ANALYSIS OF WATER QUALITY USING MACHINE LEARNING

PROJECT GUIDE: Prof. Suvarna Ranade

GROUP MEMBERS:

SHREYA ANKALKOPE - 51

PRIYANKA BAGUL -34

INTRODUCTION

In the face of growing concerns over water quality, this project harnesses the power of machine learning, specifically K-Nearest Neighbor (KNN) and Logistic Regression. Leveraging a dataset sourced from **Kaggle**, the analysis aims to uncover patterns and indicators crucial for understanding water quality dynamics using **KNN Classification and Logistic Regression.**

We choose 3 metals from 20 metals provided in dataset:

- Aluminium - dangerous if greater than **2.8**
- Ammonia - dangerous if greater than **10**
- Barium - dangerous if greater than **2**

REASON:

After analysing 20 metals present in our dataset, we have selected **aluminium, ammonia, and barium** for water quality analysis. These metals have been identified as posing potential dangers to human health, and their presence in water can render it unsafe for drinking.

1] Data Loading and Preparation:

# We found through exploratory data analysis that our data is clean without any null or missing values.

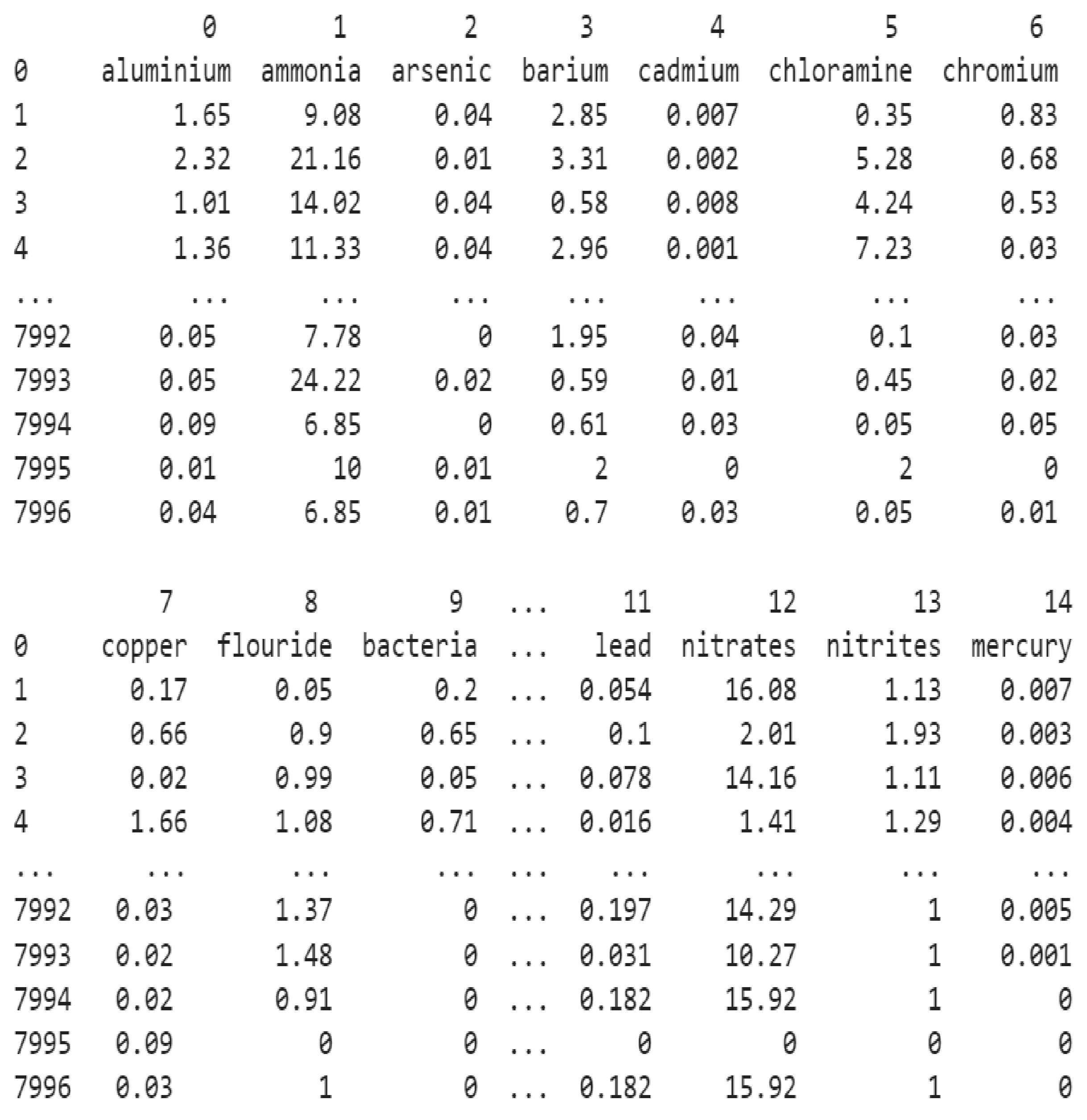
# Importing all the libraries which are required.

