Water Quality Analysis

INTRODUCTION:
Water quality analysis involves assessing various physical, chemical, and biological characteristics of water to determine its suitability for different purposes such as drinking, industrial use, or ecological health. In a data analysis context, the process can be broken down into several steps. Keep in mind that the specifics can vary based on the data available and the goals of the analysis, but here is a general framework

Implementation Steps

Step-1: Define Objectives:
Clearly define the objectives of the water quality analysis. Are you assessing if water is suitable for drinking, for aquatic life, or for industrial use? The objectives will guide the selection of parameters to analyze.

Step-2: Data Collection:
Collect relevant data on water quality. This may include data from sensors, laboratory tests, historical records, or remote sensing technologies.
Ensure the data is representative of the water sources you are analyzing.

Step-3: Data Preprocessing:
Clean the data to address missing values, outliers, and inconsistencies.
Convert and standardize units if necessary.
Ensure that timestamps and other relevant metadata are in a usable format.

Step-4: Exploratory Data Analysis (EDA):
Conduct exploratory data analysis to understand the distribution of each parameter, identify patterns, and explore potential correlations.
Use visualizations such as scatter plots, histograms, and box plots to gain insights.

Step-5: Feature Engineering:
Create new features if needed, such as aggregating data over specific time periods or calculating derived parameters.
Consider transformations to make the data more suitable for analysis.

Step-6: Parameter Selection:
Based on the objectives, select the water quality parameters to analyze. This could include physical parameters (temperature, pH), chemical parameters (dissolved oxygen, nutrients), and biological parameters (bacterial counts, algae presence).

Step-7: Statistical Analysis:
Apply statistical methods to quantify relationships between different parameters.
Conduct hypothesis testing to determine if water quality meets specific standards or if there are significant differences between different samples.

Step-8: Machine Learning (if applicable):
If you have a large dataset and want to predict water quality based on certain features, consider employing machine learning models.
Split the data into training and testing sets, train the model, and evaluate its performance.

Step-9: Interpretation of Results:
Interpret the results in the context of the defined objectives.
Assess if water quality meets regulatory standards or if there are trends that require attention.

Step-10: Visualization and Reporting:
Create visualizations and reports summarizing the findings.
Clearly communicate the results, including any recommendations or actions to be taken.

Step-11: Continuous Monitoring and Improvement:
Establish a system for continuous monitoring of water quality.
Update the analysis as new data becomes available and refine models or methods based on ongoing results.

Step-12: Documentation:
Document all steps, methods, and findings for transparency and future reference.

Step-13: Communication:
Communicate the results to relevant stakeholders, whether they are policymakers, water treatment facilities, or the general public.

Development part Objective:
The objective of water quality analysis using IBM Cognos is to leverage advanced data analytics and reporting capabilities to assess and monitor the chemical, physical, and biological parameters of water sources. IBM Cognos enables organizations to collect, process, and visualize water quality data to ensure compliance with environmental regulations, identify contamination sources, and make informed decisions for water resource management. This analysis helps in safeguarding public health, preserving ecosystems, and optimizing water treatment processes, ultimately promoting sustainable and safe water supplies for communities and industries.

Data loading:
Load water quality data into analysis tools for evaluating quality of the water.

Data Preprocessing:
Data preprocessing for water quality analysis using IBM Cognos is a crucial process that involves several key steps to prepare and clean the collected data for meaningful analysis and reporting. The first step is data collection, where information from various sources like water quality monitoring stations, sensors, and laboratory tests is gathered. Next, data integration is necessary to combine data from different sources into a unified dataset, ensuring

standardized formats and units for consistency. Data cleaning is essential to address missing values, outliers, and inconsistencies, which improves data quality. Following this, data transformation may be required to make the data more suitable for analysis, such as scaling or normalizing variables. Feature engineering involves creating new derived variables that can offer additional insights, like calculating the Water Quality Index (WQI) or aggregating data over specific time intervals. Data validation ensures compliance with quality standards and regulations, and the preprocessed data is stored in a suitable format for IBM Cognos.

Given data set:

	A	B	C	D	E	F	G	H	I	J
1	ph	Hardness	Solids	Chloramin	Sulfate	Conductivi	Organic_ca	Trihalome	Turbidity	Potability
2		204.8905	20791.32	7.300212	368.5164	564.3087	10.37978	86.99097	2.963135	0
3	3.71608	129.4229	18630.06	6.635246		592.8854	15.18001	56.32908	4.500656	0
4	8.099124	224.2363	19909.54	9.275884		418.6062	16.86864	66.42009	3.055934	0
5	8.316766	214.3734	22018.42	8.059332	356.8861	363.2665	18.43652	100.3417	4.628771	0
6	9.092223	181.1015	17978.99	6.5466	310.1357	398.4108	11.55828	31.99799	4.075075	0
7	5.584087	188.3133	28748.69	7.544869	326.6784	280.4679	8.399735	54.91786	2.559708	0
8	10.22386	248.0717	28749.72	7.513408	393.6634	283.6516	13.7897	84.60356	2.672989	0
9	8.635849	203.3615	13672.09	4.563009	303.3098	474.6076	12.36382	62.79831	4.401425	0
10		118.9886	14285.58	7.804174	268.6469	389.3756	12.70605	53.92885	3.595017	0
11	11.18028	227.2315	25484.51	9.0772	404.0416	563.8855	17.92781	71.9766	4.370562	0
12	7.36064	165.5208	32452.61	7.550701	326.6244	425.3834	15.58681	78.74002	3.662292	0
13	7.974522	218.6933	18767.66	8.110385		364.0982	14.52575	76.48591	4.011718	0
14	7.119824	156.705	18730.81	3.606036	282.3441	347.715	15.92954	79.50078	3.445756	0
15		150.1749	27331.36	6.838223	299.4158	379.7618	19.37081	76.51	4.413974	0
16	7.496232	205.345	28388	5.072558		444.6454	13.22831	70.30021	4.777382	0
17	6.347272	186.7329	41065.23	9.629596	364.4877	516.7433	11.53978	75.07162	4.376348	0
18	7.051786	211.0494	30980.6	10.0948		315.1413	20.39702	56.6516	4.268429	0
19	9.18156	273.8138	24041.33	6.90499	398.3505	477.9746	13.38734	71.45736	4.503661	0
20	8.975464	279.3572	19460.4	6.204321		431.444	12.88876	63.82124	2.436086	0
21	7.37105	214.4966	25630.32	4.432669	335.7544	469.9146	12.50916	62.79728	2.560299	0
22		227.435	22305.57	10.33392		554.8201	16.33169	45.38282	4.133423	0
23	6.660212	168.2837	30944.36	5.858769	310.9309	523.6713	17.88424	77.04232	3.749701	0
24		215.9779	17107.22	5.60706	326.944	436.2562	14.18906	59.85548	5.459251	0
25	3.902476	196.9032	21167.5	6.996312		444.4789	16.60903	90.18168	4.528523	0
26	5.400302	140.7391	17266.59	10.05685	328.3582	472.8741	11.25638	56.93191	4.824786	0

