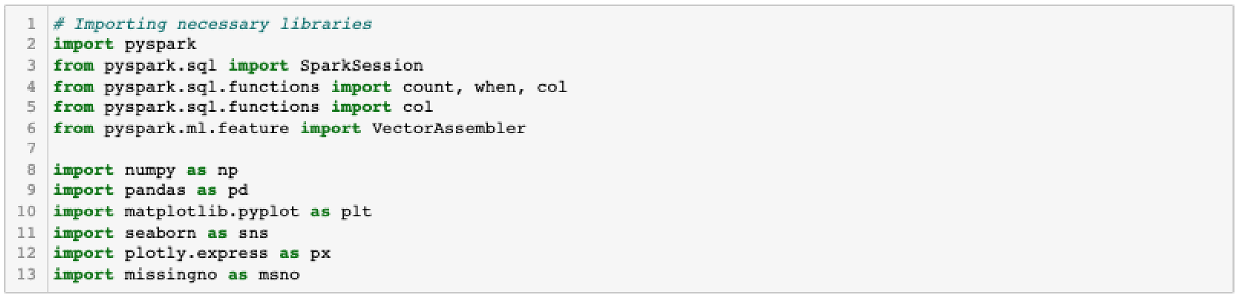
Analyzing Water Quality Data using Distributed Cloud Services and Spark

Abdoulaye Diallo
New York Institute Of Technology
New York, NY.
adiall04@nyit.edu

1. INTRODUCTION:

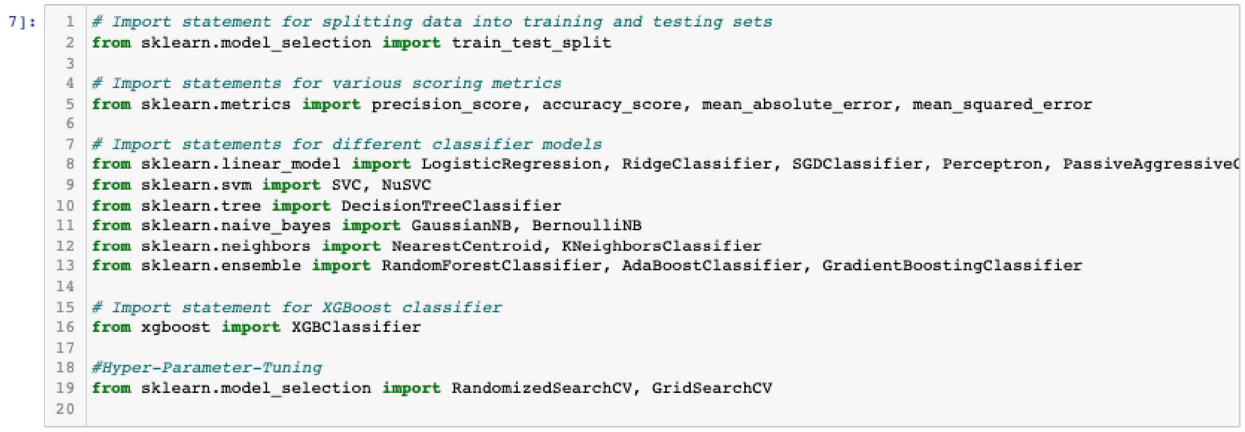
Having clean water is crucial for our health. Determining if water is safe involves many factors. Since water quality depends on the amounts of certain elements in it, we can create a model to check water quality based on these elements.

