**PROJECT: WATER QUALITY ANALYSIS**

**PHASE-3 DEVELOPMENT PART 1**

**TEAM MEMBER NAME:** Krithiga M

**OBJECTIVE:**

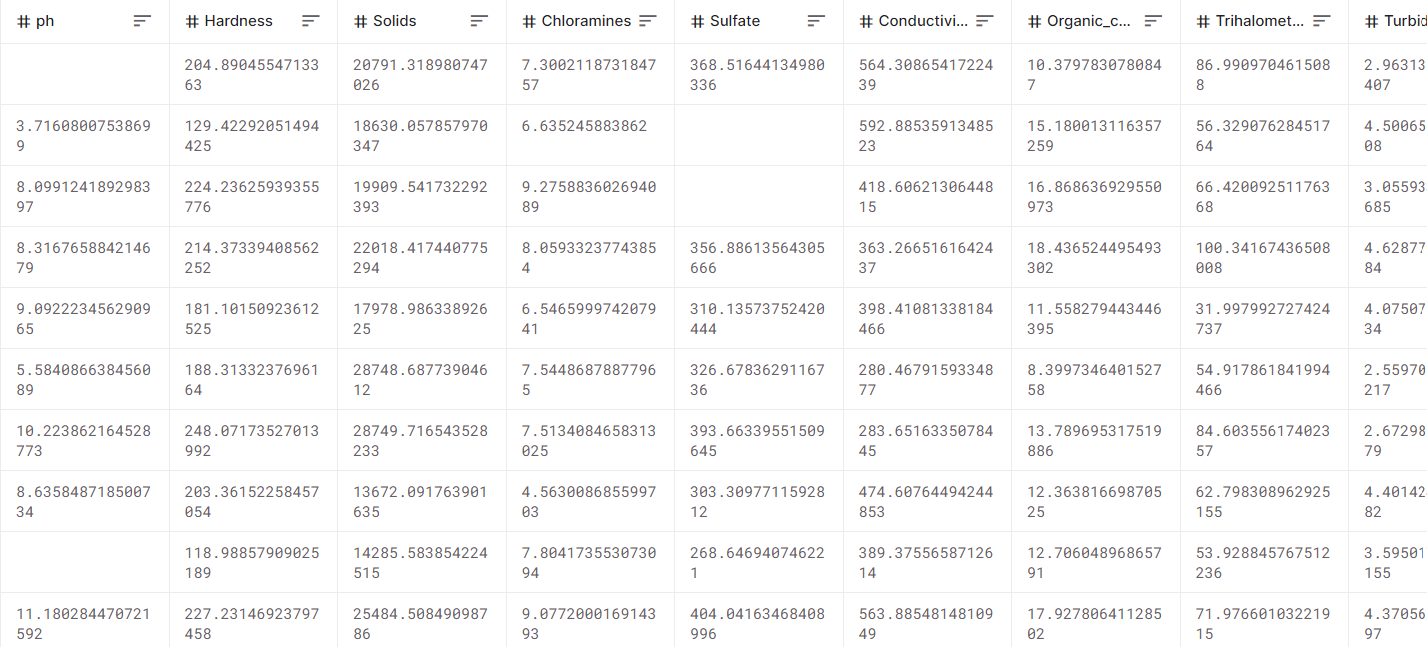
The main objective of this document is to build this project water quality analysis by loading, preprocessing the data and performing exploratory data analysis. To obtain the water quality dataset and preprocess it by handling missing values and outliers,

Conduct EDA to visualize parameter distributions, correlations, and potential deviations from standards.