	A	B	C	D	E	F	G	H	I	J
3250	6.260111	211.5941	18577.62	7.154891	340.7926	357.0984	7.99221	82.36538	5.403615	1
3251	10.80816	198.5968	29614.35	5.782418	304.6221	383.2694	14.90282	47.89641	4.362542	1
3252	7.371914	148.1937	42059.38	7.96671	324.5463	544.8484	17.1665	62.67776	4.338957	1
3253	4.825591	234.7839	11142.39	6.442769	370.4168	370.1889	13.04635	46.31599	3.463097	1
3254	4.868827	258.679	13400.39	4.88091		328.7645	17.35208	55.96822	3.2556	1
3255	7.395451	190.4779	22561.51	8.310195	294.0304	413.9103	13.30137	63.41018	4.990236	1
3256	8.862113	131.6352	17433.6	7.639573	340.1332	399.4628	16.71221	53.5941	4.955082	1
3257	6.008974	225.0802	5100.094	7.452236	336.119	325.1345	11.07995	36.34101	4.01234	1
3258	7.607224	160.5653	39184.85	7.826411	312.0561	503.1581	13.36699	62.02231	3.525027	1
3259	6.683368	272.1117	18989.32	5.336202	336.5551	307.725	20.17872	75.40226	5.208061	1
3260	6.638411	180.8267	9772.505	8.295983		401.1111	12.60152	61.05189	5.164057	1
3261	9.271355	181.2596	16540.98	7.022499	309.2389	487.6928	13.22844		4.333953	1
3262		134.7369	9000.026	9.026293		428.214	8.668672	74.77339	3.699558	1
3263	3.629922	244.1874	24856.63	6.618071	366.9679	442.0763	13.30288	59.48929	4.754826	1
3264	8.378108	198.5112	28474.2	6.477057	319.4772	499.867	15.38908	35.2212	4.524693	1
3265	6.923636	260.5932	24792.53	5.501164	332.2322	607.7736	15.48303	51.53587	4.013339	1
3266	5.893103	239.2695	20526.67	6.349561	341.2564	403.6176	18.96371	63.84632	4.390702	1
3267	8.197353	203.1051	27701.79	6.472914	328.8868	444.6127	14.25088	62.90621	3.361833	1
3268	8.37291	169.0871	14622.75	7.547984		464.5256	11.08303	38.43515	4.906358	1
3269	8.9899	215.0474	15921.41	6.297312	312.931	390.4102	9.899115	55.0693	4.613843	1
3270	6.702547	207.3211	17246.92	7.708117	304.5102	329.266	16.2173	28.8786	3.442983	1
3271	11.49101	94.81255	37188.83	9.263166	258.9306	439.8936	16.17276	41.5585	4.369264	1
3272	6.069616	186.659	26138.78	7.747547	345.7003	415.887	12.06762	60.41992	3.669712	1
3273	4.668102	193.6817	47580.99	7.166639	359.9486	526.4242	13.89442	66.68769	4.435821	1
3274	7.808856	193.5532	17329.8	8.061362		392.4496	19.90323		2.798243	1
3275	9.41951	175.7626	33155.58	7.350233		432.0448	11.03907	69.8454	3.298875	1

Importance of loading and processing dataset:

1. Accurate data loading and processing are essential for water analysis as they ensure the reliability of results and decisions made regarding water quality and safety.
2. Proper data handling enables the identification of trends, anomalies, and critical patterns in water quality, allowing for timely intervention in case of contamination or environmental changes.
3. Effective data processing enhances the efficiency of modeling and predictive algorithms, aiding in the development of early warning systems for water-related issues.
4. Overall, the quality and precision of water analysis heavily depend on meticulous data loading and processing, which are vital for safeguarding public health and the environment.
5.

Missing Data:

1. Imputation methods like mean, median, or regression can be used to estimate missing values.
2. Careful consideration of the missing data mechanism (e.g., missing completely at random or not) is crucial to choose the right imputation technique.
3. Robust data handling and imputation practices are essential to maintain the integrity of water quality assessments and ensure the safety of water resources.
4.

Scaling the features:

Scaling features in water analysis is critical to harmonize variables with varying units and scales, enabling a fair contribution of each feature to the analysis. Methods like min-max scaling, zscore standardization, log transformation, normalization, and robust scaling are employed based on data characteristics. Min-max scaling preserves relative relationships within a predefined range, while z-score standardization provides compatibility for scale-sensitive algorithms. Log transformation helps with skewed data, normalization ensures unit norm scaling, and robust scaling mitigates outlier influence. The choice of scaling method is data-dependent, promoting unbiased water quality assessments by mitigating feature magnitude disparities.