2. PACKAGES



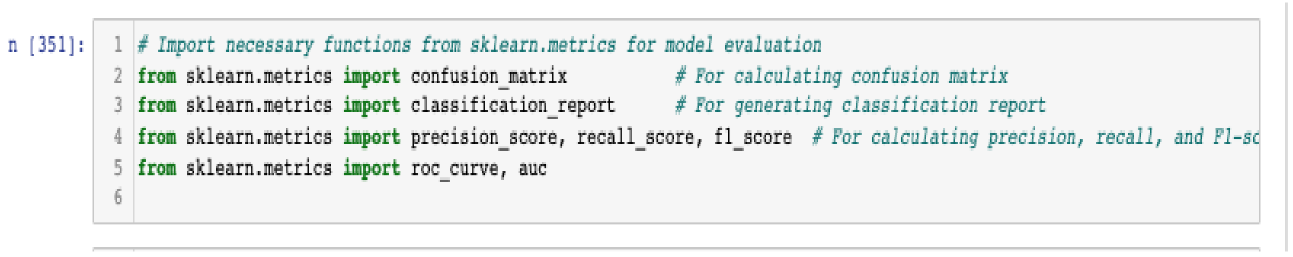
```python
# Importing necessary libraries
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import count, when, col
from pyspark.sql.functions import col
from pyspark.ml.feature import VectorAssembler

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import missingno as msno
```

PREPARING THE DATA FOR MODELLING



7]:

```python
# Import statement for splitting data into training and testing sets
from sklearn.model_selection import train_test_split

# Import statements for various scoring metrics
from sklearn.metrics import precision_score, accuracy_score, mean_absolute_error, mean_squared_error

# Import statements for different classifier models
from sklearn.linear_model import LogisticRegression, RidgeClassifier, SGDClassifier, Perceptron, PassiveAggressiveC
from sklearn.svm import SVC, NuSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB, BernoulliNB
from sklearn.neighbors import NearestCentroid, KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier

# Import statement for XGBoost classifier
from xgboost import XGBClassifier

#Hyper-Parameter-Tuning
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
```



n [351]:

```python
# Import necessary functions from sklearn.metrics for model evaluation
from sklearn.metrics import confusion_matrix          # For calculating confusion matrix
from sklearn.metrics import classification_report     # For generating classification report
from sklearn.metrics import precision_score, recall_score, f1_score  # For calculating precision, recall, and F1-sc
from sklearn.metrics import roc_curve, auc
```

3. BACKGROUND AND MOTIVATION:

Having access to safe drinking water is not only crucial for maintaining good health, but it's also recognized as a fundamental human right and a key element of well-rounded health protection policies. This significance extends across national, regional, and local contexts, serving as a vital concern for both health and developmental matters. Demonstrated in some areas, investments made in water supply and sanitation have proven to bring about a positive net economic outcome. This is due to the fact that the economic gains resulting from reduced health issues and lowered healthcare expenses outweigh the costs associated with implementing these interventions.

4. **DATA LAKE VS DATA STORE:**

Definitions:

Data warehouse:

A data warehouse is a specialized system for managing data, strategically crafted to facilitate and empower business intelligence (BI) projects, particularly analytics. These repositories are purpose-built for executing queries and conducting analyses, often housing extensive historical data. The content of a data warehouse typically originates from diverse origins, encompassing elements like application log files and transactional applications.

Data lake:

A data lake serves as a centralized storage hub where you can house all your structured and unstructured data, regardless of its volume. It permits the storage of data in its raw state, eliminating the need for upfront structuring. This setup facilitates a spectrum of analytical activities, spanning from creating dashboards and visualizations to engaging in substantial data processing, real-time analytics, and leveraging machine learning for informed decision-making.
I hosted the data in **AWS S3,** made it public ang granted a user full S3 Access which permitted me to get a **secret** and **access keys** that I used in a jupyter notebook to get the data using spark.
However, choosing between a Data Lake and a Data Warehouse depends on various factors such as the company goals and requirements, the nature of the data itself, the goals of the analysis and many factors. Let's consider some specific aspects to our given dataset. In this case, Lets lean more towards a data warehouse, even though S3 is mostly used as storage due to its flexibility, scalability and distributed across three **Availability zones.**

- **Data Variety and Flexibility:** If the water quality dataset includes a wide range of data types, such as sensor readings, images, text reports, and more, a Data Lake might be a better choice. Data Lakes excel at handling diverse and unstructured data types without the need for extensive upfront transformation.
- **Data Volume:** If the water quality dataset contains massive volumes of data, particularly if it's generated from multiple sources like sensors and monitoring stations, a Data Lake might be more scalable. Data Lakes can handle the raw and unaggregated data efficiently.
- **Data Exploration and Research:** If the primary goal is to explore the water quality data, conduct research, and gain insights into patterns and anomalies, a Data Lake can provide more flexibility. Researchers can directly access the raw data and perform exploratory analysis using various tools.
- **Regulatory Reporting and Compliance:** If the focus is on structured reporting, compliance, and regular business intelligence reporting related to water quality metrics, a Data Warehouse might be more suitable. Data Warehouses allow you to structure and model the data in a way that aligns with reporting requirements.