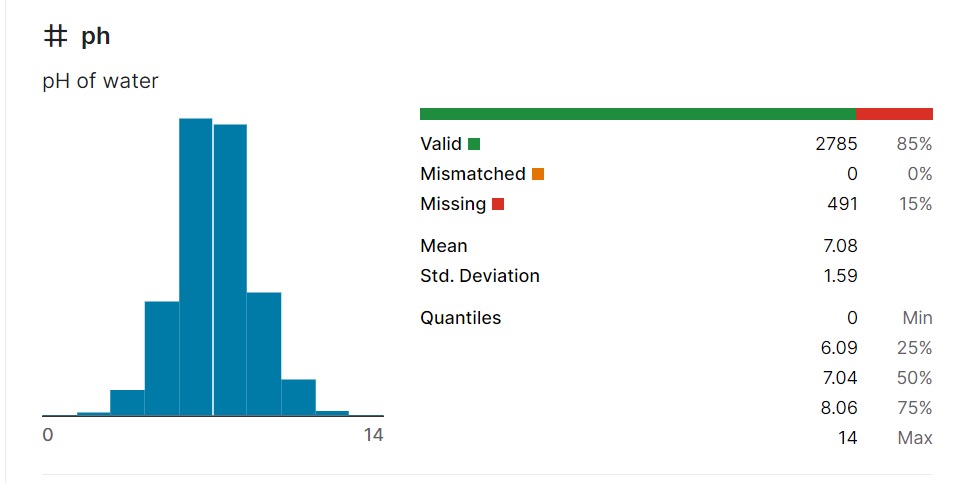
**DATA LOADING AND PREPROCESSING:**

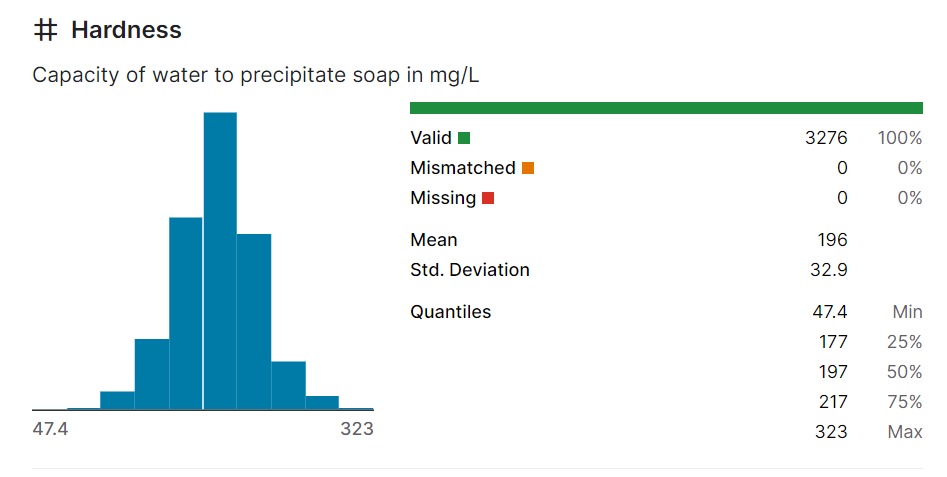
* Data preprocessing is crucial in water quality analysis to ensure that the data is clean, consistent, and ready for analysis. Water quality data can come from various sources, including sensors, laboratory measurements, and field observations.
* Data loading in water quality analysis refers to the process of acquiring and importing water quality data from various sources into a software or data analysis environment for further processing, analysis, and interpretation. Water quality analysis involves the assessment of different parameters, such as temperature, pH, dissolved oxygen, turbidity, chemical concentrations, and more to evaluate the quality of water in a particular environment, such as a river, lake, or reservoir. Data loading is a crucial initial step in this analysis process

