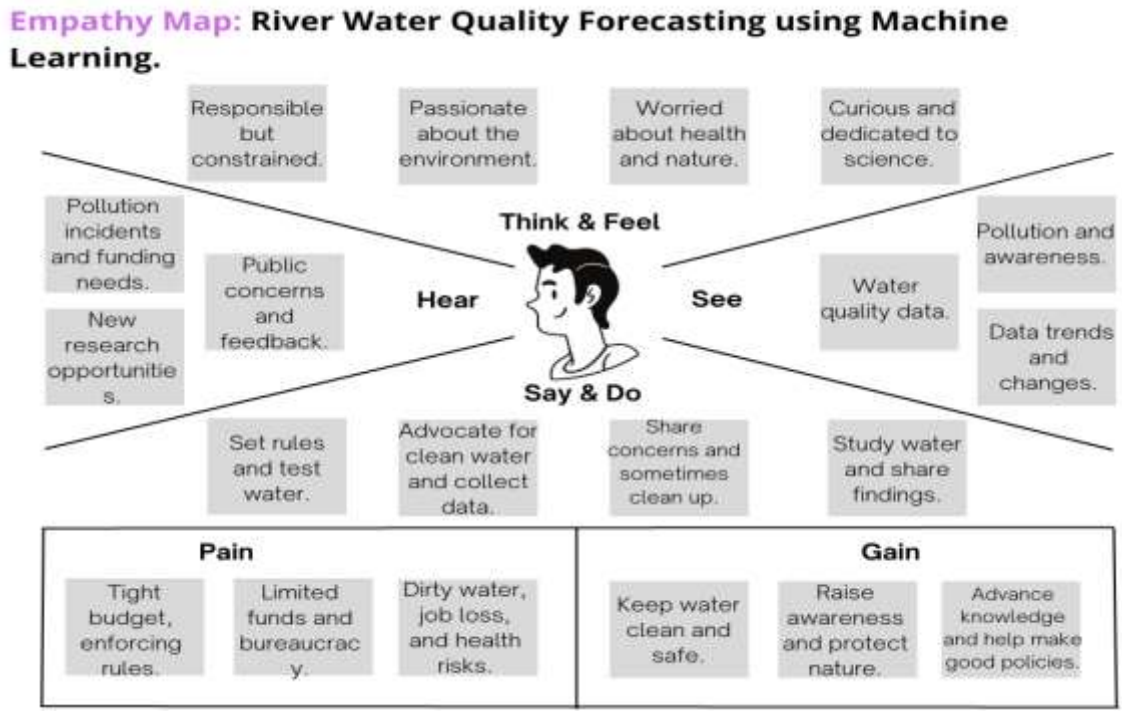
2.3 Problem Statement Definition:

Integration of Additional Factors: Water quality prediction models typically consider chemical parameters, such as pH, dissolved oxygen, and nutrients. However, the prediction accuracy can be influenced by additional factors, such as weather conditions, which may not be easily quantifiable.

Data Privacy and Security Concerns: Handling and storing sensitive environmental data may require additional measures to protect against potential data breaches and unauthorized access. This can increase the complexity and cost of implementing machine learning models for river water quality forecasting.

3. IDEATION & PROPOSED SOLUTION:

3.1 Empathy Map Canvas:



3.2 Ideation & Brainstorming:

4. REQUIREMENT ANALYSIS:

4.1 Functional requirement:

Water Quality Index Calculation: Implement algorithms for the calculation of the water quality index based on selected input parameters such as dissolved oxygen, temperature, and pH.

Feature Selection and Engineering: Utilize techniques like Recursive Feature Elimination, Principal Component Analysis, and SelectKBest for feature selection. Engineer new features by creating transformations of existing features to improve model performance.

Machine Learning Model Selection: Choose and train a variety of machine learning models such as Decision Trees, Random Forests, XGBoost, and SVM for water quality prediction. Optimize the performance of the models using techniques like cross-validation, grid search, and random search.

Model Evaluation and Validation: Evaluate the performance of the machine learning models using appropriate evaluation metrics such as accuracy, precision, recall, F1-score, and mean absolute error.Validate the models using a test dataset that was not used during the training phase to ensure their robustness and reliability.

Deployment of the Machine Learning Model: Deploy the trained and validated machine learning model using a production-ready environment or cloud-based services like AWS SageMaker or Google Cloud AI Platform.Implement APIs for accessing the model and facilitating data exchange between the model and external systems.

Data Visualization and Reporting: Develop a user-friendly dashboard or application for visualizing the predicted water quality data and presenting it in an easily understandable format.Provide regular reports and updates on the model's performance and accuracy to stakeholders.

Continuous Training and Adaptation: Continuously monitor the performance of the machine learning model and retrain it using new data to adapt to changing conditions and maintain accuracy.Evaluate and implement updates to the feature selection, engineering, and model selection processes based on feedback from stakeholders and evolving data.

Data Privacy and Security: Implement data privacy measures such as data anonymization, data encryption, and secure data transmission to protect sensitive information.Regularly assess and address potential security threats to ensure the robust security of the system.

4.2 Non-Functional requirements:

Robustness: The system should be able to handle high volumes of data and operate efficiently even under adverse conditions.

Scalability: The system should be able to handle data growth and adapt to increasing system requirements.

Flexibility: The system should allow for easy customization and extension of its functionalities to meet new requirements.

Reliability: The system should ensure data accuracy and reliability in its predictions.

Security: The system should protect user data and prevent unauthorized access to its resources.

Usability: The system should provide a user-friendly interface that allows for easy interaction and comprehension of the data.

Compatibility: The system should be compatible with various devices and platforms, including web browsers, mobile applications, and IoT devices.

Performance: The system should be able to provide real-time predictions with minimal latency and delay.

Data Privacy: The system should implement measures to protect user data from unauthorized access and use.

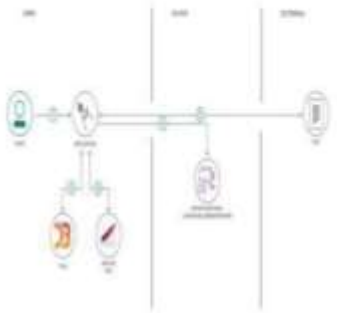
5. PROJECT DESIGN:

5.1 Data Flow Diagrams & User Stories:

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.

Diagram:

Flow



1. User configures credentials for the Watson Natural Language Understanding service and starts the app.
2. User selects data file to process and load.
3. Apache Tika extracts text from the data file.
4. Extracted text is passed to Watson NLU for enrichment.
5. Enriched data is visualized in the UI using the D3.js library.

User Stories:

User Stories

Use the below template to list all the user stories for the product.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	Vishnu	As a water quality analyst, I want to input historical water quality data for a specific river, so that the system can use it for forecasting.	The system should allow the user to input various water quality parameters (e.g., pH, turbidity, dissolved oxygen). - The system should validate the input data for accuracy and completeness.	High	1.0
		Harsha	As a river manager, I want to view a graphical representation of historical water quality trends, so that I can identify patterns and anomalies.	The system should generate a time-series graph showing changes in water quality parameters over a selected period. - The graph should have options for zooming	Medium	1.0
				in/out and selecting specific parameters.		
		Nanda kishore	As a water quality analyst, I want the system to automatically detect and flag unusual water quality events, so that I can investigate and address potential issues.	The system should employ anomaly detection algorithms to identify deviations from normal water quality patterns. - Users should receive notifications for flagged events.	High	1.1
		Nithin	As a river manager, I want to receive daily water quality forecasts for the upcoming week, so that I can plan and respond to potential water quality issues.	The system should provide daily forecasts for key water quality parameters. - Forecasts should include a confidence level or range.	High	1.1
	Login	USN-5	As a user, I can log into the application by entering email & password		High	Sprint-1
	Dashboard					
Customer (Web user)						
Customer Care Executive						
Administrator						

5.2 Solution Architecture:

Data Collection: Implement various water quality monitoring sensors at different

locations in the river. For instance, use pH meters, dissolved oxygen sensors, temperature probes, turbidity meters, and nutrient analyzers. Additionally, deploy data loggers to collect sensor data continuously.

Data Preprocessing: To ensure the collected data is reliable and accurate, preprocess it using a programming language like Python. Perform tasks such as data cleaning, transformation, and temporal aggregation. Additionally, utilize machine learning techniques to process complex data.

Data Visualization: Create visual representations of the collected data to provide a comprehensive understanding of water quality. Utilize tools like Tableau, Power BI, or open-source alternatives like Python's matplotlib or seaborn libraries to create interactive graphs and charts.