- **Data Quality and Governance:** If ensuring data quality and governance is crucial, a Data Warehouse can provide better control over data transformation and consistency. A well-structured Data Warehouse ensures that only high-quality, relevant data is used for analysis.
- **Future-Proofing and Agility:** If the water quality dataset is expected to evolve with new data sources, sensors, and changing requirements, a Data Lake offers future-proofing by storing raw data that can be transformed and structured as needs change.
- **Hybrid Approach:** Depending on the use cases, a hybrid approach could also be considered. Raw water quality data can be stored in a Data Lake for exploratory analysis, and refined and structured data can be moved to a Data Warehouse for reporting and structured analytics.

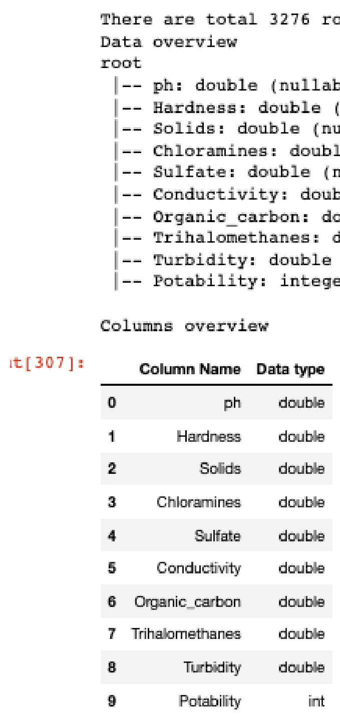
5. DATA CONTENT:

- **ph**: It is an important parameter in evaluating the acid–base balance of water. It is also the indicator of acidic or alkaline condition of water status.
- **Hardness**: Mainly caused by calcium and magnesium salts. These salts are dissolved from geologic deposits through which water travels. The length of time water is in contact with hardness producing material helps determine how much hardness there is in raw water. Hardness was originally defined as the capacity of water to precipitate soap caused by Calcium and Magnesium.
- **Solids**: Water has the ability to dissolve a wide range of inorganic and some organic minerals or salts such as potassium, calcium, sodium, bicarbonates, chlorides, magnesium, sulfates etc. These minerals produced an unwanted taste and diluted color in the appearance of water. This is the important parameter for the use of water.
- **Chlorine**: Chlorine and chloramine are the major disinfectants used in public water systems. Chloramines are most commonly formed when ammonia is added to chlorine to treat drinking water.
- **Sulfate**: Sulfates are naturally occurring substances that are found in minerals, soil, and rocks. They are present in ambient air, groundwater, plants, and food. The principal commercial use of sulfate is in the chemical industry.
- **Conductivity**: Pure water is not a good conductor of electric current rather it's a good insulator. Increase in ions concentration enhances the electrical conductivity of water. Generally, the amount of dissolved solids in water determines the electrical conductivity
- **Organic_carbon**: Total Organic Carbon (TOC) in source waters comes from decaying natural organic matter (NOM) as well as synthetic sources. TOC is a measure of the total amount of carbon in organic compounds in pure water.
- **Trihalomethanes**: THMs are chemicals which may be found in water treated with chlorine. The concentration of THMs in drinking water varies according to the level of organic material in the water, the amount of chlorine required to treat the water, and the temperature of the water that is being treated.

- **Turbidity**: The turbidity of water depends on the quantity of solid matter present in the suspended state. It is a measure of light emitting properties of water and the test is used to indicate the quality of waste discharge with respect to colloidal matter.
- **Potability**: Indicates if water is safe for human consumption where 1 means Potable and 0 means Not potable.