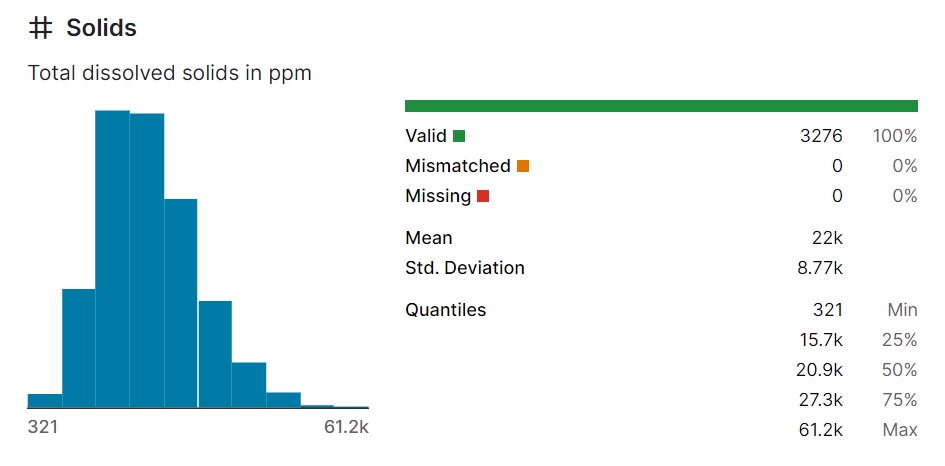
**GIVEN DATASET:**

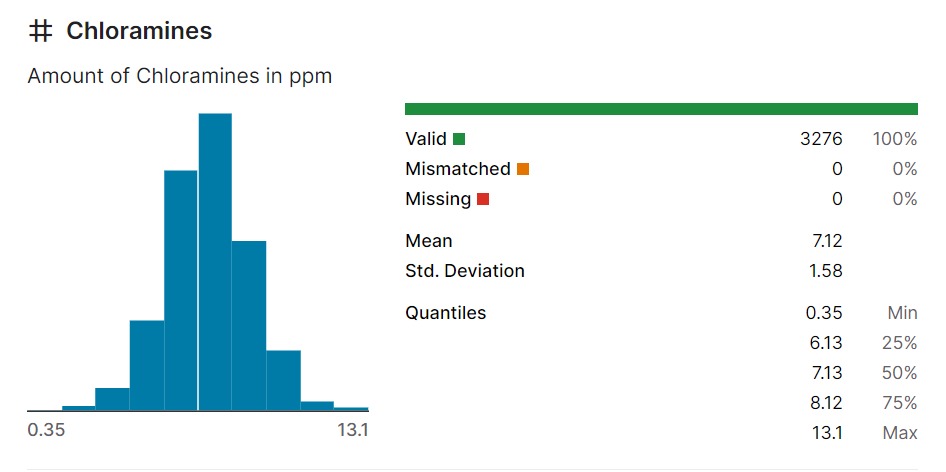
****

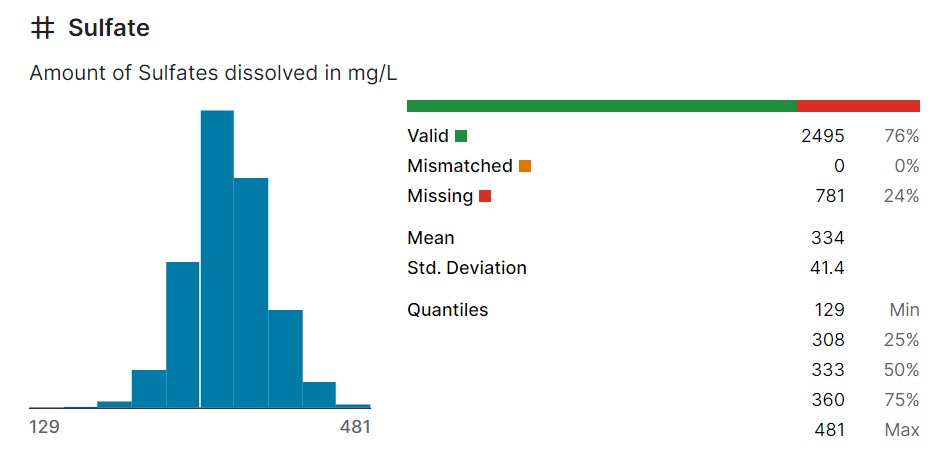
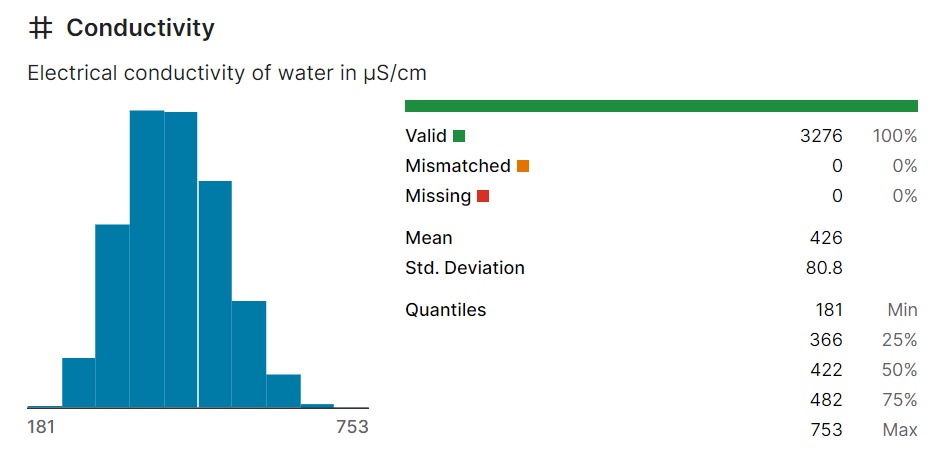
**WE WILL PREPROCESS THE FOLLOWING ATTRIBUTES:**