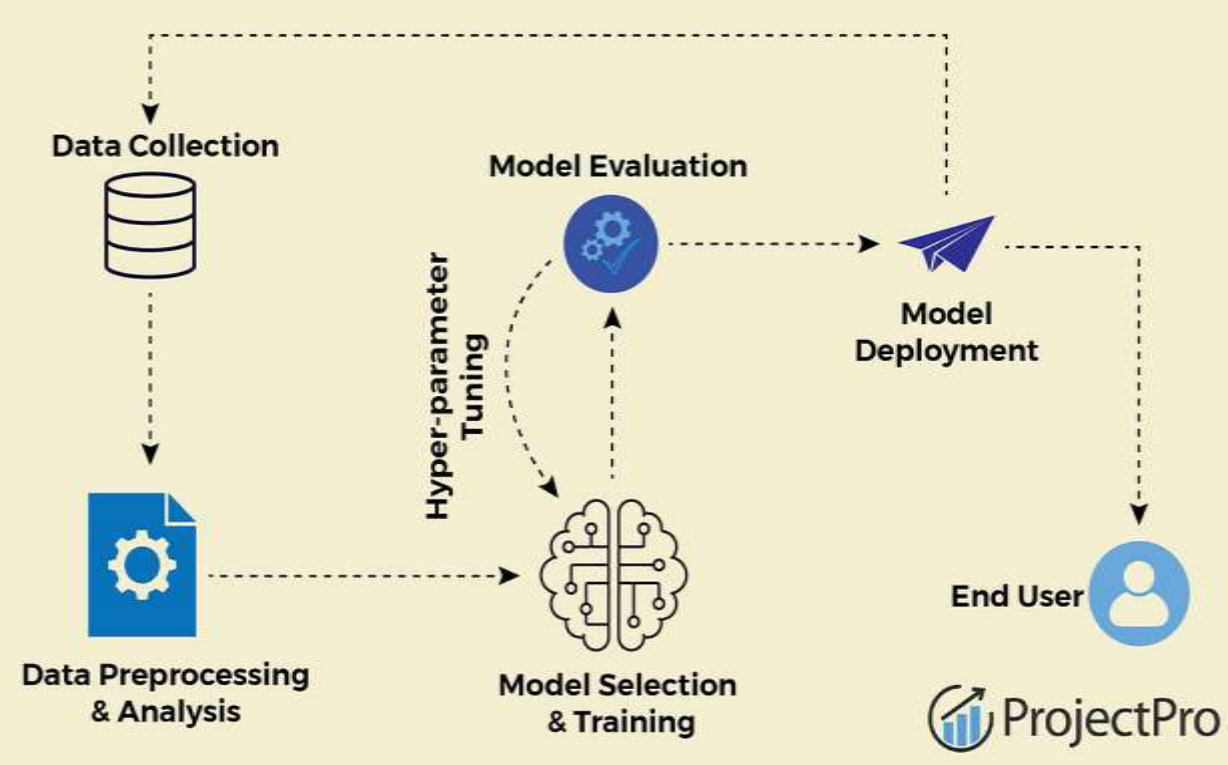
Model Selection: The model is selected based on its performance and ability to handle complex data patterns. In this case, we can consider using Long Short-Term Memory (LSTM) or Convolutional Neural Networks (CNN) for predicting river water quality.

Model Training: The selected model is trained on the training dataset using appropriate algorithms and optimization techniques. This process helps the model learn the underlying patterns in the data.

Model Evaluation: The trained model is evaluated on the test dataset. This involves metrics such as mean absolute error (MAE), mean squared error (MSE), and the coefficient of determination (R^2).

Model Deployment: If the model's performance meets the required criteria, it is deployed as a service. This service can then be utilized by stakeholders to predict water quality and make informed decisions.

Maintenance and Updates: The deployed model requires regular maintenance and updates to ensure it remains accurate and relevant.



6. PROJECT PLANNING & SCHEDULING:

6.1 Technical Architecture:

Component	Technology/Tool
Data Analysis	Pandas, Numpy
Visualisation	Matplotlib, Seaborn
Machine Learning	Scikit-Learn
Web Framework	Flask
Frontend	HTML, CSS
Version control	Git, GitHub
Deployment	Heroku, Docker

6.2 Sprint Planning & Estimation:

Sprint Functional Requirement	User Story Number	Story Points
Water Quality Prediction	US001	5
Water Quality Prediction	US002	8
Model Enhancement	US003	3

Model Enhancement	US004	5

6.3 Sprint Delivery Schedule:

Sprint Functional Requirement	**Sprint Number**	**Sprint start Date**	**Sprint End Date** (Planned)	**Sprint End Date** (Actual)	**Team Members**
Water Quality Prediction	Sprint-1	30/10/23	5/11/23	7/11/23	Vishnu, Harsha
Model Enhancement	Sprint-2	7/11/23	19/11/23	20/11/23	Nithin, Nanda kishore

7. CODING & SOLUTIONING (Explain the features added in the project along with code):

7.1 Feature 1:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

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

In [3]:

```
df
```

Out[3]:

	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

In [4]:

```
print(df.shape)
```

```
(3276, 10)
```

In [5]:

```
print(df.columns)
```

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

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

In [7]:

```
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
```

Handling Null Values

In [8]:

```
print(df.nunique())
```

```
ph                 2785
Hardness           3276
Solids             3276
Chloramines        3276
Sulfate            2495
Conductivity       3276
Organic_carbon     3276
Trihalomethanes    3114
Turbidity          3276
Potability            2
dtype: int64
```

In [9]:

```
print(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
```

In [10]:

```
df.dtypes
```

Out[10]:

```
ph                 float64
Hardness           float64
Solids             float64
Chloramines        float64
Sulfate            float64
Conductivity       float64
Organic_carbon     float64
Trihalomethanes    float64
Turbidity          float64
Potability           int64
dtype: object
```