# A dataset with 8000 rows and 21 columns is loaded into a Data frame.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

#The libraries are used for data manipulation and analysis **(pandas)**, data visualization **(matplotlib.pyplot),** numerical operations **(numpy),** synthetic data generation for clustering **(make_blobs),** k-nearest neighbors classification **(KNeighborsClassifier),** and splitting datasets for training and testing **(train_test_split)** in machine learning.

```
df=pd.read_csv("/content/waterQuality1 (1).csv",
header = None)
dataset1= pd.DataFrame(df)
print(dataset1)
```



```
             0         1        2       3        4          5         6
0     aluminium  ammonia  arsenic  barium  cadmium  chloramine  chromium
1          1.65     9.08     0.04    2.85    0.007        0.35      0.83
2          2.32    21.16     0.01    3.31    0.002        5.28      0.68
3          1.01    14.02     0.04    0.58    0.008        4.24      0.53
4          1.36    11.33     0.04    2.96    0.001        7.23      0.03
...         ...      ...      ...     ...      ...         ...       ...
7992     0.05      7.78        0    1.95     0.04         0.1      0.03
7993     0.05     24.22     0.02    0.59     0.01        0.45      0.02
7994     0.09      6.85        0    0.61     0.03        0.05      0.05
7995     0.01        10     0.01       2        0           2         0
7996     0.04      6.85     0.01     0.7     0.03        0.05      0.01

          7         8         9  ...     11        12        13       14
0     copper  flouride  bacteria  ...   lead  nitrates  nitrites  mercury
1       0.17      0.05       0.2  ...  0.054     16.08      1.13    0.007
2       0.66       0.9      0.65  ...    0.1      2.01      1.93    0.003
3       0.02      0.99      0.05  ...  0.078     14.16      1.11    0.006
4       1.66      1.08      0.71  ...  0.016      1.41      1.29    0.004
...      ...       ...       ...  ...    ...       ...       ...      ...
7992   0.03       1.37         0  ...  0.197     14.29         1    0.005
7993   0.02       1.48         0  ...  0.031     10.27         1    0.001
7994   0.02       0.91         0  ...  0.182     15.92         1        0
7995   0.09          0         0  ...      0         0         0        0
7996   0.03          1         0  ...  0.182     15.92         1        0
```

```
dataset1.shape
```

