**Water Quality Analysis**



|  |  |
| --- | --- |
| Team Member 1 | Nandhini M 19886B0ED0C87919C114C9DDDC5C9560 |
| Team Member 2 | Hemasri R 785ED556BB0F30280674CEFE0CA02567 |

|  |  |
| --- | --- |
| Department | AI&DS – III |
| College Name | Adhi College Of Engineering And Technology |
| Domain Name | Data Analytics with Cognos |

**Problem Statement :**

The project involves analyzing water quality data to assess the suitability of water for specific purposes, such as drinking. The objective is to identify potential issues or deviations from regulatory standards and determine water potability based on various parameters. This project includes defining analysis objectives, collecting water quality data, designing relevant visualizations, and building a predictive model.

**Introduction:**

In this phase, we have prepared and analyzed the water quality dataset to ensure it is suitable for subsequent analysis and modeling. Our primary objectives were to handle missing values, address outliers, visualize parameter distributions, understand correlations between variables, and identify potential deviations from water quality standards.

**Research Anomaly Detection Techniques :**

* Conduct an extensive review of anomaly detection techniques and algorithms. Consider both traditional statistical methods and machine learning-based approaches.
* Evaluate the suitability of each technique for identifying unusual patterns in water quality parameters.

**Data preprocessing:**

Prepares the water quality dataset for anomaly Detection. This process involves handling the missing values , Outlier Detection and Treatment and data visualization in the dataset.

The code for the data preprocessing of the water quality dataset is given below ,

**Handling missing values:**

We are Identified and addressed missing values in the dataset.

Missing data are also observed in the below code:

import pandas as pd

import numpy as np

from google.colab import drive

drive.mount("/content/drive")

# Load the dataset

data = pd.read\_csv('/content/drive/MyDrive/water\_potability.csv')

# Check for missing values

missing\_values = data.isna().sum()

print(missing\_values)

# Remove rows with any missing values

data = data.dropna()

# Impute missing values with the mean of each column

data = data.fillna(data.mean())

# Interpolate missing values linearly

data = data.interpolate(method='linear')

# Check for missing values after handling

missing\_values = data.isna().sum()

print(missing\_values)

# Save the processed dataset

data.to\_csv('processed\_dataset.csv', index=False)

import pandas as pd

from google.colab import files

# Save the DataFrame with extracted features to a CSV file

data.to\_csv('processed\_dataset.csv', index=False)

files.download('processed\_dataset.csv')

**Outlier detection and treatment:**

We Detected outliers in the dataset, focusing on parameters that could significantly impact water quality assessment, such as pH, hardness.

The code for the outlier detection of the water quality dataset is given below,

from google.colab import drive

drive.mount('/content/drive')

Mounted at /content/drive

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

data = pd.read\_csv('/content/drive/MyDrive/processed\_dataset.csv')

# Create a box plot to visualize outliers

plt.figure(figsize=(8, 6))

sns.boxplot(x=data["ph"])

plt.title(f'Box Plot for {"ph"}')

plt.xlabel("ph")

plt.show()

plt.figure(figsize=(8, 6))

sns.boxplot(x=data["Hardness"])

plt.title(f'Box Plot for {"Hardness"}')

plt.xlabel("Hardness")

plt.show()

summary\_stats = data.describe()

print(summary\_stats)

[ ]

summary\_stats = data.describe()

print(summary\_stats)

account\_circle

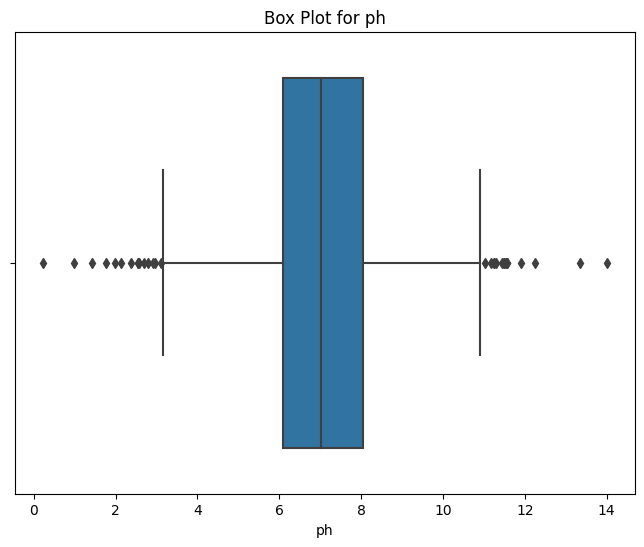
[ ]