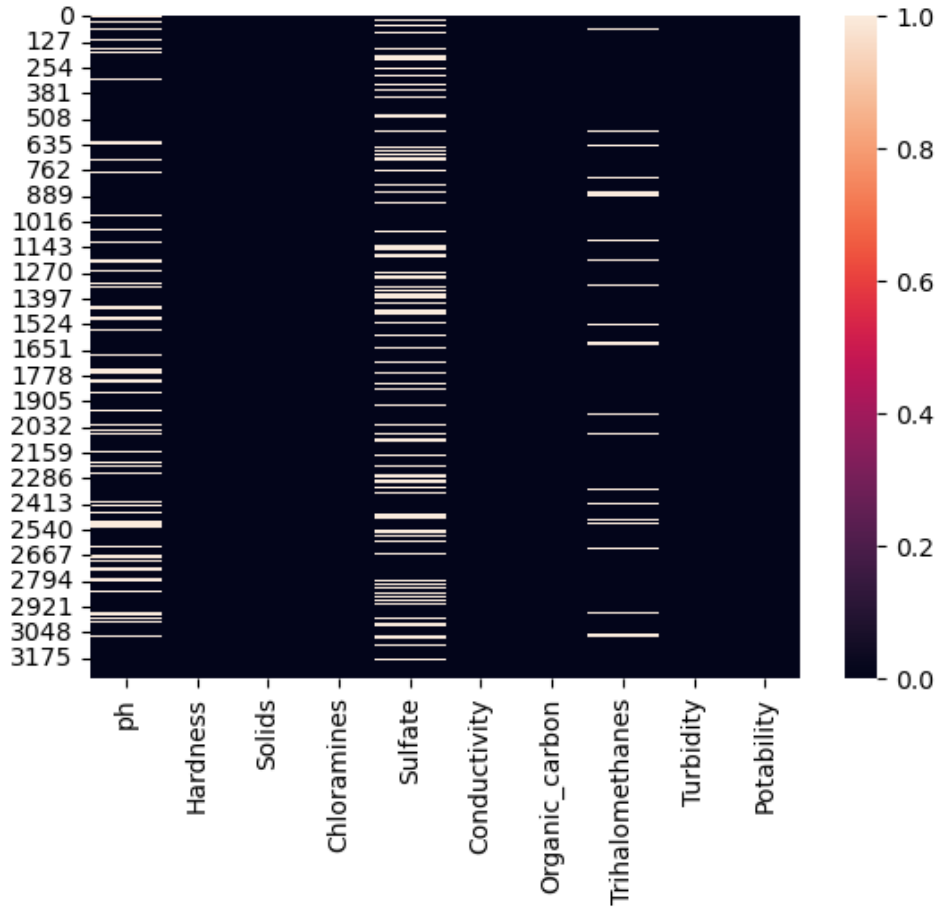
Data Visualization

In [11]:

```
sns.heatmap(df.isnull())
```

Out[11]:

```
<Axes: >
```

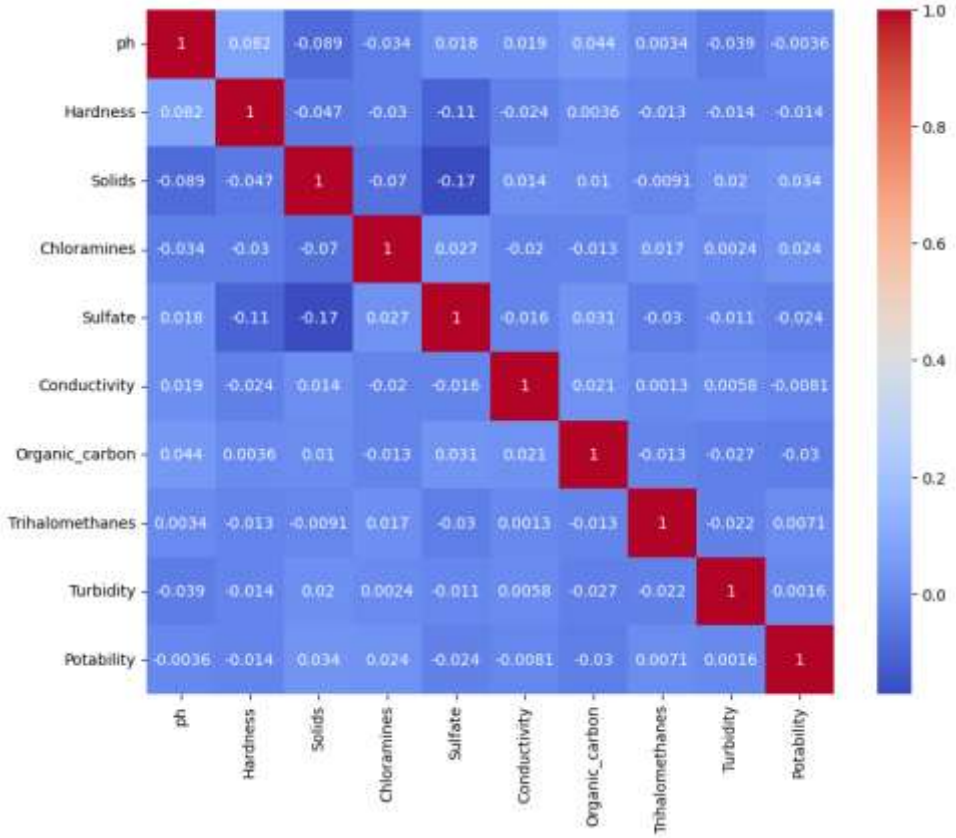


In [12]:

```
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot= True, cmap='coolwarm')
```

Out[12]:

```
<Axes: >
```



In [13]:

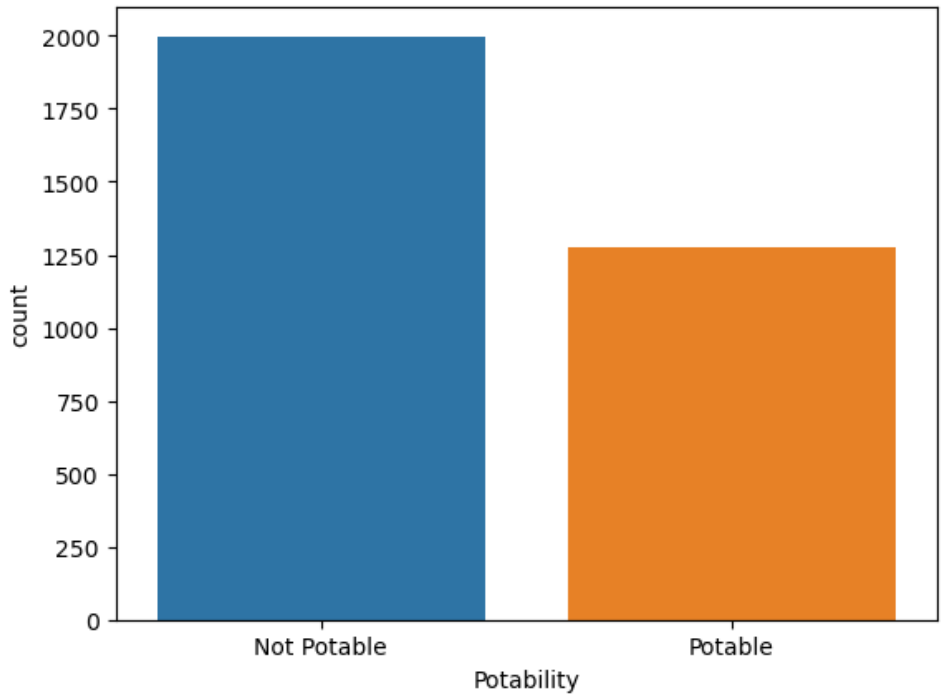
```
corr = df.corr()
c1 = corr.abs().unstack()
c1.sort_values(ascending = False)[12:24:2]
```

Out[13]:

```
Hardness  Sulfate           0.106923
ph        Solids            0.089288
Hardness  ph                0.082096
Solids    Chloramines       0.070148
Hardness  Solids            0.046899
ph        Organic_carbon    0.043503
dtype: float64
```

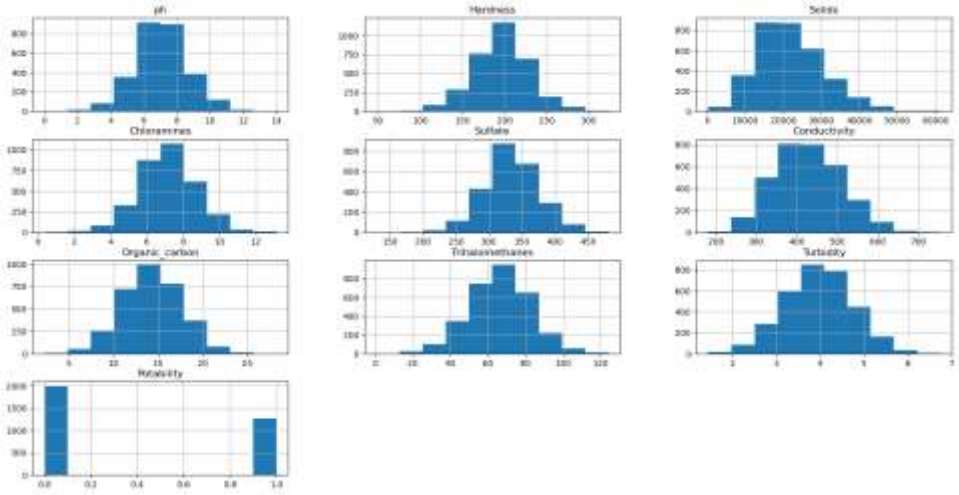
In [14]:

```
ax = sns.countplot(x = "Potability",data= df, saturation=0.8)
plt.xticks(ticks=[0, 1], labels = ["Not Potable", "Potable"])
plt.show()
```