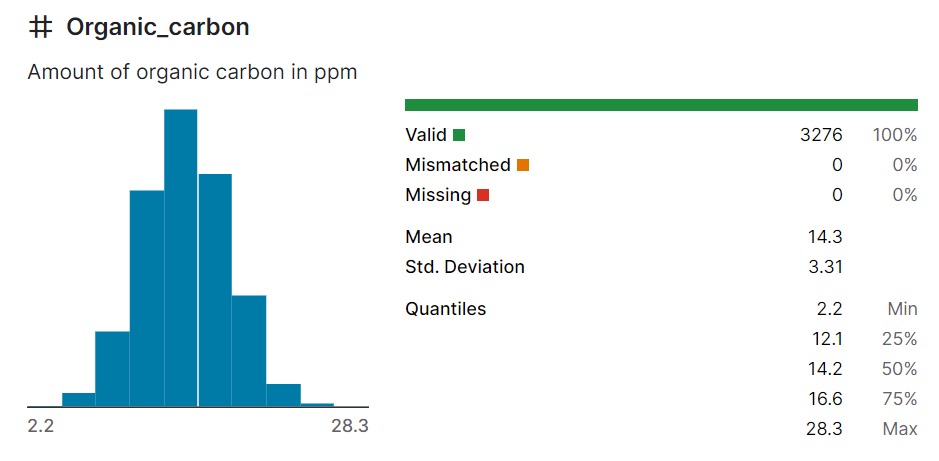


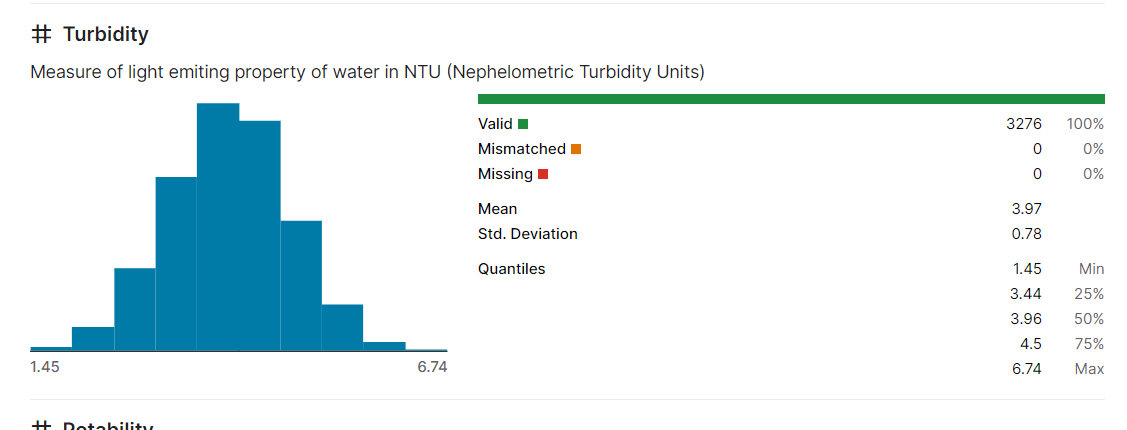


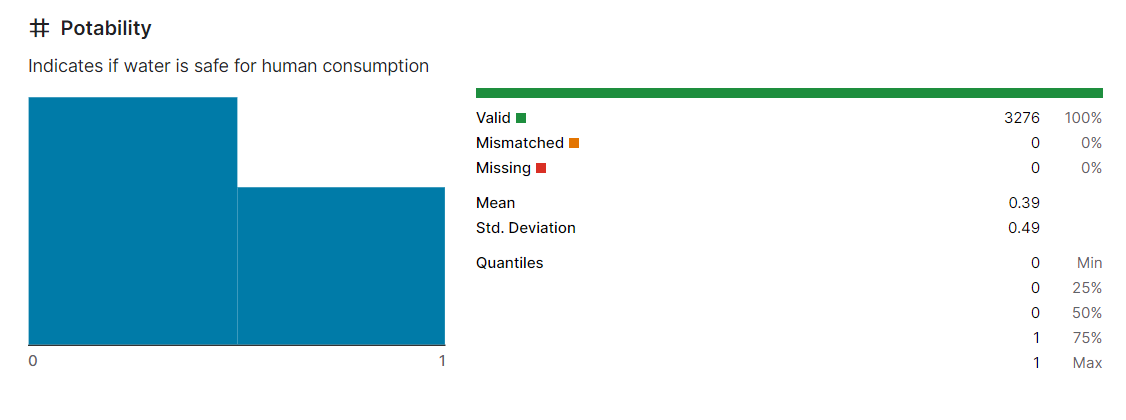






****

****

**Building a water quality analysis project involves loading and preprocessing a dataset to extract meaningful insights or predictions. Here's a step-by-step guide using and some common libraries:**

1. **Import necessary libraries:**

First, import the libraries you'll need for this project:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.preprocessing import StandardScaler

**2. Load the dataset:**

You should have a water quality analysis dataset in a format such as CSV. Replace `'water\_quality\_data.csv'` with your dataset file.

data = pd.read\_csv('water\_quality\_data.csv')

**3. Explore the dataset:**

Before preprocessing, it's essential to understand your data. Explore the first few rows, check for missing values, and look at summary statistics.

# Display the first few rows of the dataset

print(data.head())

# Check for missing values

print(data.isnull().sum())

# Get summary statistics

print(data.describe())

**4. Data Preprocessing:**

Data preprocessing is crucial in water quality analysis to ensure that the data is clean, consistent, and ready for analysis. Water quality data can come from various sources, including sensors, laboratory measurements, and field observations.

Clean and preprocess the data to make it suitable for analysis. Common preprocessing steps include handling missing values and encoding categorical variables

**Handle missing values:**

data = data.dropna()

# Encode categorical variables (if any)

# For example, you can use one-hot encoding for categorical variables.

data = pd.get\_dummies(data, columns=['location'])

**5. Split the data:**

Split the dataset into training and testing sets. This is crucial for model evaluation.