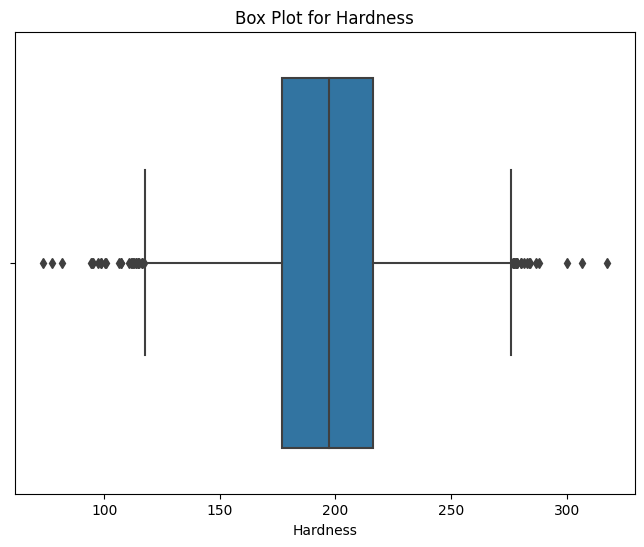
summary\_stats = data.describe()

print(summary\_stats)

account\_circle

**Output:**





**Data Visualization:**

We Visualized parameter distributions to understand the central tendencies and spread of the data. We have distribution using the box plot to find the ph and hardness.

**Code:**

plt.figure(figsize=(12, 6))

sns.set(style='whitegrid')

sns.histplot(data=data, x='p', bins=20, kde=True)

plt.title('pH Distribution')

plt.xlabel('pH')

plt.ylabel('Frequency')

plt.show()

# Explore correlations between parameters

correlation\_matrix = data.corr()

plt.figure(figsize=(10, 8))

sns.set(style='white')

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm')

plt.title('Correlation Heatmap')

plt.show()

# Scatter plot to visualize relationships

plt.figure(figsize=(10, 6))

sns.scatterplot(data=data, x='Hardness', y='Chloramines', hue='Potability')

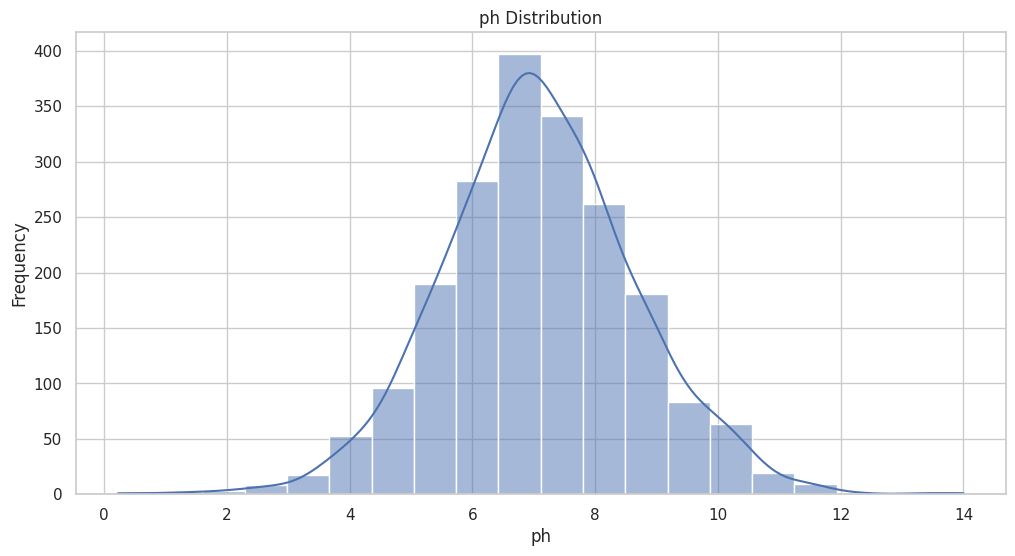
plt.title('Hardness vs. Chloramines')