1.Loading the dataset:

To load a dataset, identify the data source and use programming languages or tools like Python with libraries (e.g., pandas) to read and import the data, followed by data verification and preprocessing.

1.Identify the dataset:

Identifying a dataset involves recognizing its source, content, and relevance, typically through descriptive metadata, file format, and documentation. Understanding the dataset's structure and context is crucial for effective analysis and interpretation.

2.Load the Dataset:

To load a dataset, use a programming language or tool to read the data from its source, such as a CSV file or a database.

Program:
trans = pd.read_csv(' ') trans.shape trans.head(10)

```
trans = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/20140711.CSV')
trans.shape
trans.head(10)
```

(10857234, 6)

	TripID	RouteID	StopID	StopName	WeekBeginning	NumberOfBoardings
0	23631	100	14156	181 Cross Rd	2013-06-30 00:00:00	1
1	23631	100	14144	177 Cross Rd	2013-06-30 00:00:00	1
2	23632	100	14132	175 Cross Rd	2013-06-30 00:00:00	1
3	23633	100	12266	Zone A Arndale Interchange	2013-06-30 00:00:00	2
4	23633	100	14147	178 Cross Rd	2013-06-30 00:00:00	1
5	23634	100	13907	9A Marion Rd	2013-06-30 00:00:00	1
6	23634	100	14132	175 Cross Rd	2013-06-30 00:00:00	1
7	23634	100	13335	9A Holbrooks Rd	2013-06-30 00:00:00	1
8	23634	100	13875	9 Marion Rd	2013-06-30 00:00:00	1
9	23634	100	13045	206 Holbrooks Rd	2013-06-30 00:00:00	1

3. Exploring data:
Exploring data involves analyzing, visualizing, and summarizing it to gain insights, identify patterns, and understand its characteristics, which is crucial for informed decision-making and further analysis.
Program:

```
df = pd.read_csv("/water_potability.csv")
```

```
[4] df.head()
```

Output

	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

+ Code + Text

Program:

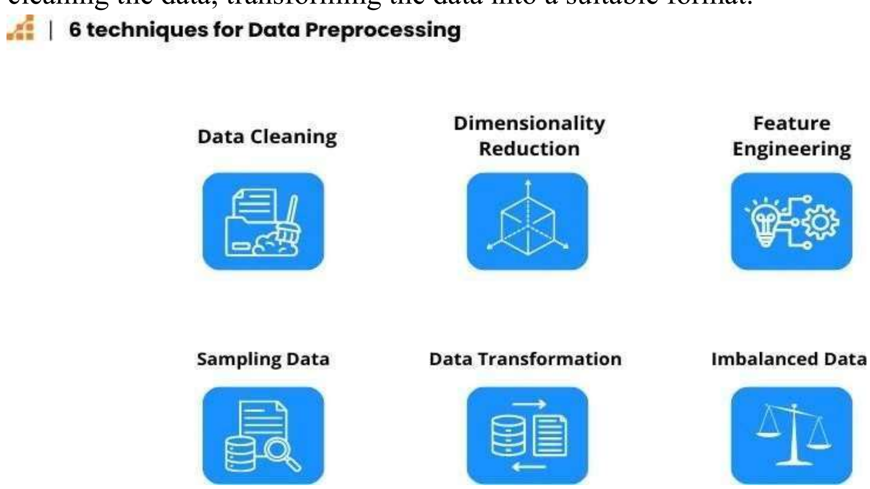
```
df.describe()
```

Output

	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

4.Preprocess the dataset:

Once the dataset is loaded into the machine learning environment, you may need to preprocess it before you can start training and evaluating your model. This may involve cleaning the data, transforming the data into a suitable format.



1. Data Cleaning: This step involves handling data imperfections like missing values and outliers. Missing values can be filled in using imputation methods, or rows with missing values can be removed. Outliers can be detected and treated, depending on the analysis requirements.

2. Feature Selection: Identifying relevant variables (features) and eliminating irrelevant ones. This reduces dimensionality and enhances the efficiency of the analysis.

3. Data Transformation: Transforming data to make it more suitable for analysis. This may include scaling numerical features, such as normalizing values to a common range, and encoding categorical variables into numerical format (e.g., one-hot encoding).

4. Data Integration: Combining data from different sources or tables, if applicable, to create a unified dataset for analysis.

5. Handling Imbalanced Data: If there is a class imbalance in the dataset (e.g., rare events), techniques like oversampling, undersampling, or the use of synthetic data can balance the classes.

6. Feature Engineering: Creating new features based on domain knowledge or relationships within the data can provide additional information for analysis.

7. Data Splitting: Dividing the dataset into training, validation, and testing sets. This ensures that the model is developed and evaluated on different subsets of the data.

8. Dimensionality Reduction: Reducing the number of features, especially in high-dimensional datasets, using methods like Principal Component Analysis (PCA) or Linear Discriminant Analysis (LDA).

9. Handling Time-Series Data: If the dataset is time-series data, time-related aspects like data interpolation, aggregation, or rolling window statistics might be applied.