X = data.drop('target\_variable', axis=1) # Replace 'target\_variable' with the actual target variable name

y = data['target\_variable']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**6. Feature Scaling:**

Scale the features, especially if you plan to use machine learning models that are sensitive to feature scales.

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

**7. Data Visualization:**

Visualize your data to gain insights and understand its characteristics. This can include histograms, scatter plots, or correlation matrices.

# Example: Create a histogram of a specific feature

plt.hist(X\_train['feature\_name'], bins=20)

plt.xlabel('Feature Name')

plt.ylabel('Frequency')

plt.title('Histogram of Feature Name')

plt.show()

**8. Build your Water Quality Analysis Model:**

Depending on your project's goals, you can now build a model. It could be a regression model, classification model, or any other analytical technique suitable for water quality analysis. Implement and train your chosen model using the preprocessed data.

Remember to adapt the above steps to your specific dataset, target variable, and analysis objectives. Water quality analysis can range from basic descriptive statistics to complex predictive modeling, so your approach will vary accordingly.

**PROGRAM:**

**INPUT:**

**data=pd.read\_csv("water\_potability.csv")**

**Data.head()**

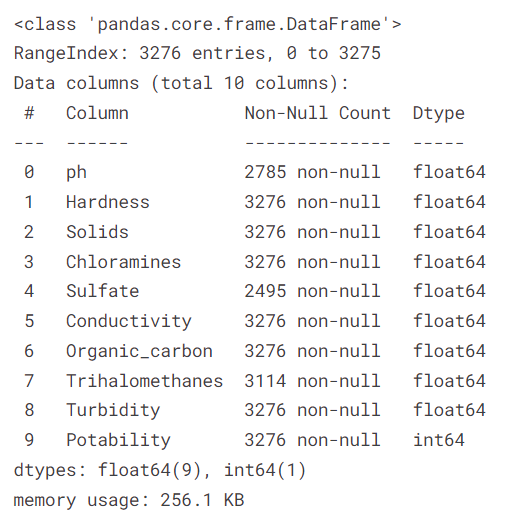
**OUTPUT:**



**INPUT:**

**data.info()**

**OUTPUT**

****

DATA PRE-PROCESSING

**Finding Missing Values**

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.shape()**

Out[7]:

(3276, 10)

**DATA NORMALIZATION AND STANDARDIZATION**

In [17]:

from sklearn.preprocessing import MinMaxScaler,StandardScaler

normalizer=MinMaxScaler()

standardizer=StandardScaler()

X= normalizer.fit\_transform(X)

X=standardizer.fit\_transform(X)

In [18]:

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

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.2,random\_state=62)

**Replacing Missing Values with any of the central Tendencies. I am choosing Mean**

In [8]:

data=data.dropna()

In [9]:

data.head()

Out[9]:

|  | ph | Hardness | Solids | Chloramines | Sulfate | Conductivity | Organic\_carbon | Trihalomethanes | Turbidity | Potability |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |
| 5 | 5.584087 | 188.313324 | 28748.687739 | 7.544869 | 326.678363 | 280.467916 | 8.399735 | 54.917862 | 2.559708 | 0 |
| 6 | 10.223862 | 248.071735 | 28749.716544 | 7.513408 | 393.663396 | 283.651634 | 13.789695 | 84.603556 | 2.672989 | 0 |
| 7 | 8.635849 | 203.361523 | 13672.091764 | 4.563009 | 303.309771 | 474.607645 | 12.363817 | 62.798309 | 4.401425 | 0 |

**FINDING OUTLIERS:**

**Choosing IQR Range to find the Outliers**

**def Outliers(column):**

q1=column.quantile(0.25)

q3=column.quantile(0.75)

IQR=q3-q1

lower=q1-1.5\*(IQR)

upper=q3+1.5\*(IQR)

return column[(column<lower) | (column>upper)]

outliers\_dict = {}

for column in data.select\_dtypes(include=['number']):

outliers = Outliers(data[column])

if not outliers.empty:

outliers\_dict[column] = outliers

# Print potential outliers for each column

for column, outliers in outliers\_dict.items():

print(f"Potential outliers in column '{column}':")

print(outliers)