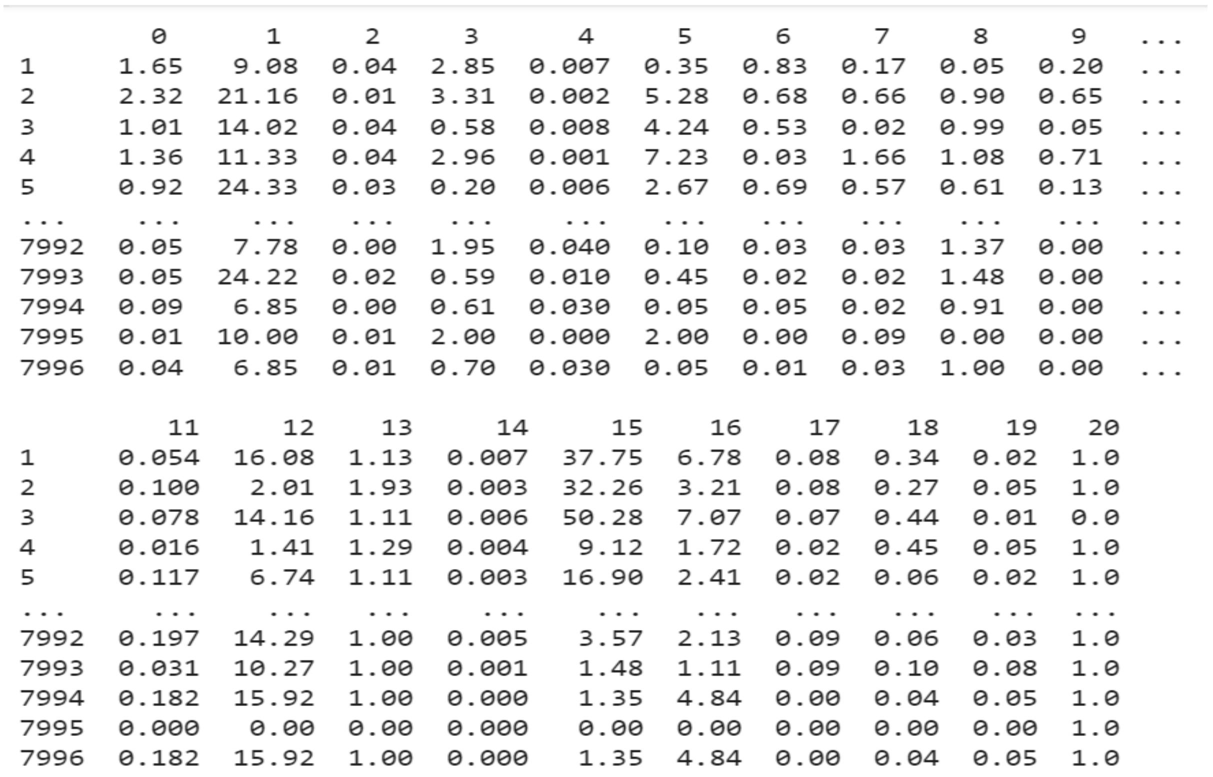
```
(8000, 21)
```

#8000 rows and 21 columns are loaded into a Data frame.

#The first row, assumed to contain column headers, is excluded from the analysis.

# The dataset is converted to a floating-point format for numerical analysis. As shown below:

```
dataset1=dataset1[1:]
dataset1= dataset1.astype(float)
print(dataset1)
```



```
         0      1     2     3      4     5     6     7     8     9   ...
1     1.65   9.08  0.04  2.85  0.007  0.35  0.83  0.17  0.05  0.20  ...
2     2.32  21.16  0.01  3.31  0.002  5.28  0.68  0.66  0.90  0.65  ...
3     1.01  14.02  0.04  0.58  0.008  4.24  0.53  0.02  0.99  0.05  ...
4     1.36  11.33  0.04  2.96  0.001  7.23  0.03  1.66  1.08  0.71  ...
5     0.92  24.33  0.03  0.20  0.006  2.67  0.69  0.57  0.61  0.13  ...
...    ...    ...   ...   ...    ...   ...   ...   ...   ...   ...  ...
7992  0.05   7.78  0.00  1.95  0.040  0.10  0.03  0.03  1.37  0.00  ...
7993  0.05  24.22  0.02  0.59  0.010  0.45  0.02  0.02  1.48  0.00  ...
7994  0.09   6.85  0.00  0.61  0.030  0.05  0.05  0.02  0.91  0.00  ...
7995  0.01  10.00  0.01  2.00  0.000  2.00  0.00  0.09  0.00  0.00  ...
7996  0.04   6.85  0.01  0.70  0.030  0.05  0.01  0.03  1.00  0.00  ...

          11     12    13     14     15    16    17    18    19   20
1     0.054  16.08  1.13  0.007  37.75  6.78  0.08  0.34  0.02  1.0
2     0.100   2.01  1.93  0.003  32.26  3.21  0.08  0.27  0.05  1.0
3     0.078  14.16  1.11  0.006  50.28  7.07  0.07  0.44  0.01  0.0
4     0.016   1.41  1.29  0.004   9.12  1.72  0.02  0.45  0.05  1.0
5     0.117   6.74  1.11  0.003  16.90  2.41  0.02  0.06  0.02  1.0
...     ...    ...   ...    ...    ...   ...   ...   ...   ...  ...
7992  0.197  14.29  1.00  0.005   3.57  2.13  0.09  0.06  0.03  1.0
7993  0.031  10.27  1.00  0.001   1.48  1.11  0.09  0.10  0.08  1.0
7994  0.182  15.92  1.00  0.000   1.35  4.84  0.00  0.04  0.05  1.0
7995  0.000   0.00  0.00  0.000   0.00  0.00  0.00  0.00  0.00  1.0
7996  0.182  15.92  1.00  0.000   1.35  4.84  0.00  0.04  0.05  1.0
```