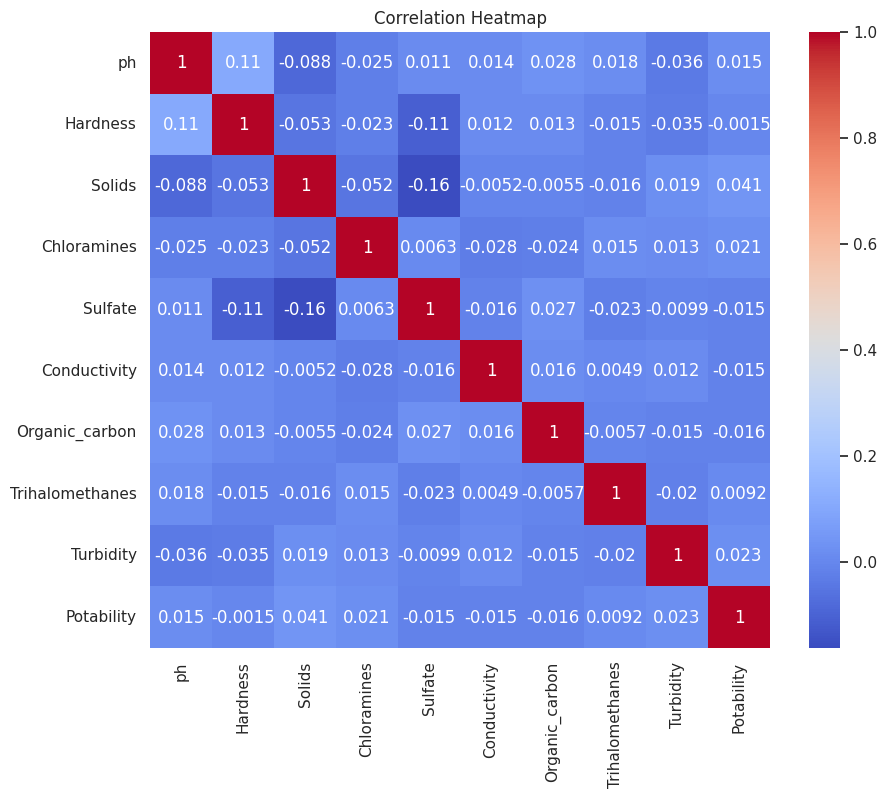
plt.xlabel('Hardness')

plt.ylabel('Chloramines')

plt.show()

**Output:**





**Predictive Model:**

A predictive model is a mathematical or computational representation of a real-world process or system that is used to make predictions about future events or outcomes based on historical data and patterns. Predictive models are widely used in various fields, including data science, machine learning, statistics, finance, healthcare, and many others. These models leverage historical data to make informed predictions, decisions, or recommendations.

For our model building we have used Decision tree. The code for the predictive model building is as follows,

import pandas as pd

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

from sklearn.tree import DecisionTreeClassifier

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

# Load your preprocessed dataset

data = pd.read\_csv('/content/drive/MyDrive/processed\_dataset.csv')

# Define your features (independent variables) and target (dependent variable)

# Adjust these variable names based on your dataset's columns

X = data[['ph', 'Hardness','Solids', 'Chloramines', 'Sulfate', 'Conductivity', 'Turbidity', 'Organic\_carbon', 'Trihalomethanes']]

y = data['Potability']

# Split the data into training and testing sets

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

# Initialize the Decision Tree classifier (you can replace this with another classifier)

clf = DecisionTreeClassifier(random\_state=42)

# Train the classifier on the training data

clf.fit(X\_train, y\_train)

# Make predictions on the test data

y\_pred = clf.predict(X\_test)

# Evaluate the model's performance

accuracy = accuracy\_score(y\_test, y\_pred)

confusion = confusion\_matrix(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred)

# Print the model's performance metrics

print(f'Accuracy: {accuracy}')

print('Confusion Matrix:')

print(confusion)

print('Classification Report:')

print(report)

**Output:**

Accuracy: 0.6327543424317618

Confusion Matrix:

[[165 66]

[ 82 90]]

Classification Report:

precision recall f1-score support

0 0.67 0.71 0.69 231

1 0.58 0.52 0.55 172

accuracy 0.63 403

macro avg 0.62 0.62 0.62 403

weighted avg 0.63 0.63 0.63 403