6. EXPLORATORY DATA ANALYSIS:

A. There are 3276 Observations.

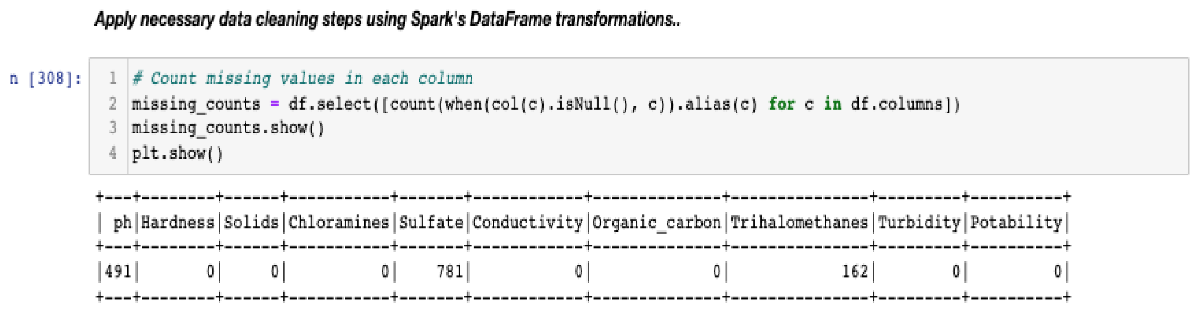


```
There are total 3276 row, Let print first 2 data rows:
Data overview
root
 |-- ph: double (nullable = true)
 |-- Hardness: double (nullable = true)
 |-- Solids: double (nullable = true)
 |-- Chloramines: double (nullable = true)
 |-- Sulfate: double (nullable = true)
 |-- Conductivity: double (nullable = true)
 |-- Organic_carbon: double (nullable = true)
 |-- Trihalomethanes: double (nullable = true)
 |-- Turbidity: double (nullable = true)
 |-- Potability: integer (nullable = true)

Columns overview
```

t[307]:

	Column Name	Data type
0	ph	double
1	Hardness	double
2	Solids	double
3	Chloramines	double
4	Sulfate	double
5	Conductivity	double
6	Organic_carbon	double
7	Trihalomethanes	double
8	Turbidity	double
9	Potability	int

B. The missing values are mostly on the sulfate (781), 'ph' (491) and 'Trihalomethanes(162) columns.



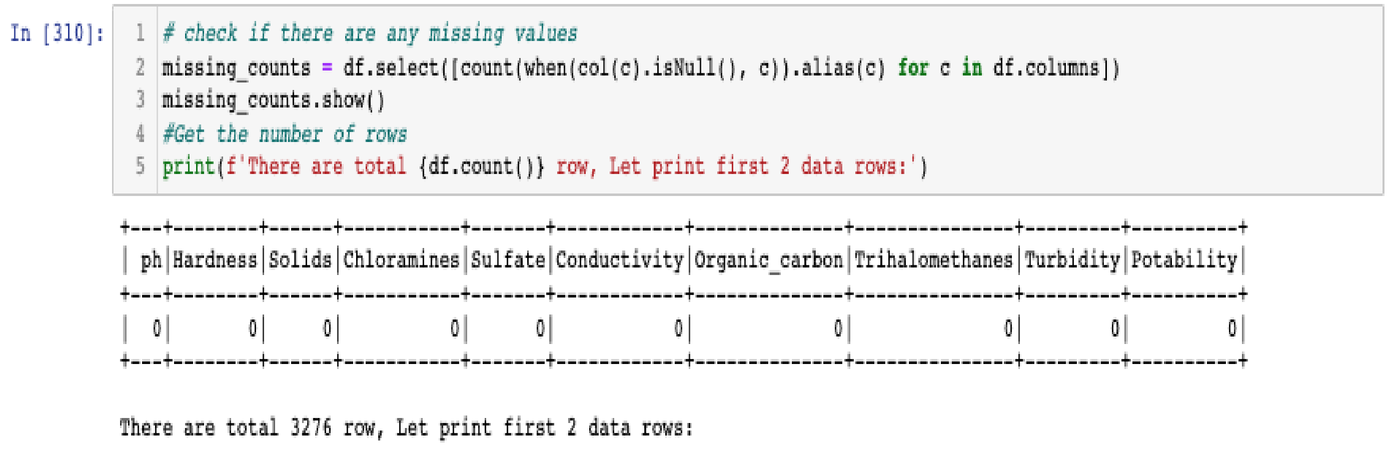
Apply necessary data cleaning steps using Spark's DataFrame transformations..