2]Data Selection and Splitting:

# Columns 0, 1, 3, and 20 are selected for further analysis.

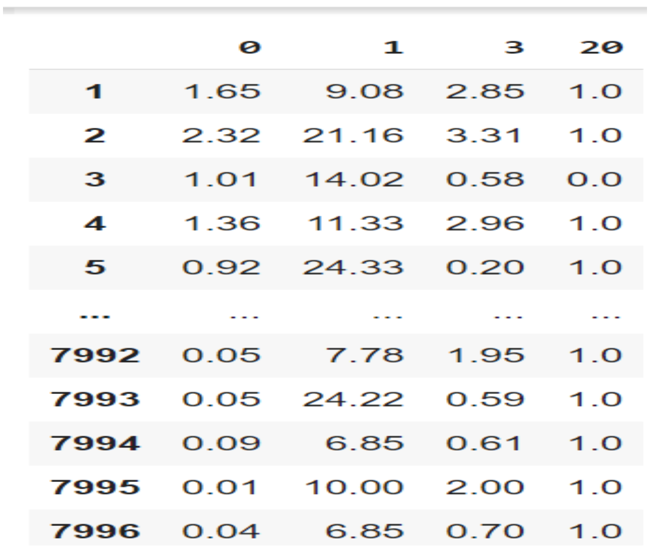
0 : Aluminium

1 : Ammonia

3 : Barium

20: is_safe

```
d = [0,1,3,20]
data=dataset1[d]
data
```



	0	1	3	20
1	1.65	9.08	2.85	1.0
2	2.32	21.16	3.31	1.0
3	1.01	14.02	0.58	0.0
4	1.36	11.33	2.96	1.0
5	0.92	24.33	0.20	1.0
...	...	...	...	...
7992	0.05	7.78	1.95	1.0
7993	0.05	24.22	0.59	1.0
7994	0.09	6.85	0.61	1.0
7995	0.01	10.00	2.00	1.0
7996	0.04	6.85	0.70	1.0

#The data is split into features (**X**) and target variable (**y**).

#A train-test split is performed with 80% of the data used for training and 20% for testing as data is large.

```
X = data.iloc[:, [0, 1, 2]].values
y = data.iloc[:, 3].values
```

```
#split the data into 80-20
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.20,
random_state=42)
```

3] KNN Classification :

#Standard scaling is applied to the features.

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X_train)
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
```

#A K-Nearest Neighbors (KNN) classifier with **n_neighbors=15** is trained on the training data.

```
from sklearn.neighbors import KNeighborsClassifier
classifier=KNeighborsClassifier(n_neighbors=20)
classifier.fit(X_train, y_train)
y_pred=classifier.predict(X_test)
y_pred
```

```
array([0., 1., 0., ..., 1., 0., 1.])
```

#The model is used to predict the target variable on the test set.

4]Model Evaluation (KNN Classification):

#The confusion matrix, classification report, and accuracy score are printed to evaluate the performance of the KNN classifier.

- **K=20:**

```
 from sklearn.neighbors import KNeighborsClassifier
classifier=KNeighborsClassifier(n_neighbors=20)
classifier.fit(X_train, y_train)
y_pred=classifier.predict(X_test)
y_pred
from sklearn.metrics import classification_report ,
confusion_matrix, accuracy_score

result= confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
result1=classification_report(y_test, y_pred)
print("classification_report:")
print(result1)
result2=accuracy_score(y_test, y_pred)
print("Accuracy:", result2)
```