10. Quality Control: Ensuring data quality by checking for anomalies, inconsistencies, and data entry errors.

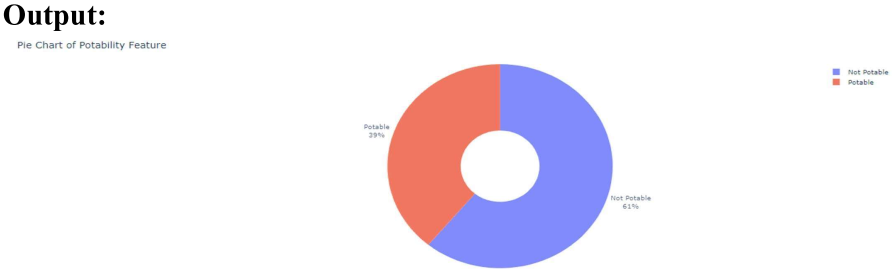
11. Data Normalization: Transforming data distributions to improve model performance, particularly for algorithms sensitive to data scale.

Data visualization:

Dependent Variable Analysis Program:

```
d = pd.DataFrame(df["Potability"].value_counts())
fig = px.pie(d, values = "Potability", names = ["Not Potable", "Potable"], hole = 0.35, opacity = 0.8,
            labels = {"label" :"Potability","Potability":"Number of Samples"})
fig.update_layout(title = dict(text = "Pie Chart of Potability Feature"))
fig.update_traces(textposition = "outside", textinfo = "percent+label")
fig.show()
```

Output:



Correlation Between Features

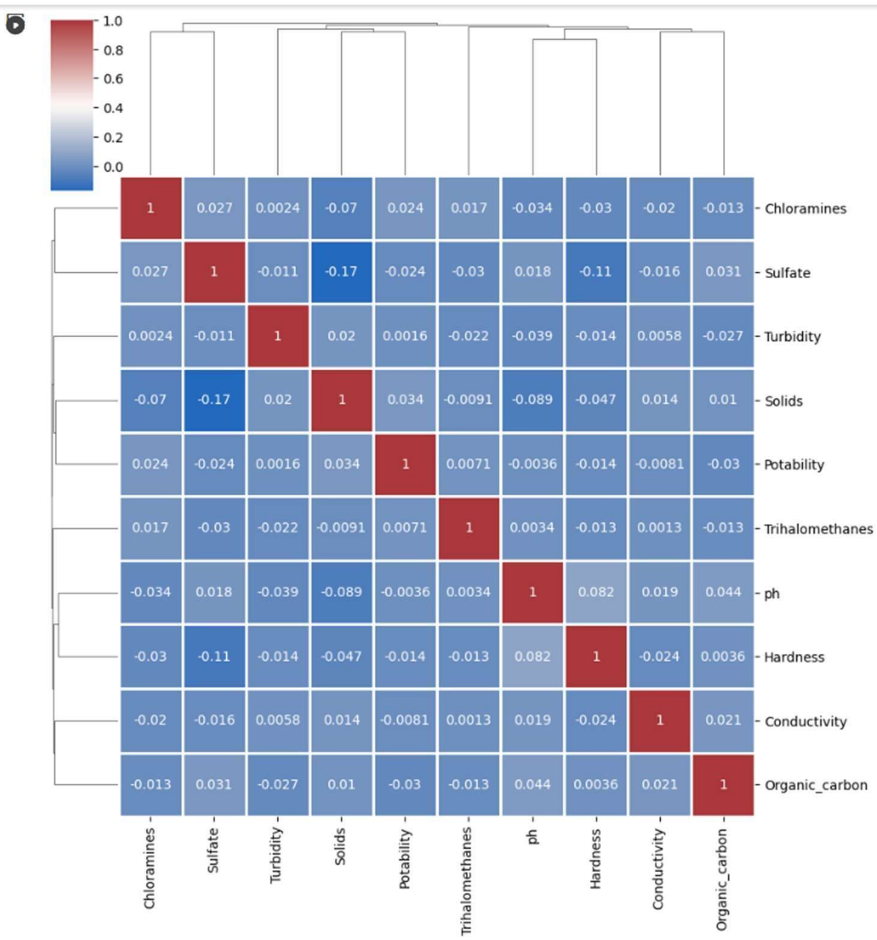
Program:

```
[ ] df.corr()
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
ph	1.000000	0.082096	-0.089288	-0.034350	0.018203	0.018614	0.043503	0.003354	-0.039057	-0.003556
Hardness	0.082096	1.000000	-0.046899	-0.030054	-0.106923	-0.023915	0.003610	-0.013013	-0.014449	-0.013837
Solids	-0.089288	-0.046899	1.000000	-0.070148	-0.171804	0.013831	0.010242	-0.009143	0.019546	0.033743
Chloramines	-0.034350	-0.030054	-0.070148	1.000000	0.027244	-0.020486	-0.012653	0.017084	0.002363	0.023779
Sulfate	0.018203	-0.106923	-0.171804	0.027244	1.000000	-0.016121	0.030831	-0.030274	-0.011187	-0.023577
Conductivity	0.018614	-0.023915	0.013831	-0.020486	-0.016121	1.000000	0.020966	0.001285	0.005798	-0.008128
Organic_carbon	0.043503	0.003610	0.010242	-0.012653	0.030831	0.020966	1.000000	-0.013274	-0.027308	-0.030001
Trihalomethanes	0.003354	-0.013013	-0.009143	0.017084	-0.030274	0.001285	-0.013274	1.000000	-0.022145	0.007130
Turbidity	-0.039057	-0.014449	0.019546	0.002363	-0.011187	0.005798	-0.027308	-0.022145	1.000000	0.001581
Potability	-0.003556	-0.013837	0.033743	0.023779	-0.023577	-0.008128	-0.030001	0.007130	0.001581	1.000000