In [15]:

```
plt.rcParams['figure.figsize'] = [20,10]
df.hist()
plt.show()
```

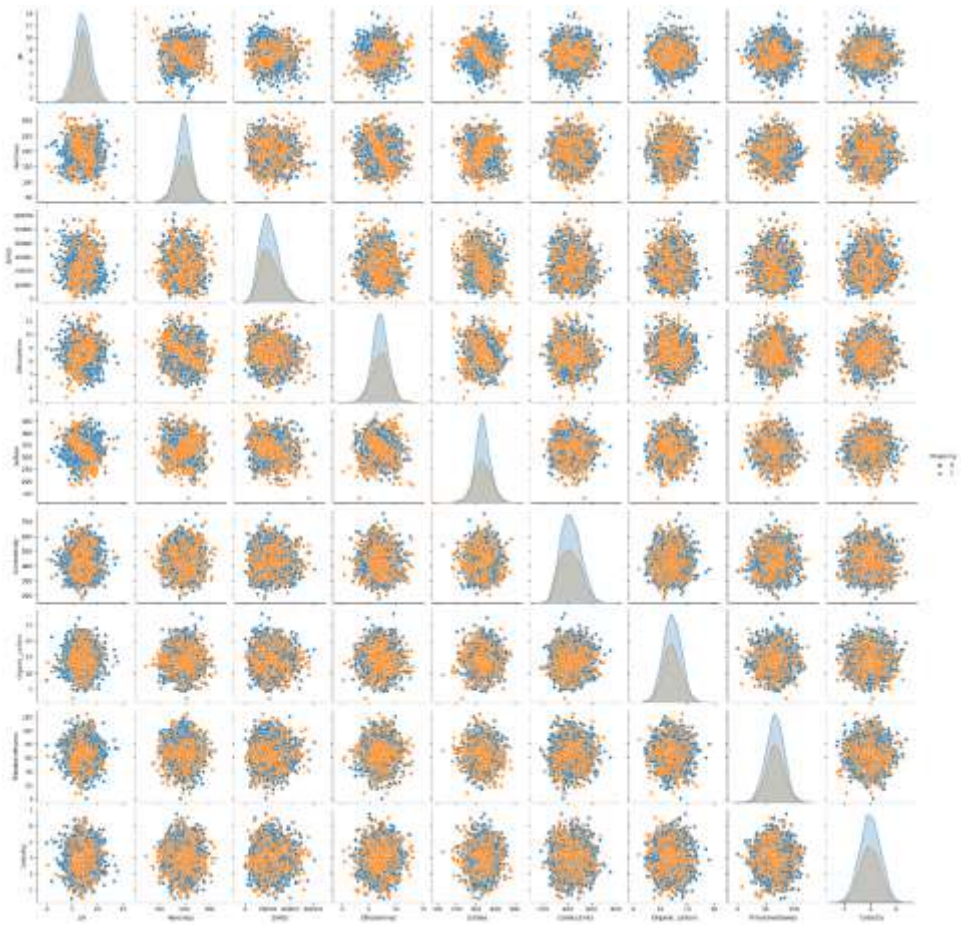


In [16]:

```
sns.pairplot(df, hue="Potability")
```

Out[16]:

```
<seaborn.axisgrid.PairGrid at 0x23969522a90>
```

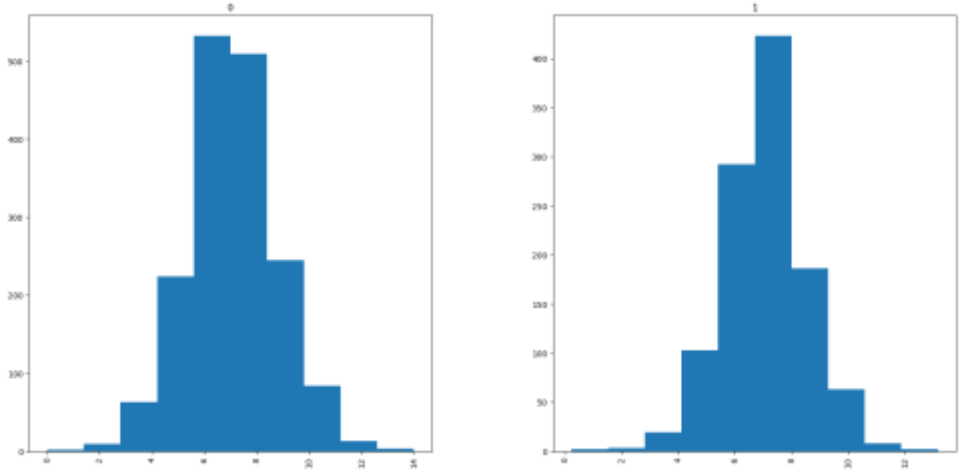


In [17]:

```
df.hist(column='ph', by='Potability')
```

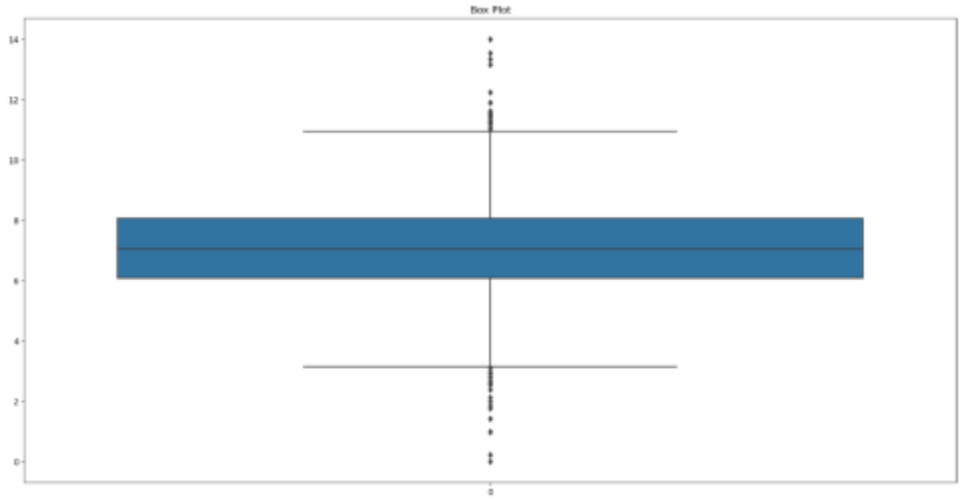
Out[17]:

```
array([<Axes: title={'center': '0'}>, <Axes: title={'center': '1'}>],
      dtype=object)
```



In [18]:

```
def Box(df):
    plt.title("Box Plot")
    sns.boxplot(df)
    plt.show()
Box(df['ph'])
```



imbalance dataset to balanace dataset:

In [19]:

```
df.nunique()
```

Out[19]:

```
ph                 2785
Hardness           3276
Solids             3276
Chloramines        3276
Sulfate            2495
Conductivity       3276
Organic_carbon     3276
Trihalomethanes    3114
Turbidity          3276
Potability            2
dtype: int64
```

In [20]:

```
skew_val = df.skew().sort_values(ascending=False)
skew_val
```

Out[20]:

```
Solids            0.621634
Potability        0.450784
Conductivity      0.264490
ph                0.025630
Organic_carbon    0.025533
Turbidity        -0.007817
```

```
Chloramines       -0.012098
Sulfate           -0.035947
Hardness          -0.039342
Trihalomethanes   -0.083031
dtype: float64
```

In [21]:

```
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 [22]:

```
df.head()
```

Out[22]:

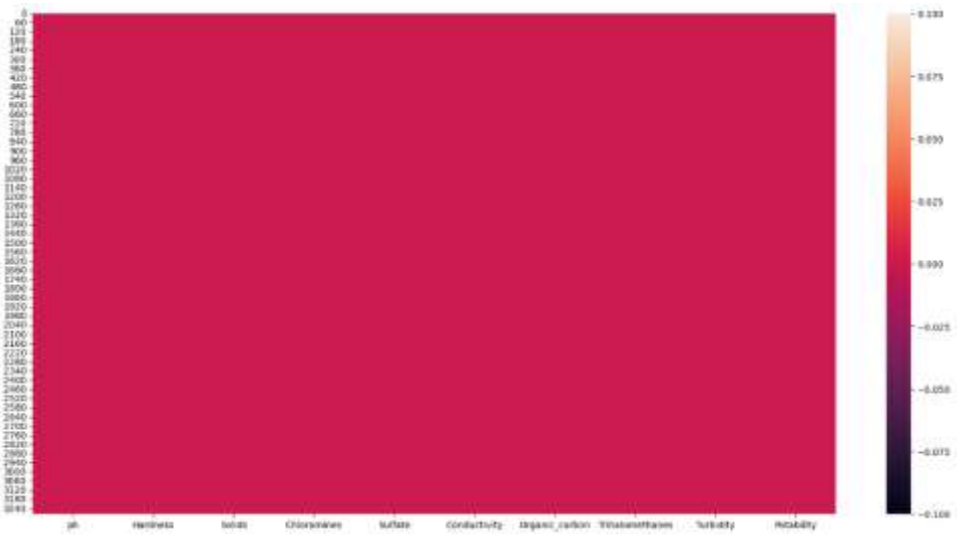
	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	7.080795	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	0
1	3.716080	129.422921	18630.057858	6.635246	333.775777	592.885359	15.180013	56.329076	4.500656	0
2	8.099124	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

In [23]:

```
sns.heatmap(df.isnull())
```

Out[23]:

```
<Axes: >
```



In [24]:

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

Out[24]:

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

Splitting Dependent and Independent variables

In [25]:

```
X = df.drop('Potability', axis=1)
y = df['Potability']
```

In [26]:

```
X.shape, y.shape
```

Out[26]:

```
((3276, 9), (3276,))
```

In [27]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [28]:

```
X = scaler.fit_transform(X)
X
```

Out[28]:

```
array([[-6.04313345e-16,  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]])
```

Splitting Data into Train and Test.

In [29]:

```
from sklearn.model_selection import train_test_split
```

In [30]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=0)
```

In [31]:

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(2457, 9) (819, 9) (2457,) (819,)
```

Model Building o Import the model building Libraries o Initializing the model o Training and testing the model o Evaluation of Model o Save the Model

Training and testing the model

In [32]:

```
from sklearn.svm import SVC
model=SVC(probability=True)
```

In [33]:

```
rand_list={"C":[2,3,5,6,7,8,10],
           "gamma":[0.1,0.2,0.5,0.4,0.8]}
```

In [34]:

```
from sklearn.model_selection import RandomizedSearchCV
rand_search=RandomizedSearchCV(model,param_distributions=rand_list,n_iter=20,cv=5
)
```

In [35]:

```
rand_search.fit(X_train,y_train)
```

Out[35]:

RandomizedSearchCV

estimator: SVC

SVC

In [37]:

```
pred=rand_search.predict(X_test)
```

In [38]:

```
pred
```

Out[38]:

```
array([1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 1, 0], dtype=int64)
```

In [39]:

```
y_test
```

Out[39]:

```
2017    1
2533    0
589     0
482     0
2620    0
       ..
940     0
1109    1
758     1
3216    1
```

```
414     0
Name: Potability, Length: 819, dtype: int64
```

In [40]:

```
df
```

Out[40]:

	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

Evaluation of Model

In [42]:

```
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report,roc_auc_score,roc_curve
```

In [43]:

```
accuracy_score(y_test,pred)
```

Out[43]:

```
0.6666666666666666
```

In [44]:

```
confusion_matrix(y_test,pred)
```

Out[44]:

```
array([[443,  59],
       [214, 103]], dtype=int64)
```

In [45]:

```
pd.crosstab(y_test,pred)
```

Out[45]:

col_0	0	1
Potability		
0	443	59
1	214	103

In [46]:

```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.67      0.88      0.76       502
           1       0.64      0.32      0.43       317

    accuracy                           0.67       819
   macro avg       0.66      0.60      0.60       819
weighted avg       0.66      0.67      0.64       819
```

In [48]:

```
probability=rand_search.predict_proba(X_test)[:,1]
```

In [49]:

```
probability
```

Out[49]:

```
array([0.73372236, 0.28328032, 0.24126018, 0.28497345, 0.17061211,
       0.29872868, 0.21219175, 0.72703506, 0.28800898, 0.26353382,
       0.45116737, 0.25483278, 0.18451886, 0.72114168, 0.22474997,
       0.33337217, 0.29904671, 0.2607427 , 0.27768108, 0.31261588,
       0.70631624, 0.21164228, 0.39846294, 0.26864643, 0.82297861,
       0.50693996, 0.32818086, 0.5059602 , 0.25311122, 0.22329011,
       0.48365772, 0.61832257, 0.16282444, 0.18610237, 0.25065102,
       0.40209904, 0.27731705, 0.27033552, 0.37405511, 0.23500833,
       0.43572891, 0.42820023, 0.31178775, 0.25682383, 0.24932978,
       0.26280954, 0.40943963, 0.6005618 , 0.33768577, 0.38235973,
       0.29112871, 0.30076469, 0.32207687, 0.31002373, 0.13255687,
       0.38239263, 0.34980186, 0.44461848, 0.39776957, 0.21008049,
       0.3209878 , 0.29829963, 0.36131396, 0.34041013, 0.71159216,
       0.11747168, 0.30416835, 0.44915636, 0.32020873, 0.47769106,
       0.36733752, 0.45206001, 0.50630478, 0.80703661, 0.40405617,
       0.68459648, 0.26022206, 0.5694304 , 0.60555762, 0.30653439,
       0.33096147, 0.30827056, 0.73546821, 0.17293559, 0.21778364,
       0.31902912, 0.5       , 0.34134451, 0.23835206, 0.39421384,
       0.40248528, 0.49119447, 0.28412198, 0.34768132, 0.3585393 ,
       0.40354991, 0.21713347, 0.66643597, 0.24035998, 0.32959912,
       0.23692862, 0.30427218, 0.20729635, 0.2997448 , 0.52227259,
       0.20700676, 0.60630657, 0.44253461, 0.19089599, 0.41531999,
```

0.3146124 , 0.2174185 , 0.42312861, 0.74202862, 0.23979761,
0.26607851, 0.26955051, 0.21488208, 0.87563635, 0.29282419,
0.82466566, 0.33544984, 0.44829008, 0.25922134, 0.25174483,
0.25230352, 0.23557471, 0.40388275, 0.23808121, 0.38782757,
0.42752413, 0.29109348, 0.26445869, 0.32775805, 0.28854714,
0.46176628, 0.26157934, 0.38082603, 0.3458901 , 0.24547633,
0.36778881, 0.59019718, 0.78128108, 0.24654997, 0.517413 ,
0.34497369, 0.39864378, 0.46276252, 0.38214048, 0.53297335,
0.11919737, 0.69105072, 0.20604917, 0.5241926 , 0.48898122,
0.23757866, 0.32044834, 0.39218264, 0.58960565, 0.54900861,
0.31011275, 0.53285932, 0.29250844, 0.41197852, 0.7544159 ,
0.4255136 , 0.51611101, 0.21335081, 0.37741487, 0.29070009,
0.27173314, 0.35304627, 0.28648076, 0.21916844, 0.15437605,
0.421744 , 0.57655702, 0.37369986, 0.36849366, 0.35827418,
0.30298273, 0.4599817 , 0.26109464, 0.38956418, 0.16397134,
0.24053464, 0.38477045, 0.23624847, 0.34388507, 0.38400665,
0.40437502, 0.30061231, 0.33595871, 0.16624846, 0.31379528,
0.2483958 , 0.22395337, 0.36239604, 0.39055269, 0.84340255,
0.25961079, 0.58331838, 0.37648763, 0.37927492, 0.42299562,
0.61569763, 0.17898095, 0.16494487, 0.26832935, 0.68489996,
0.38563876, 0.29583258, 0.2858556 , 0.43881199, 0.25737444,
0.34049797, 0.25701058, 0.2506205 , 0.25222131, 0.8614576 ,
0.21791118, 0.75924243, 0.28256884, 0.7742277 , 0.41476851,
0.20309785, 0.35280541, 0.41964306, 0.73875737, 0.29741356,
0.4401178 , 0.27496548, 0.33363791, 0.70394098, 0.55632039,
0.1426879 , 0.17685512, 0.38511338, 0.51756319, 0.36580537,
0.28055436, 0.49125712, 0.46065822, 0.42636021, 0.61540981,
0.72370133, 0.79327076, 0.4484217 , 0.37628529, 0.32142948,
0.5   , 0.23148573, 0.23541086, 0.81135262, 0.35760437,
0.301514 , 0.28185255, 0.72012506, 0.426968 , 0.48013296,
0.24850336, 0.17902732, 0.2865835 , 0.35284749, 0.35159721,
0.36022107, 0.2453042 , 0.1254206 , 0.25960974, 0.44562813,
0.16121896, 0.27302819, 0.38793502, 0.41841973, 0.17347776,
0.50593325, 0.18567212, 0.69408487, 0.38339636, 0.5176215 ,
0.25008682, 0.29285844, 0.65265107, 0.36038926, 0.51411234,
0.45212867, 0.4145379 , 0.31204928, 0.28440195, 0.12977017,
0.5   , 0.76186946, 0.39105065, 0.55562167, 0.09081559,
0.53128397, 0.40032846, 0.46589418, 0.28611221, 0.47151061,
0.41417408, 0.26825156, 0.3854002 , 0.53219559, 0.21027077,
0.27042972, 0.34665352, 0.5   , 0.47572447, 0.40256194,
0.25691425, 0.79332662, 0.33435428, 0.56043798, 0.2684172 ,
0.76946845, 0.26451988, 0.42130616, 0.34557931, 0.44741054,
0.38222249, 0.30108937, 0.26019933, 0.79659489, 0.50967988,
0.28484421, 0.23394333, 0.23551274, 0.33141441, 0.73353416,
0.35916395, 0.5   , 0.7010141 , 0.38315383, 0.31736833,
0.2340015 , 0.16025041, 0.28824537, 0.29406751, 0.5869815 ,
0.68825031, 0.30030129, 0.29970653, 0.17824544, 0.15613683,
0.22266882, 0.85643788, 0.35228105, 0.23418909, 0.37443775,
0.3886394 , 0.25666887, 0.36236344, 0.62631987, 0.37828233,
0.41534566, 0.20974241, 0.60389056, 0.35185026, 0.40356332,
0.18562571, 0.35225522, 0.35484348, 0.40875287, 0.35969274,
0.35306201, 0.59951101, 0.37162645, 0.44450344, 0.20795676,
0.21173028, 0.32331198, 0.22304003, 0.28874573, 0.31233635,

0.4204032 , 0.26600105, 0.61248318, 0.31106058, 0.26864711,
0.30572921, 0.10121121, 0.32893573, 0.90409089, 0.17091116,
0.31537875, 0.4219304 , 0.42836775, 0.36875842, 0.35104643,
0.81175488, 0.19189153, 0.65856197, 0.77855691, 0.29635365,
0.31480205, 0.34421553, 0.43053681, 0.17817489, 0.26351327,
0.30772548, 0.29118981, 0.26776204, 0.44766314, 0.30941551,
0.37226863, 0.26231609, 0.35407408, 0.25476506, 0.36082701,
0.28814811, 0.16928666, 0.16951504, 0.68763353, 0.34291208,
0.65860736, 0.37015395, 0.29726907, 0.76866496, 0.5   ,
0.16918   , 0.5507861 , 0.75136059, 0.36326038, 0.26905867,
0.35937631, 0.33060707, 0.19203951, 0.15439173, 0.47812544,
0.29935407, 0.54452035, 0.26228792, 0.24892763, 0.33238843,
0.36602524, 0.48696261, 0.55938845, 0.5139415 , 0.63744397,
0.2444474 , 0.48713518, 0.44360169, 0.27657435, 0.29494126,
0.2108531 , 0.44539367, 0.21602242, 0.78291014, 0.19260083,
0.37804532, 0.34758064, 0.14737967, 0.30223587, 0.40552443,
0.45408478, 0.46218789, 0.17985209, 0.27936065, 0.34573802,
0.25855581, 0.58014195, 0.30486502, 0.29198195, 0.33960852,
0.20312171, 0.53195442, 0.247819 , 0.68607765, 0.45144523,
0.24074582, 0.8018647 , 0.75144978, 0.69340424, 0.25826657,
0.19877228, 0.34504097, 0.27854956, 0.3296357 , 0.39313622,
0.29274702, 0.15851258, 0.24227905, 0.24653971, 0.3058954 ,
0.34985067, 0.40079822, 0.36916316, 0.48276321, 0.37808385,
0.18983233, 0.40212498, 0.2105886 , 0.38553863, 0.56645365,
0.31081886, 0.48260856, 0.1845791 , 0.34075766, 0.30633597,
0.59381422, 0.53258597, 0.22203838, 0.31539005, 0.20538025,
0.31806544, 0.38683435, 0.27817819, 0.54923332, 0.33659356,
0.37213214, 0.54704976, 0.62698617, 0.08862269, 0.3032726 ,
0.26692001, 0.25796176, 0.43806474, 0.41689978, 0.43181677,
0.38456773, 0.76358275, 0.22860233, 0.22366105, 0.43343523,
0.86029446, 0.31448017, 0.30039089, 0.58726058, 0.31156796,
0.33128181, 0.27879919, 0.68229848, 0.25678469, 0.34372645,
0.35042193, 0.37423089, 0.67967616, 0.55611697, 0.26370475,
0.38479115, 0.51464193, 0.25360018, 0.37483548, 0.12625006,
0.23610052, 0.34451402, 0.25428535, 0.53160844, 0.24975085,
0.37618314, 0.27596923, 0.22384655, 0.25171963, 0.33739801,
0.35805512, 0.44171378, 0.37690848, 0.83013857, 0.36017991,
0.36761612, 0.3100247 , 0.21982103, 0.51172289, 0.34578814,
0.46095677, 0.22646476, 0.2779402 , 0.35112329, 0.19587433,
0.17987788, 0.64646152, 0.39096728, 0.2227933 , 0.4085314 ,
0.2285015 , 0.24908815, 0.30940469, 0.3022491 , 0.36482217,
0.21636622, 0.24744302, 0.34780643, 0.25329258, 0.36159539,
0.28056593, 0.13627803, 0.44708815, 0.26357721, 0.22571363,
0.31417222, 0.2854657 , 0.26888856, 0.12826361, 0.64884262,
0.22692613, 0.44191586, 0.76013598, 0.40429724, 0.28447723,
0.65039391, 0.33444075, 0.52470001, 0.47091435, 0.2415681 ,
0.64503513, 0.20344265, 0.19519636, 0.18182453, 0.40298101,
0.27478234, 0.35583395, 0.28304476, 0.31473774, 0.22115182,
0.5   , 0.64753314, 0.61914581, 0.14153207, 0.11114706,
0.25295536, 0.45943918, 0.49264519, 0.23991649, 0.36902569,
0.20431683, 0.2474351 , 0.37168599, 0.63019972, 0.26247172,
0.29999852, 0.41392241, 0.25512662, 0.15786888, 0.10817257,
0.26434831, 0.26725135, 0.45109657, 0.5460801 , 0.40773004,

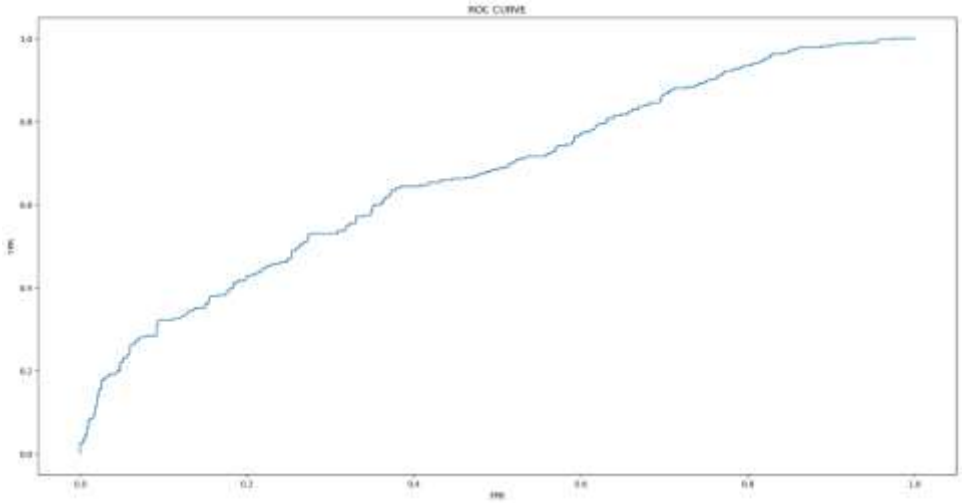
```
       0.2487219 , 0.25492298, 0.37399093, 0.87892182, 0.29095367,
       0.23158988, 0.46543108, 0.42044621, 0.51682828, 0.26735774,
       0.46852034, 0.71214235, 0.46929534, 0.3387666 , 0.47579165,
       0.53465943, 0.69327654, 0.71076491, 0.25244561, 0.49346395,
       0.59168609, 0.37407042, 0.35161748, 0.26982912, 0.30838741,
       0.36585556, 0.30473261, 0.17096602, 0.25315942, 0.6742579 ,
       0.67309905, 0.07630451, 0.55827631, 0.36173283, 0.59603499,
       0.58412988, 0.08790678, 0.27253148, 0.48227286, 0.47360165,
       0.78339334, 0.47918514, 0.9280928 , 0.36944034, 0.4494597 ,
       0.33586696, 0.14220134, 0.48000428, 0.24793365, 0.76854367,
       0.21695739, 0.44738269, 0.41290153, 0.5       , 0.26960966,
       0.51090436, 0.13386634, 0.34461216, 0.81735025, 0.26406992,
       0.29050614, 0.18320888, 0.19924598, 0.23580083, 0.36589384,
       0.4026348 , 0.31266641, 0.51838114, 0.48730182, 0.21350442,
       0.36117021, 0.54639962, 0.41500967, 0.86575244, 0.22454906,
       0.22150922, 0.39836156, 0.57844438, 0.28204959, 0.22208603,
       0.60170737, 0.33845475, 0.2509819 , 0.31020252, 0.50723911,
       0.12876439, 0.69931755, 0.16792373, 0.37384695, 0.18197902,
       0.5       , 0.85417904, 0.18272843, 0.2261301 , 0.29719705,
       0.20312248, 0.24671459, 0.21193307, 0.37920254, 0.60041395,
       0.30905406, 0.31815847, 0.305733  , 0.64449159, 0.24613036,
       0.60276269, 0.13233244, 0.21126279, 0.39968291, 0.29914173,
       0.25770172, 0.28330546, 0.23064426, 0.41114391, 0.36992011,
       0.23190624, 0.30406775, 0.34534643, 0.25997826, 0.67913705,
       0.34189648, 0.3706534 , 0.5       , 0.30334958, 0.25544724,
       0.41559442, 0.21526505, 0.23080645, 0.23702502, 0.23521414,
       0.86353947, 0.49016604, 0.62672234, 0.32206588, 0.20086889,
       0.43177914, 0.51481339, 0.83013705, 0.3059974 , 0.21933212,
       0.29476169, 0.29373247, 0.27033468, 0.29469538, 0.46104439,
       0.43838189, 0.26411645, 0.40359672, 0.18025714, 0.28337242,
       0.31329988, 0.61112963, 0.36402615, 0.14120663, 0.70885931,
       0.3625899 , 0.48751121, 0.67049586, 0.71949952, 0.64745446,
       0.66984353, 0.43751774, 0.38019862, 0.42410113, 0.2463082 ,
       0.25262596, 0.27178612, 0.70558982, 0.29861834, 0.60678983,
       0.21385787, 0.28927394, 0.55438434, 0.27362536, 0.17718679,
       0.31664902, 0.40109701, 0.57567166, 0.21699217])
```

In [50]:

```
fpr,tpr,threshsholds = roc_curve(y_test,probability)
```

In [51]:

```
plt.plot(fpr,tpr)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC CURVE')
plt.show()
```



In [ ]:

Save the Model:

```
import pickle
import joblib

model_filename = 'riverwater1_model.pkl'
with open(model_filename, 'wb') as file:
    pickle.dump(rand_search, file)

scaler_filename = 'scaler1.pkl'
joblib.dump(scaler, scaler_filename)
```

```
['scaler1.pkl']
```

7.2 Feature 2:

Flask Code:

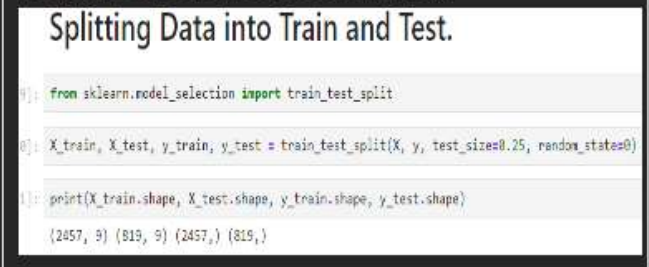
```
#!/usr/bin/env python
# coding: utf-8

# In[ ]:
# Import necessary libraries
```

```
from flask import Flask, request, render_template
import pickle
import numpy as np
import joblib

# Initialize Flask app
app = Flask(__name__)

# Load the trained model
model_filename = 'riverwater1_model.pkl'
with open(model_filename, 'rb') as file:
    model = pickle.load(file)

# Load the scaler
scaler_filename = 'scaler1.pkl'
scaler = joblib.load(scaler_filename)

# Define the home route
@app.route('/')
def home():
    return render_template('index1.html')

# Define the prediction route
@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Get user input from the HTML form
        features = [float(request.form[name]) for name in ['ph', 'Hardness',
'Solids', 'Chloroamines', 'Sulfate', 'Conductivity', 'Organic_Carbon',
'Trihalomethanes', 'Turbidity']]


       # Transform the features using the scaler
```

```
        scaled_features = scaler.transform([features])

        # Make predictions using the loaded model
        prediction = model.predict(scaled_features)

        # Determine the output based on the prediction
        if prediction[0] == 1:
            output = "Water is Good you can Drink"
        elif prediction[0] == 0:
            output = "Dont Drink The Water"
        else:
            output = "Not sure"

        # Pass the prediction result to the HTML template
        return render_template('index1.html', prediction_text=output)

    except Exception as e:
        # Handle errors gracefully and provide an appropriate message
        return render_template('index1.html', prediction_text='An error occurred. Please check your input.')

# Run the Flask app
if __name__ == '__main__':
    app.run(debug=True)
```

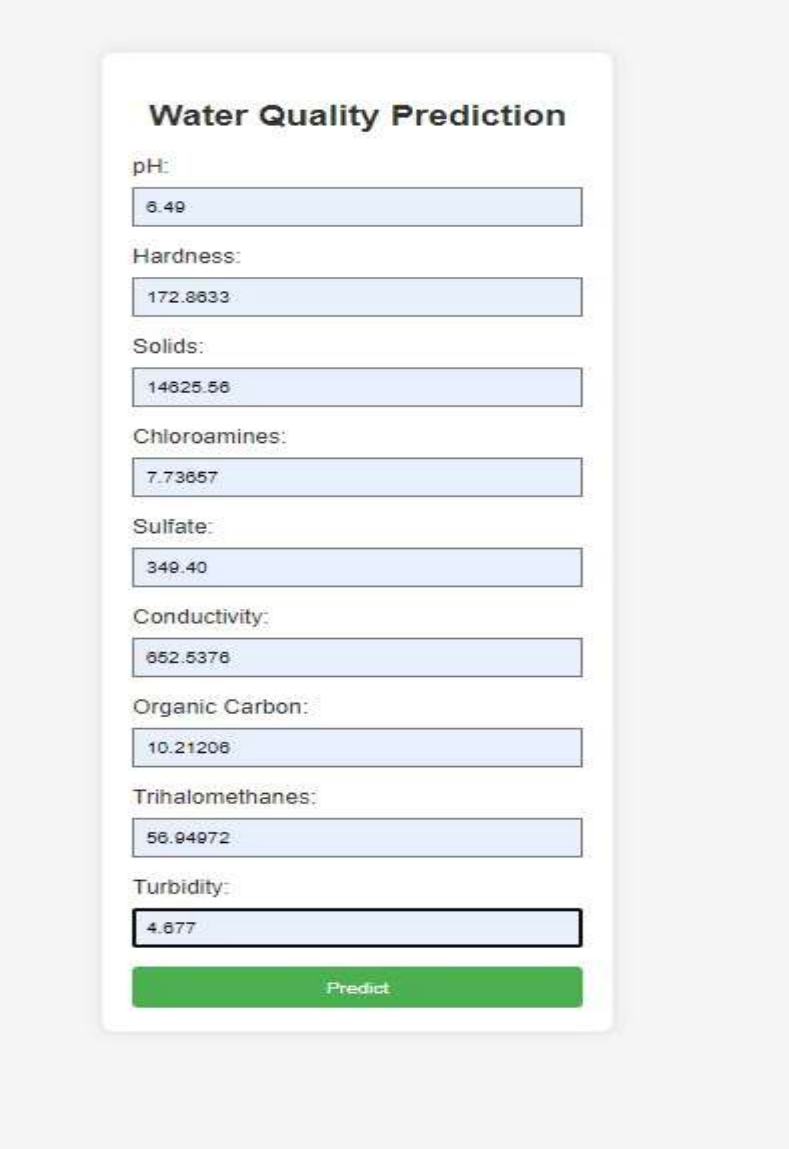
8. PERFORMANCE TESTING:

8.1 Performance Metrics:

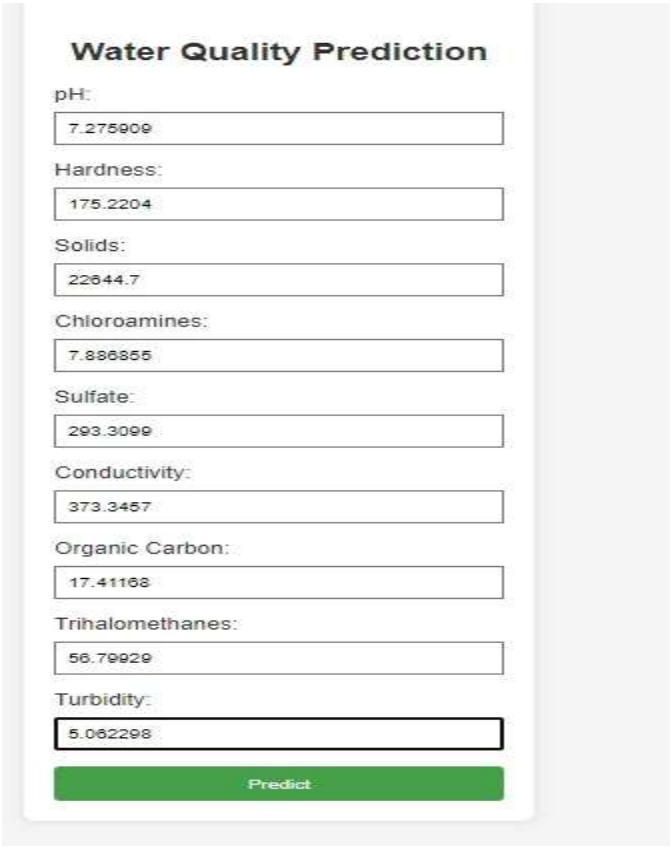
S.No.	Parameter	Values	Screenshot
1.	Metrics	**Regression Model:** NaN **Classification Model:** accuracy_score, confusion_matrix, classification_report, roc_auc_score, roc_curve	**Classification Model Screenshot:** 
2.	Tune the Model	**Hyperparameter Tuning:** RandomizedSearchCV **Validation Method:** train_test_split	**Hyperparameter Tuning Screenshot:**  **Validation Method Screenshot:** 

9. RESULTS:

9.1 Output Screenshots:




Prediction Result
Don't Drink The Water


Water Quality Prediction
pH:
7.275909
Hardness:
175.2204
Solids:
22644.7
Chloroamines:
7.886855
Sulfate:
293.3099
Conductivity:
373.3457
Organic Carbon:
17.41168
Trihalomethanes:
56.79929
Turbidity:
5.062298
Predict

10. ADVANTAGES & DISADVANTAGES:

Advantages:

Data-Driven: By analyzing historical data and trends, machine learning algorithms can identify patterns and establish correlations.

Time-Efficient: Machine learning algorithms can process large amounts of data and provide real-time predictions, saving valuable time and resources.

Reliable: These algorithms utilize complex mathematical models to predict outcomes with a high degree of accuracy.

Adaptive: Machine learning algorithms can learn from their mistakes and improve their accuracy over time.

Cost-Effective: Machine learning solutions require less manual intervention, resulting in lower operational costs.

Customizable: These algorithms can be tailored to specific requirements, enhancing their efficiency and effectiveness.

Integrative: Machine learning can be used to integrate and analyze data from multiple sources, providing a comprehensive understanding of the water quality situation.

Enhanced Decision-Making: Accurate water quality forecasts enable more informed decision-making by stakeholders, including policymakers, managers, and community members.

Disadvantages:

Limited Accuracy: Despite advancements in machine learning, it is important to recognize that the accuracy of predictions is not guaranteed. There is always a risk of errors or inaccuracies.

Complexity: Machine learning algorithms require complex data analysis and advanced mathematical skills to develop and interpret. This can be challenging for those without specialized expertise.

Dependence on Data Quality: The accuracy of machine learning predictions is directly dependent on the quality and quantity of data used for training. If the data is insufficient, inconsistent, or inaccurate, the algorithm's performance will be compromised.

Potential Bias: Machine learning algorithms may inadvertently perpetuate or amplify existing biases in the data. This can lead to skewed predictions and inaccurate conclusions.

Lack of Interpretability: Machine learning algorithms, especially those using advanced mathematical models, can be difficult to interpret and explain. This can make it challenging to gain support for or confidence in the algorithm's predictions.

11. CONCLUSION:

In conclusion, While the proposed model shows promising results, there is still room for improvement. More extensive feature engineering and data augmentation could be performed to enhance the model's predictive capabilities. Additionally, more complex model architectures such as CNNs and Transformers can be explored for potential superior performance.

12. FUTURE SCOPE:

To further advance river water quality forecasting, future research could explore the integration of multi-source data and incorporation of domain knowledge in the model architecture. This would help improve the accuracy and robustness of the predictions.

13. APPENDIX:

Source Code:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [ ]:

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

In [ ]:

```
df
```

In [ ]:

```
print(df.shape)
```

In [ ]:

```
print(df.columns)
```

In [ ]:

```
df.describe()
```

In [ ]:

```
df.info()
```

Handling Null Values:

In [ ]:

```
print(df.nunique())
```

In [ ]:

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

In [ ]:

```
df.dtypes
```

Data Visualization:

In [ ]:

```
sns.heatmap(df.isnull())
```

In [ ]:

```
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot= True, cmap='coolwarm')
```

In [ ]:

```
corr = df.corr()
c1 = corr.abs().unstack()
c1.sort_values(ascending = False)[12:24:2]
```

In [ ]:

```
ax = sns.countplot(x = "Potability",data= df, saturation=0.8)
plt.xticks(ticks=[0, 1], labels = ["Not Potable", "Potable"])
plt.show()
```

In [ ]:

```
plt.rcParams['figure.figsize'] = [20,10]
df.hist()
plt.show()
```

In [ ]:

```
sns.pairplot(df, hue="Potability")
```

In [ ]:

```
df.hist(column='ph', by='Potability')
```

In [ ]:

```
def Box(df):
    plt.title("Box Plot")
    sns.boxplot(df)
    plt.show()
Box(df['ph'])
```

imbalance dataset to balanace dataset:

In [ ]:

```
df.nunique()
```

In [ ]:

```
skew_val = df.skew().sort_values(ascending=False)
skew_val
```

In [ ]:

```
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 [ ]:

```
df.head()
```

In [ ]:

```
sns.heatmap(df.isnull())
```

In [ ]:

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

Splitting Dependent and Independent variables:

In [ ]:

```
X = df.drop('Potability', axis=1)
y = df['Potability']
```

In [ ]:

```
X.shape, y.shape
```

In [ ]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [ ]:

```
X = scaler.fit_transform(X)
X
```

Splitting Data into Train and Test:

In [ ]:

```
from sklearn.model_selection import train_test_split
```

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=0)
```

In [ ]:

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

Model Building o Import the model building Libraries o Initializing the model o Training and testing the model o Evaluation of Model o Save the Model

Training and testing the model:

In [ ]:

```
from sklearn.svm import SVC
model=SVC(probability=True)
```

In [ ]:

```
rand_list={"C":[2,3,5,6,7,8,10],
           "gamma":[0.1,0.2,0.5,0.4,0.8]}
```

In [ ]:

```
from sklearn.model_selection import RandomizedSearchCV
rand_search=RandomizedSearchCV(model,param_distributions=rand_list,n_iter=20,cv=5
)
```

In [ ]:

```
rand_search.fit(X_train,y_train)
```

In [ ]:

```
pred=rand_search.predict(X_test)
```

In [ ]:

```
pred
```

In [ ]:

```
y_test
```

In [ ]:

```
df
```

Evaluation of Model:

In [ ]:

```
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report,roc_auc_score,roc_curve
```

In [ ]:

```
accuracy_score(y_test,pred)
```

In [ ]:

```
confusion_matrix(y_test,pred)
```

In [ ]:

```
pd.crosstab(y_test,pred)
```

In [ ]:

```
print(classification_report(y_test,pred))
```

In [ ]:

```
probability=rand_search.predict_proba(X_test)[:,1]
```

In [ ]:

```
probability
```

In [ ]:

```
fpr,tpr,threshsholds = roc_curve(y_test,probability)
```

In [ ]:

```
plt.plot(fpr,tpr)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC CURVE')
plt.show()
```

Save the Model:

```
import pickle

import joblib


model_filename = 'riverwater1_model.pkl'

with open(model_filename, 'wb') as file:

    pickle.dump(rand_search, file)


scaler_filename = 'scaler1.pkl'

joblib.dump(scaler, scaler_filename)


['scaler1.pkl']
```

Flask Code:

```
#!/usr/bin/env python

# coding: utf-8


# In[ ]:
```

```
# Import necessary libraries
from flask import Flask, request, render_template
import pickle
import numpy as np
import joblib

# Initialize Flask app
app = Flask(__name__)

# Load the trained model
model_filename = 'riverwater1_model.pkl'
with open(model_filename, 'rb') as file:
    model = pickle.load(file)

# Load the scaler
scaler_filename = 'scaler1.pkl'
scaler = joblib.load(scaler_filename)

# Define the home route
@app.route('/')
def home():
    return render_template('index1.html')

# Define the prediction route
@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Get user input from the HTML form
        features = [float(request.form[name]) for name in ['ph', 'Hardness',
'Solids', 'Chloroamines', 'Sulfate', 'Conductivity', 'Organic_Carbon',
'Trihalomethanes', 'Turbidity']]
```

```
        # Transform the features using the scaler
         scaled_features = scaler.transform([features])

        # Make predictions using the loaded model
         prediction = model.predict(scaled_features)

        # Determine the output based on the prediction
         if prediction[0] == 1:
             output = "Water is Good you can Drink"
         elif prediction[0] == 0:
             output = "Dont Drink The Water"
         else:
             output = "Not sure"


         # Pass the prediction result to the HTML template
         return render_template('index1.html', prediction_text=output)


    except Exception as e:
        # Handle errors gracefully and provide an appropriate message
        return render_template('index1.html', prediction_text='An error
occurred. Please check your input.')
# Run the Flask app
if __name__ == '__main__':
    app.run(debug=True)
```

GitHub & Project Demo Link:

https://github.com/smartinternz02/SI-GuidedProject-612942-1699454793