```
Confusion Matrix:
[[2729   68]
 [ 243  159]]
classification_report:
              precision    recall  f1-score   support

         0.0       0.92      0.98      0.95      2797
         1.0       0.70      0.40      0.51       402

    accuracy                           0.90      3199
   macro avg       0.81      0.69      0.73      3199
weighted avg       0.89      0.90      0.89      3199

Accuracy: 0.9027821194123163
```

- **K=15:**

```
from sklearn.neighbors import KNeighborsClassifier
classifier=KNeighborsClassifier(n_neighbors=15)
classifier.fit(X_train, y_train)
y_pred=classifier.predict(X_test)
y_pred
from sklearn.metrics import classification_report ,
confusion_matrix, accuracy_score

result= confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
result1=classification_report(y_test, y_pred)
print("classification_report:")
print(result1)
result2=accuracy_score(y_test, y_pred)
print("Accuracy:", result2)
```

```
Confusion Matrix:

[[2741   56]

 [ 354   48]]
```

```
classification_report:
              precision    recall  f1-score   support

         0.0       0.89      0.98      0.93      2797
         1.0       0.46      0.12      0.19       402

    accuracy                           0.87      3199
   macro avg       0.67      0.55      0.56      3199
weighted avg       0.83      0.87      0.84      3199

Accuracy: 0.8718349484213817
```

Conclusion:

- The confusion matrix provides a detailed breakdown of the model's predictions.
- In this case, there are two classes: 0.0 and 1.0.
- The matrix shows that 2729 instances of class 0.0 were correctly predicted (True Negatives), 68 instances of class 0.0 were incorrectly predicted as class 1.0 (False Positives), 243 instances of class 1.0 were incorrectly predicted as class 0.0 (False Negatives), and 159 instances of class 1.0 were correctly predicted (True Positives).
- The overall accuracy of the KNN classifier is calculated as approximately 90.28% for k=20 and for k=15 its 87.1. **Therefore we can say that approximate range for k is 15 to 20.**

5]GRAPHICAL REPRESENTATION:

```
#KNN Classification
#generate isotopic gaussian blobs for clustering
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

The code is generating synthetic data using the **make_blobs** function from **sklearn.datasets**.

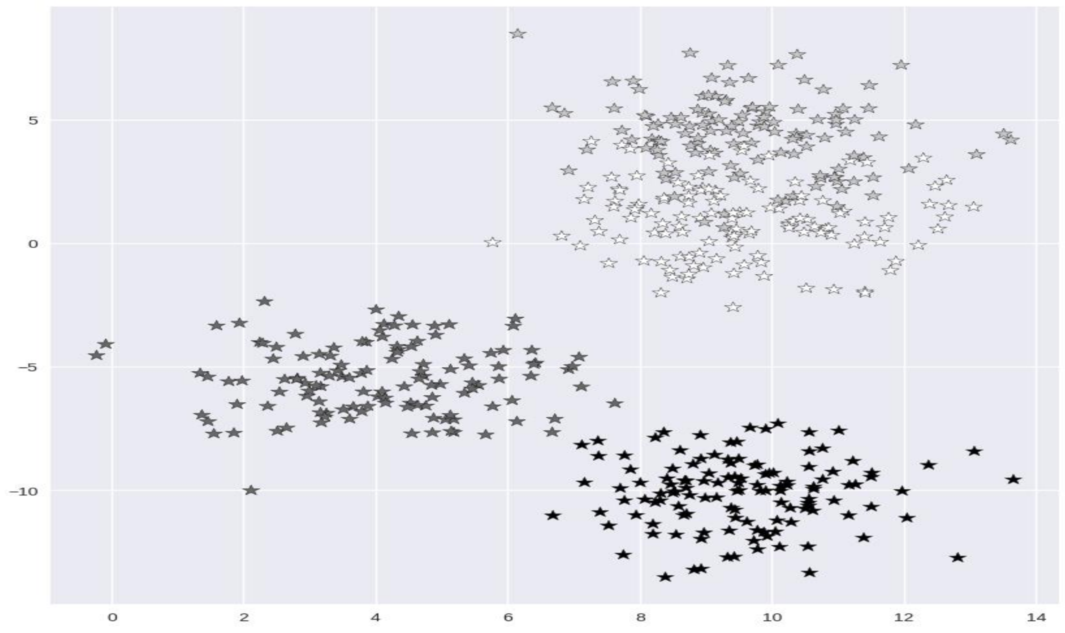
#The generated dataset consists of 500 samples (**n_samples**) with 2 features (**n_features**).