```
sns.clustermap(df.corr(), cmap = "vlag", dendrogram_ratio = (0.1, 0.2), annot = True, linewidths = .8, figsize = (9,10))
plt.show()
```

Output:

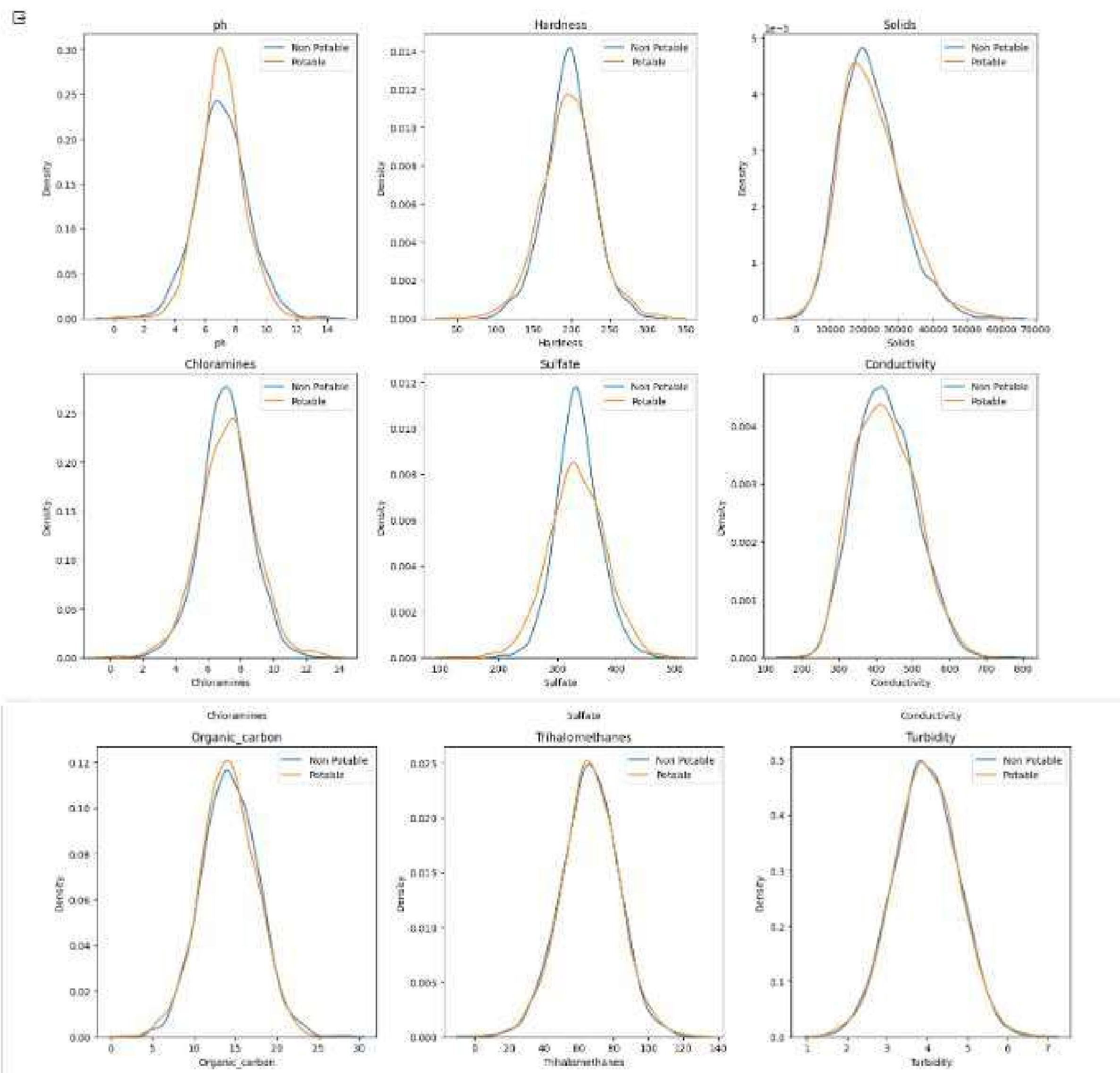


Distribution of Features:

Program:

```
non_potable = df.query("Potability == 0")
potable = df.query("Potability == 1")

plt.figure(figsize = (15,15))
for ax, col in enumerate(df.columns[:9]):
    plt.subplot(3,3, ax + 1)
    plt.title(col)
    sns.kdeplot(x = non_potable[col], label = "Non Potable")
    sns.kdeplot(x = potable[col], label = "Potable")
    plt.legend()
plt.tight_layout()
```

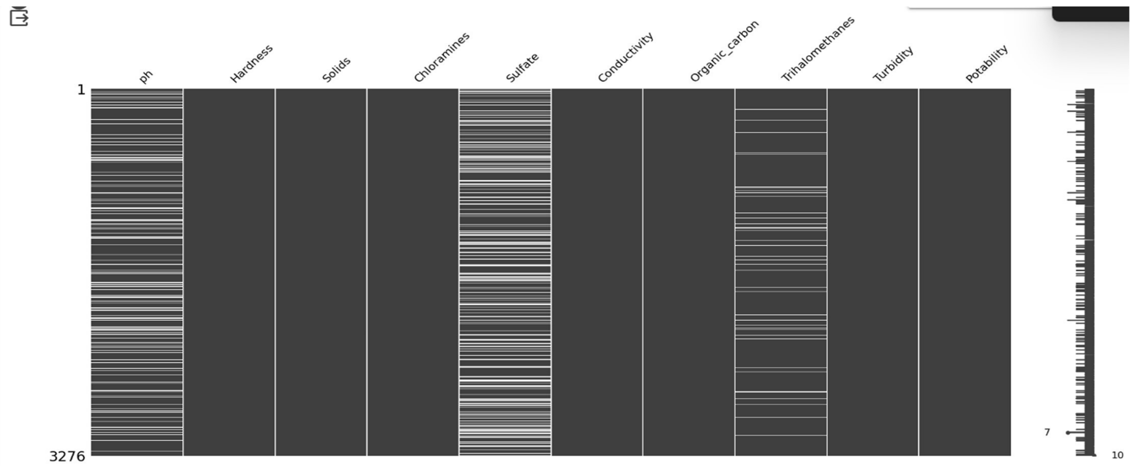
Output:



Preprocessing: Missing Value Problem:

```
msno.matrix(df)
plt.show()
```

Output:




```
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
```

```
[ ] df["ph"].fillna(value = df["ph"].mean(), inplace = True)
    df["Sulfate"].fillna(value = df["Sulfate"].mean(), inplace = True)
    df["Trihalomethanes"].fillna(value = df["Trihalomethanes"].mean(), inplace = True)
```

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

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

Preprocessing: Train-Test Split and Normalization:

Program:

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

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 3)
print("X_train",X_train.shape)
print("X_test",X_test.shape)
print("y_train",y_train.shape)
print("y_test",y_test.shape)
```

```
X_train (2293, 9)
X_test (983, 9)
y_train (2293,)
y_test (983,)
```

```
[ ] x_train_max = np.max(X_train)
    x_train_min = np.min(X_train)
    X_train = (X_train - x_train_min)/(x_train_max-x_train_min)
    X_test = (X_test - x_train_min)/(x_train_max-x_train_min)
```

Modelling: Decision Tree:

Program:

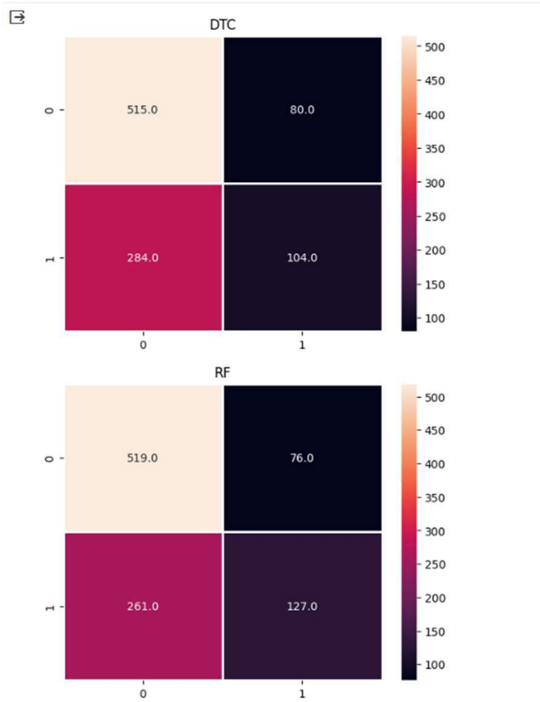
```
models = [("DTC", DecisionTreeClassifier(max_depth = 3)),
          ("RF",RandomForestClassifier()),
         ]
```

```
[ ] finalResults = []
    cmList = []
    for name, model in models:
        model.fit(X_train, y_train) # train
        model_result = model.predict(X_test) # prediction
        score = precision_score(y_test, model_result)
        cm = confusion_matrix(y_test, model_result)

        finalResults.append((name, score))
        cmList.append((name, cm))
    finalResults
```

```
[('DTC', 0.5652173913043478), ('RF', 0.625615763546798)]
```

```
[ ] for name, i in cmList:
        plt.figure()
        sns.heatmap(i, annot = True, linewidths = 0.8, fmt = ".1f")
        plt.title(name)
        plt.show()
```

Output:



Visualize Decision Tree:

Program:

```
dt_clf = models[0][1]
dt_clf
```

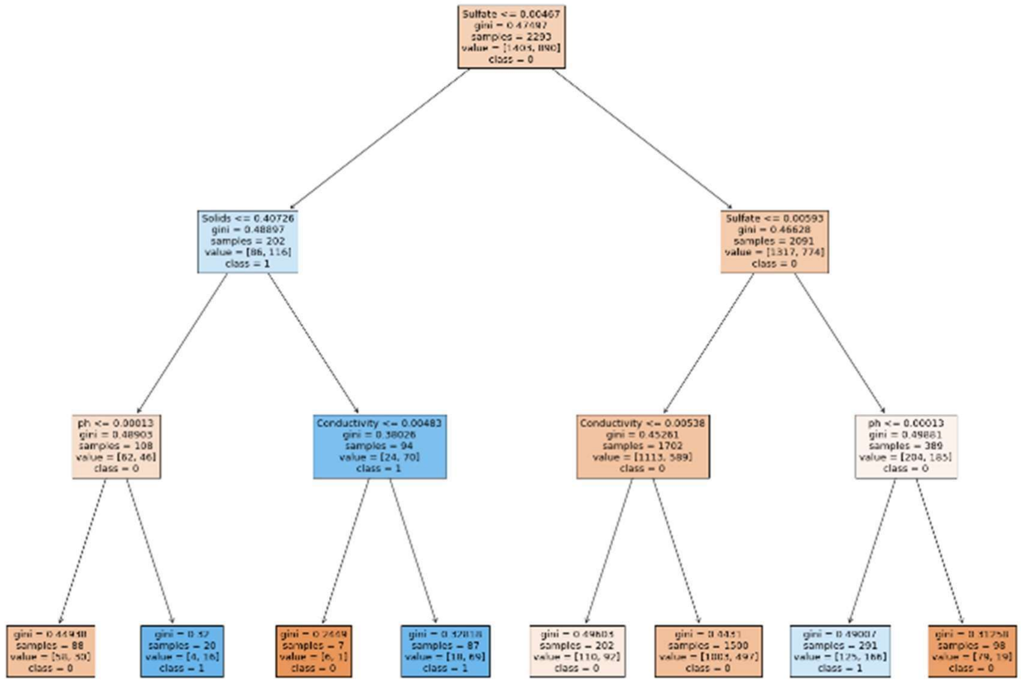
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3)
```