Potential outliers in column 'ph':

9 11.180284

317 11.301794

692 1.757037

726 0.227499

783 11.898078

810 0.989912

1023 11.027880

1162 11.244507

1231 2.690831

1303 12.246928

1343 2.569244

1353 11.534880

2075 14.000000

2096 11.568768

2165 2.803563

2189 2.558103

2263 11.235426

2300 2.974429

2343 2.538116

2681 2.376768

2895 13.349889

2899 1.431782

2925 11.563169

2932 2.925174

2945 11.496702

2993 3.102076

3017 11.496859

3088 2.128531

3094 1.985383

3108 11.449739

3269 11.491011

Name: ph, dtype: float64

Potential outliers in column 'Hardness':

51 100.457615

71 116.299330

88 300.292476

180 278.056321

189 112.299485

218 276.733569

227 112.820254

258 98.771644

260 280.082411

262 278.585105

275 280.089655

278 81.710895

309 113.831112

317 77.459586

335 94.091307

342 282.739017

346 278.147524

379 73.492234

1490 306.627481

1536 106.380113

1542 97.280909

1564 283.997284

1649 98.452931

1743 107.383327

1777 277.116946

1829 113.504698

1859 117.057314

1868 110.865788

1989 278.340358

2012 281.594235

2212 278.036360

2343 100.806520

2383 107.341982

2508 278.081446

2512 283.895864

2606 116.338278

2630 276.699765

2632 286.567991

2669 278.231754

2715 111.478582

2775 114.463900

2777 111.994028

2861 317.338124

2877 114.371450

3179 94.908977

3218 287.975540

3230 114.807578

3244 277.065713

3269 94.812545

Name: Hardness, dtype: float64

Potential outliers in column 'Solids':

142 46140.126850

186 45222.506665

283 48621.563952

378 45249.449033

516 45510.584319

546 49074.730407

583 44652.363872

648 44612.751358

987 48002.084596

1068 55334.702799

1106 44586.812651

1186 56351.396304

1332 45166.912141

1343 48204.172192

1462 45939.689158

1527 46718.555965

1554 56488.672413

1784 50279.262429

1955 49009.924656

1984 47022.745845

2012 47852.888871

2680 48175.852093

2891 45050.002276

2993 45148.808118

3130 50793.898917

3162 53735.899194

3173 44539.738323

3271 47580.991603

Name: Solids, dtype: float64

Potential outliers in column 'Chloramines':

272 12.580026

275 13.043806

322 11.078872

324 11.170789

351 13.127000

408 2.484380

434 12.062536

437 2.981379

454 2.993744

534 11.543190

738 11.523598

772 2.866073

806 2.862535

814 11.302831

1057 11.086526

1099 3.181183

1106 2.741712

1537 11.129154

1698 2.456014

1776 3.139553

1868 2.621268

2110 11.264386

2206 2.458609

2212 2.785718

2300 2.498597

2326 3.117441

2336 12.626900

2344 11.101628

2346 2.855790

2350 1.390871

2352 2.397985

2370 11.994290

2395 11.448469

2401 1.920271

2424 12.246394

2446 12.227175

2447 11.930448

2535 2.726766

2562 11.299390

2566 2.654491

2694 12.653362

2714 2.648390

2796 3.016033

Name: Chloramines, dtype: float64

Potential outliers in column 'Sulfate':