#The data is distributed across 4 clusters (**centers**) with a standard deviation of 1.5 (**cluster_std**).

```
X,y=make_blobs(n_samples=500, n_features=2,
centers=4, cluster_std=1.5, random_state=42)
plt.style.use('seaborn')
plt.figure(figsize=(10,10))
plt.scatter(X[:,0],X[:,1],c=y, marker='*', s=100,
edgecolors='black')
plt.show()
```

#Matplotlib is used to create a scatter plot for visualization.

#Each point in the scatter plot represents a data sample with its coordinates determined by the two features.



Interpretation:

- The generated isotopic Gaussian blobs visually represent a synthetic dataset with four distinct clusters.
- This synthetic dataset is suitable for demonstrating and testing K-Nearest Neighbors (KNN) classification algorithms, as it provides a clear separation between clusters.

6]Logistic Regression Model:

#A Logistic Regression model is created and trained.

#Here our features X are(Aluminum, Amonia, Barium) and Target variable Y is (is_safe)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

X = dataset1.iloc[:,0:3].values
X
```

```
array([[1.650e+00, 9.080e+00, 4.000e-02], [2.320e+00,
2.116e+01, 1.000e-02], [1.010e+00, 1.402e+01, 4.000e-
02], ..., [9.000e-02, 6.850e+00, 0.000e+00], [1.000e-
02, 1.000e+01, 1.000e-02], [4.000e-02, 6.850e+00,
1.000e-02]])
```

```
y = dataset1.iloc[:, 20].values
y
```

```
array([1., 1., 0., ..., 1., 1., 1.])
```

```python
# Extracting features and target variable
# Splitting 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)

# Creating and fitting the logistic regression model
model = LogisticRegression()
model
model.fit(X_train, y_train)
```

```python
# Accessing coefficients and intercept
coefficients = model.coef_
intercept = model.intercept_

print("Coefficients:", coefficients)
print("Intercept:", intercept)
```

```
Coefficients: [[ 0.72656026 -0.01882018 -3.93900696]]
Intercept: [-2.11878383]
```

#The coefficients and intercept of the model are printed.

```python
# Predicting on the test set
y_pred = model.predict(X_test)
y_pred
```

```
array([0., 1., 0., ..., 0., 1., 0.])
```

```python
# Evaluating the accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.884375
```

Conclusion:

- The logistic regression model achieved an accuracy of approximately 88.44% on the dataset which is good fit.

7] PREDICTION:

The logistic value is transformed into a probability using the sigmoid (logistic) function:

1/(1+e^(−logistic_value))

```
import numpy as np

# Coefficients and intercept
coefficients = np.array([0.72656026, -0.01882018, -
3.93900696])
intercept = -2.11878383

# Input features (replace these values with your
actual data)
Aluminium= 2
Ammonia= 7.97
Barium= 1.56

# Logistic regression equation
logistic_value = intercept + np.dot(coefficients,
np.array([Aluminium, Ammonia, Barium]))
probability = 1 / (1 + np.exp(-logistic_value))

print(f"Probability of Y=1: {probability}")
```

```
Probability of Y=1: 0.6734567
```

# Given the input feature values for Aluminium, Ammonia, and Barium, the logistic regression model calculates a probability of approximately **0.673** for the target variable Y being equal to 1(SAFE). **Which means water quality is good and safe for drinking.**

```
# Input features (replace these values with your
actual data)
Aluminium= 3
Ammonia= 12.56
Barium= 4.5