```
DecisionTreeClassifier(max_depth=3)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3)
```

```
plt.figure(figsize = (25,20))
tree.plot_tree(dt_clf,
               feature_names =  df.columns.tolist()[:-1],
               class_names = ["0", "1"],
               filled = True,
               precision = 5)
plt.show()
```

Output:



Modelling:

Split the data and standardizing them!

Program:

[13]:

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

[14]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y , test_size=0.3, random_state=101)
```

[15]:

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# This data is imbalanced that we have more Potability -0 than 1. We will oversample in the minority class
smt = SMOTE()
X_train, y_train = smt.fit_resample(X_train, y_train)
```

We will create functions to look at AUC graph, confusion matrix and test value score to determine whether this model is valid,

[16]:
```
from sklearn import metrics

# Creating AUC plot

def model_graphs(model, model_name):

    y_pred_prob = model.predict_proba(X_test)[::,1]
    fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_prob)
    auc = metrics.roc_auc_score(y_test, y_pred_prob)
    plt.plot(fpr,tpr,label= model_name +" auc="+str(auc))
    plt.legend(loc=4)
    plt.show()
```

[17]:
```
# Create confusion matrix to check accuracy, F1 score, and other

def confusion_matrix_graphs(y_pred):

    sns.heatmap(pd.DataFrame(confusion_matrix(y_test, y_pred)), annot=True, cmap="YlGnBu" ,fmt='g')
    ax.xaxis.set_label_position("top")
    plt.tight_layout()
    plt.title('Confusion matrix', y=1.1)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.show()
```

+ Code + Markdown

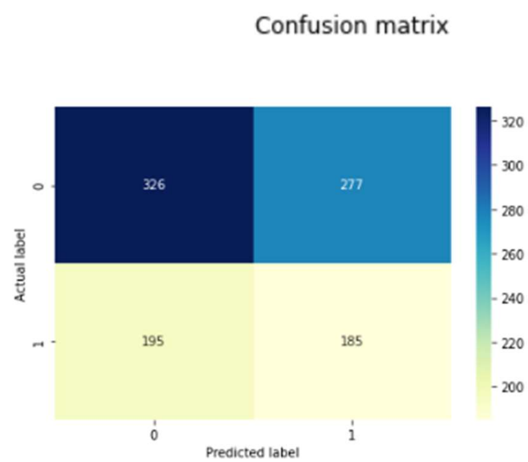
[18]:
```
# 5 folds validation and check the means accuracy score

def test_val_score(model):
    model_cross_val_score = cross_val_score(model, X_test, y_test, scoring='accuracy', cv = 5).mean()

    print("============================================================")

    print("The 5 fold cross value score is {:.2f}". format(model_cross_val_score))

    print("============================================================")
```

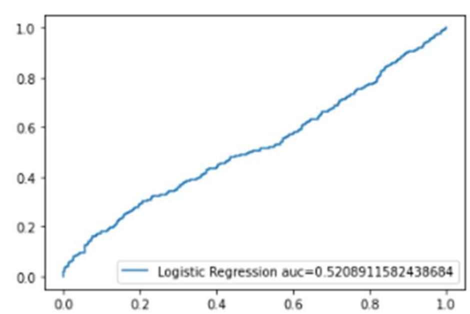
Logistric Regression:

[19]:
```
from sklearn.model_selection import cross_val_score

lr = LogisticRegression()
lr.fit(X_train, y_train)
y_lr_pred = lr.predict(X_test)

test_val_score(lr)


print(classification_report(y_lr_pred, y_test))

confusion_matrix_graphs(y_lr_pred)
model_graphs(lr, "Logistic Regression")
```

Output:

```
==========================================================
The 5 fold cross value score is 0.61
==========================================================
              precision    recall  f1-score   support

           0       0.54      0.63      0.58       521
           1       0.49      0.40      0.44       462

    accuracy                           0.52       983
   macro avg       0.51      0.51      0.51       983
weighted avg       0.52      0.52      0.51       983
```





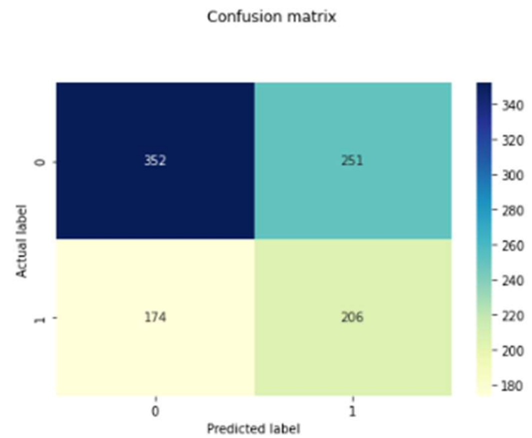
Decision Tree:

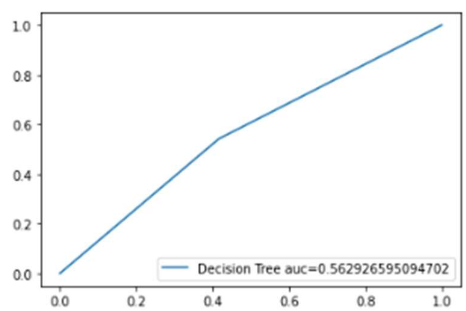
Program:

```
dt = DecisionTreeClassifier(random_state=42)
dt = dt.fit(X_train, y_train)
y_dt_pred = dt.predict(X_test)

test_val_score(dt)

print(classification_report(y_dt_pred, y_test))
confusion_matrix_graphs(y_dt_pred)
model_graphs(dt, "Decision Tree")
```

Output:

```
==========================================================
The 5 fold cross value score is 0.57
==========================================================
              precision    recall  f1-score   support