253 187.170714

272 192.033592

275 180.206746

345 444.970552

351 182.397370

365 187.424131

385 209.471058

680 223.235816

703 224.212503

781 445.938391

782 229.575561

810 444.375731

1106 219.148935

1186 219.553437

1189 227.348460

1366 203.444521

1412 442.761428

1523 206.247229

1536 441.587654

1537 475.737460

1554 129.000000

1605 476.539717

1662 225.516628

1743 460.107069

1766 445.359547

1773 458.441072

1798 214.460834

1860 207.890482

1888 439.787938

2156 447.417962

2204 205.935091

2318 481.030642

2363 437.647163

2726 450.914454

2779 227.665635

2823 440.635509

2853 446.724016

Name: Sulfate, dtype: float64

Potential outliers in column 'Conductivity':

66 669.725086

342 695.369528

1183 656.924128

1295 666.690618

2134 708.226364

2704 753.342620

2737 657.570422

Name: Conductivity, dtype: float64

Potential outliers in column 'Organic\_carbon':

43 23.917601

698 23.569645

785 2.200000

876 4.966862

1390 4.371899

1447 4.861631

1536 5.218233

2057 24.755392

2082 5.188466

2224 4.466772

2236 27.006707

2414 5.196717

2680 5.159380

3169 23.604298

Name: Organic\_carbon, dtype: float64

Potential outliers in column 'Trihalomethanes':

61 17.915723

133 23.817020

350 112.622733

518 23.792950

531 120.030077

698 19.175175

1041 23.136611

1123 8.577013

1156 116.161622

1316 114.208671

1360 16.291505

1630 17.527765

1767 22.219327

1876 110.431080

1962 113.048886

2121 15.684877

2353 18.015272

2376 124.000000

2949 14.343161

3010 111.115310

3102 111.595448

3184 114.034946

Name: Trihalomethanes, dtype: float64

Potential outliers in column 'Turbidity':

382 6.494249

593 1.680554

789 1.812529

990 6.357439

1073 6.389161

1290 1.496101

1892 1.492207

2377 6.226580

2757 6.307678

2921 6.494749

3042 1.450000

Name: Turbidity, dtype: float64

Replacing Outliers with Mean Values of the Column

In [12]:

linkcode

def replaceOut(column):

q1=column.quantile(0.25)

q3=column.quantile(0.75)

IQR=q3-q1

lower=q1-1.5\*(IQR)

upper=q3+1.5\*(IQR)

outliers=column[(column<=lower) | (column>=upper)]

if **not** outliers.empty:

column[outliers.index] = column.mean()

return column

**INPUT:**

*#for column in data.columns:*

*#data[column]=replaceOut(data[column])* In [14]:

cor=data.corr()

import seaborn as sns

import matplotlib.pyplot as plt

sns.heatmap(cor,annot=True,cmap='coolwarm')

plt.show()

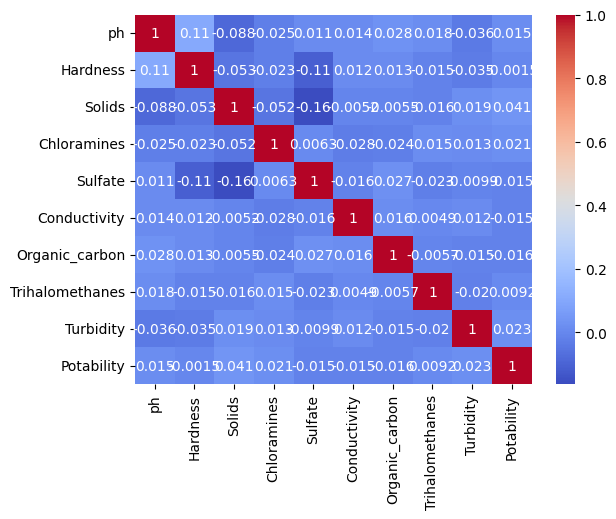
Test,Train,Split

In [16]:

X=data.drop('Potability',axis=1)

Y=data['Potability']

**OUTPUT:**



**CONCLUSION:**

In this initial phase of building a water quality analysis project, we have successfully accomplished several key tasks. We obtained a water quality dataset from a reliable source, preprocessed the data, and conducted exploratory data analysis (EDA) to gain insights into the dataset