# Logistic regression equation
```

```
logistic_value = intercept + np.dot(coefficients,
np.array([Aluminium, Ammonia, Barium]))
probability = 1 / (1 + np.exp(-logistic_value))

print(f"Probability of Y=1: {probability}")
```

```
Probability of Y=0: 0.7167994
```

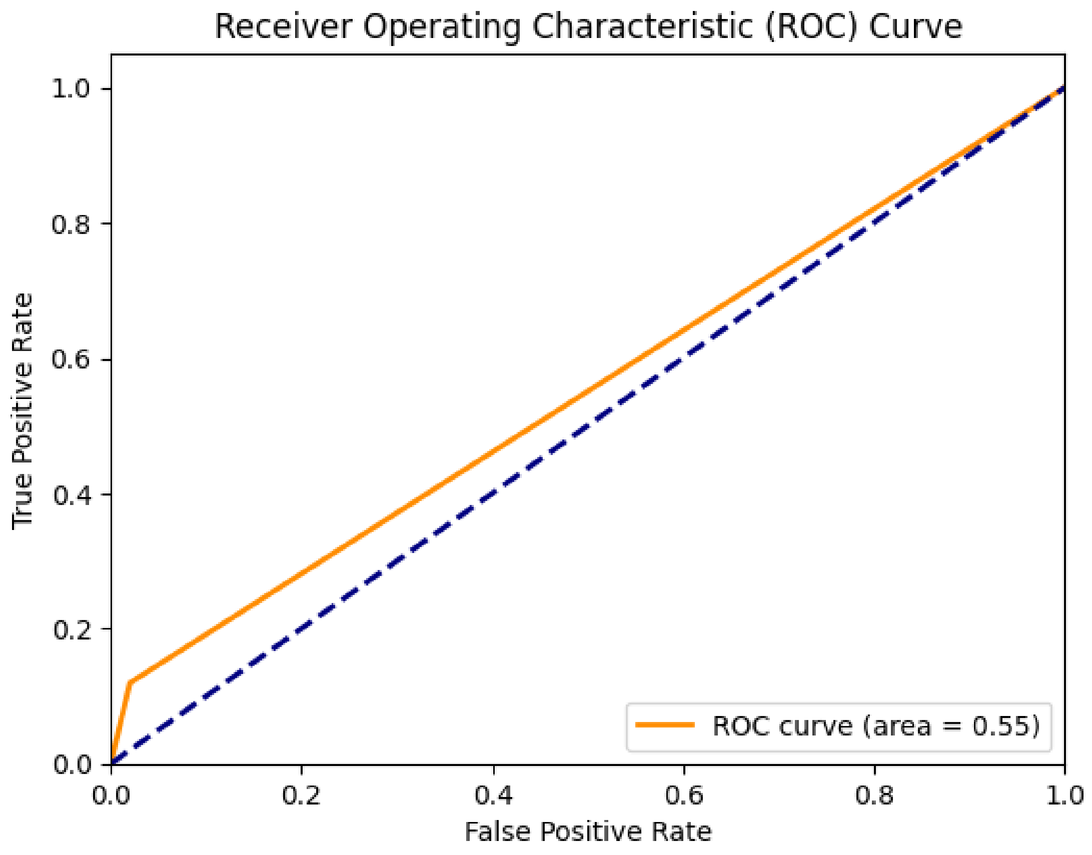
# Given the input feature values for Aluminium, Ammonia, and Barium, the logistic regression model calculates a probability of approximately **0.7167** for the target variable Y being equal to 0 (NOT SAFE) . **Which means water quality is dangerous and unsafe for drinking.**

#Therefore we can clearly see that if measure of metals are above the given threshold then the water becomes dangerous and unsafe for drinking.

8] AUC-ROC CURVE:

```
# Assuming y_true is the true labels and y_scores is
the predicted probabilities
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2,
label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2,
linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)
Curve')
plt.legend(loc="lower right")
plt.show()
```



INTERPRETATION:

- The straight section of the curve typically means that the model has a consistent performance across different thresholds.
- The bulge or curve at the bottom left suggests that the model might be too lenient at lower thresholds, leading to a higher false positive rate. This could be interpreted as the model being more sensitive but less specific.

“TRUST ME YOU CAN DRINK IT”

-

WATER

(WHAT YOU GUYS WERE THINKING…?)

🙃

THANK YOU…!!!