           0       0.58      0.67      0.62       526
           1       0.54      0.45      0.49       457

    accuracy                           0.57       983
   macro avg       0.56      0.56      0.56       983
weighted avg       0.56      0.57      0.56       983
```





\+ Code   + Markdown

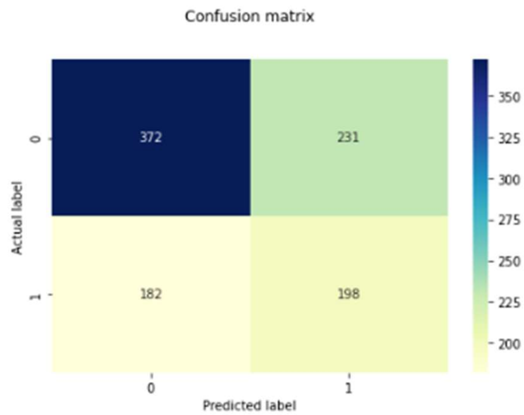
KNN:

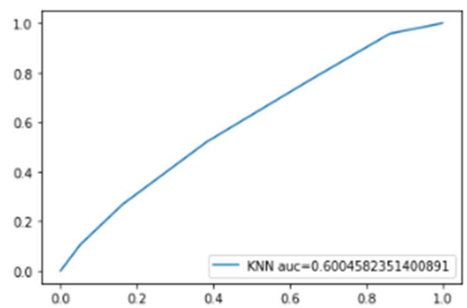
Program:

```
[22]:
KNN = KNeighborsClassifier()
KNN = KNN.fit(X_train, y_train)
y_knn_pred = KNN.predict(X_test)

test_val_score(KNN)

print(classification_report(y_knn_pred, y_test))

confusion_matrix_graphs(y_knn_pred)

model_graphs(KNN, "KNN")
```

Output:

```
==========================================================
The 5 fold cross value score is 0.61
==========================================================
              precision    recall  f1-score   support

           0       0.62      0.67      0.64       554
           1       0.52      0.46      0.49       429

    accuracy                           0.58       983
   macro avg       0.57      0.57      0.57       983
weighted avg       0.58      0.58      0.58       983
```





Naive Bayes:

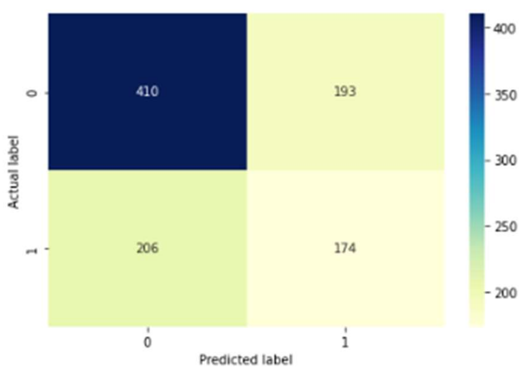
Program:

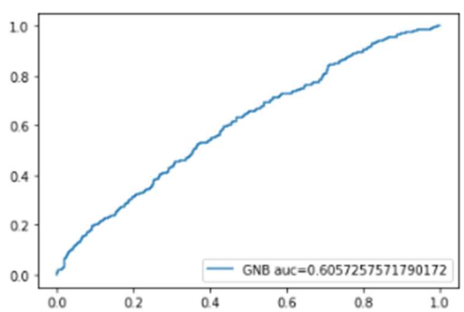
```
[23]:
GNB = GaussianNB()
GNB = GNB.fit(X_train, y_train)
y_GNB_pred = GNB.predict(X_test)

test_val_score(GNB)

print(classification_report(y_GNB_pred, y_test))

confusion_matrix_graphs(y_GNB_pred)

model_graphs(GNB, "GNB")
```

Output:

```
==========================================================
The 5 fold cross value score is 0.63
==========================================================
              precision    recall  f1-score   support

           0       0.68      0.67      0.67       616
           1       0.46      0.47      0.47       367

    accuracy                           0.59       983
   macro avg       0.57      0.57      0.57       983
weighted avg       0.60      0.59      0.60       983
```

Confusion matrix





Outcomes:

1. **Data Understanding:**
    - Start by importing the necessary libraries and loading the water quality datasets. Mention the source of the data and any data preprocessing steps.
    - Provide information on the tools and data sources used, such as water quality monitoring stations or databases.
2. **Visualization:**
    - Create various visualizations to represent the water quality data graphically. This can include line charts, scatter plots, and maps to visualize trends in water quality over time and across different locations.

- Visualizations can also show key parameters like pH levels, turbidity, dissolved oxygen, and pollutant concentrations. Heatmaps can be used to display spatial variations in water quality.

3. **Statistical Analysis:**
   - Conduct statistical analysis on important attributes related to water quality, such as calculating mean, standard deviation, and quartiles for various parameters.
   - Analyze trends and changes in water quality over time and assess variations in different geographic regions or water bodies.
4. **Insights and Interpretation:**
   - Throughout the documentation, provide insights and interpretations of the water quality data. For example, highlight areas with consistently poor water quality, identify factors influencing water quality, and explore correlations between parameters.
   - Summarize key findings related to water quality trends and potential issues, such as pollution sources or seasonal variations.
5. **Customization:**
   - Emphasize the flexibility of the documentation and how it can be customized to suit specific water quality analysis needs. Users should be able to adapt and extend the documentation to focus on specific parameters or geographical regions of interest.
   - Encourage collaboration by making it easy for others to access and work with the data and analysis tools.

Conclusion:

In conclusion, water analysis is a critical process for assessing the quality and safety of water resources. It involves a series of steps, from data collection and preprocessing to in-depth analysis and interpretation. By rigorously examining water quality data, we can make informed decisions, safeguard public health, and protect the environment, ensuring the availability of clean and safe water for all.