n [308]:

```python
# Count missing values in each column
missing_counts = df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns])
missing_counts.show()
plt.show()
```

```
+---+--------+------+-----------+-------+------------+--------------+---------------+---------+----------+
| ph|Hardness|Solids|Chloramines|Sulfate|Conductivity|Organic_carbon|Trihalomethanes|Turbidity|Potability|
+---+--------+------+-----------+-------+------------+--------------+---------------+---------+----------+
|491|       0|     0|          0|    781|           0|             0|            162|        0|         0|
+---+--------+------+-----------+-------+------------+--------------+---------------+---------+----------+
```

C. To solve the missing values, first create a list of the numeric columns (ph, Sulfate, and Trihalomethanes) where we have the missing values. Then , loop over that list and for each column, calculate the mean value and subsequently, replace missing values with the mean value. This will help us prepare the data for analysis.
D. Now we check if there is any missing value. Effectively, we find that there is no missing value.

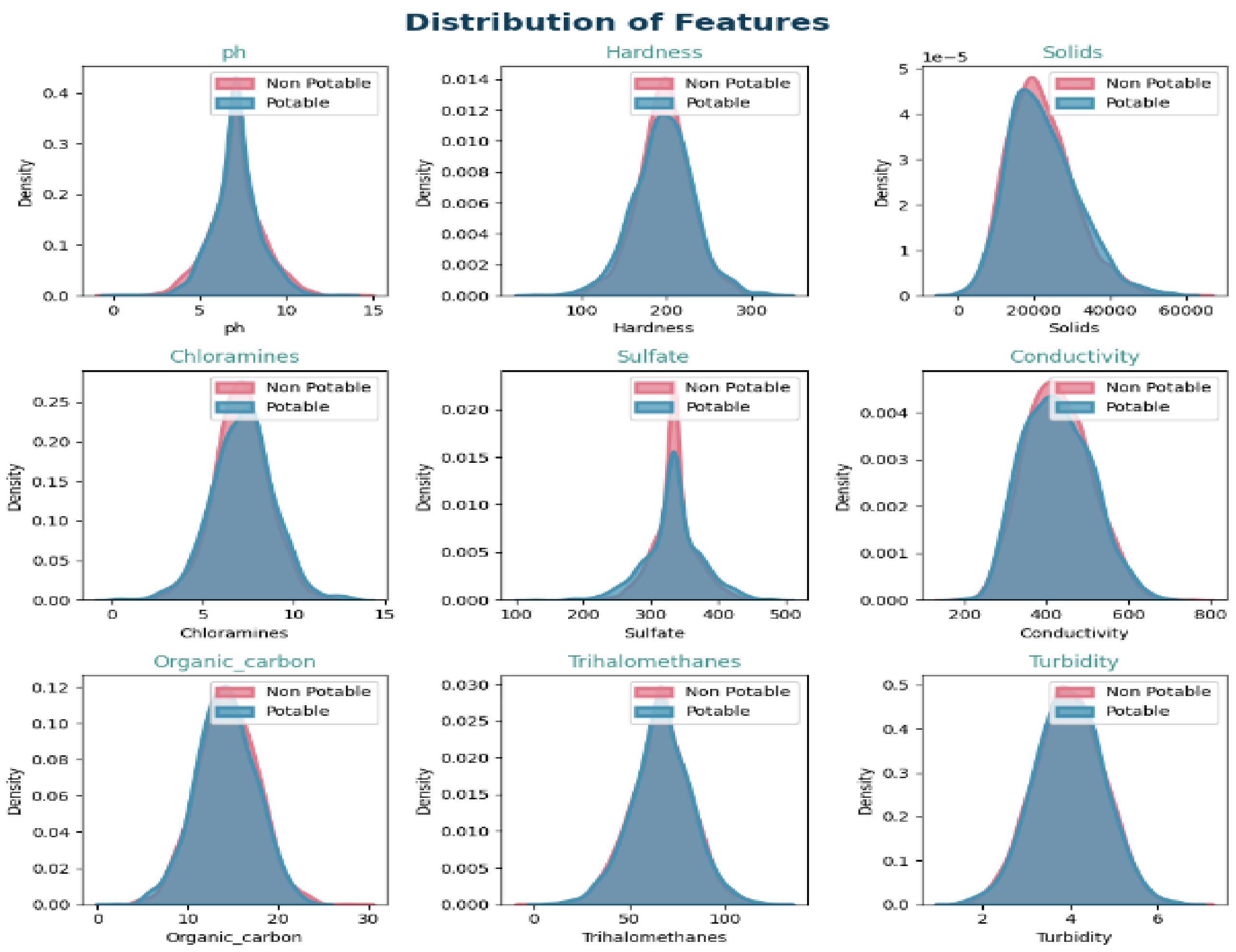


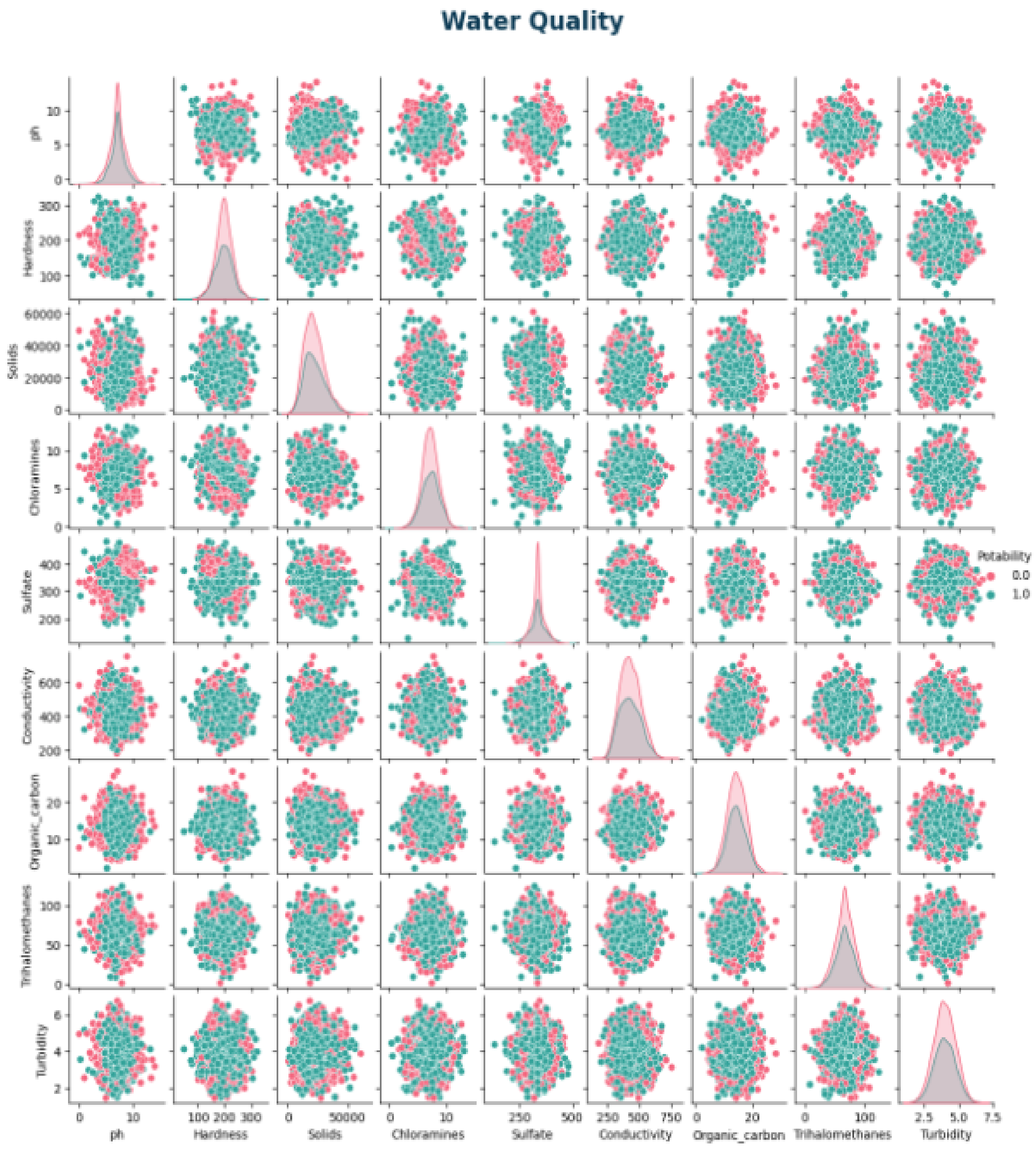
In [310]:

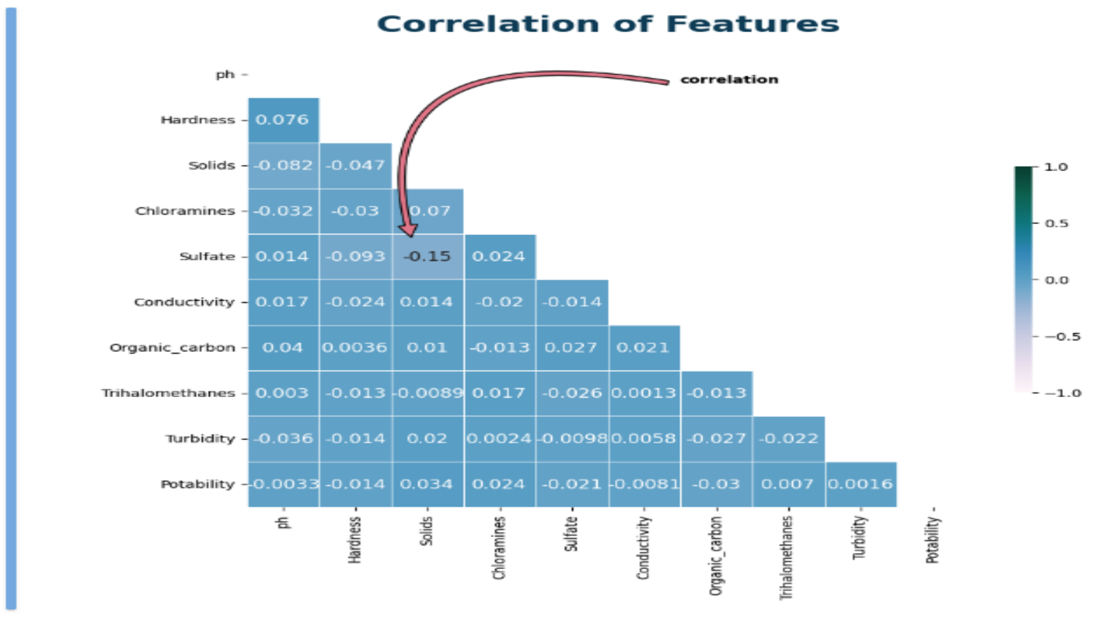
```
# check if there are any missing values
missing_counts = df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns])
missing_counts.show()
#Get the number of rows
print(f'There are total {df.count()} row, Let print first 2 data rows:')
```

```
+---+--------+------+-----------+-------+------------+--------------+---------------+---------+----------+
| ph|Hardness|Solids|Chloramines|Sulfate|Conductivity|Organic_carbon|Trihalomethanes|Turbidity|Potability|
+---+--------+------+-----------+-------+------------+--------------+---------------+---------+----------+
|  0|       0|     0|          0|      0|           0|             0|              0|        0|         0|
+---+--------+------+-----------+-------+------------+--------------+---------------+---------+----------+

There are total 3276 row, Let print first 2 data rows:
```

7. **DATA VISUALIZATION:** Since, there are no missing values, let’s visualize it to get a more intuitive way to understand the underlying patterns, trends, and relationships of our data.

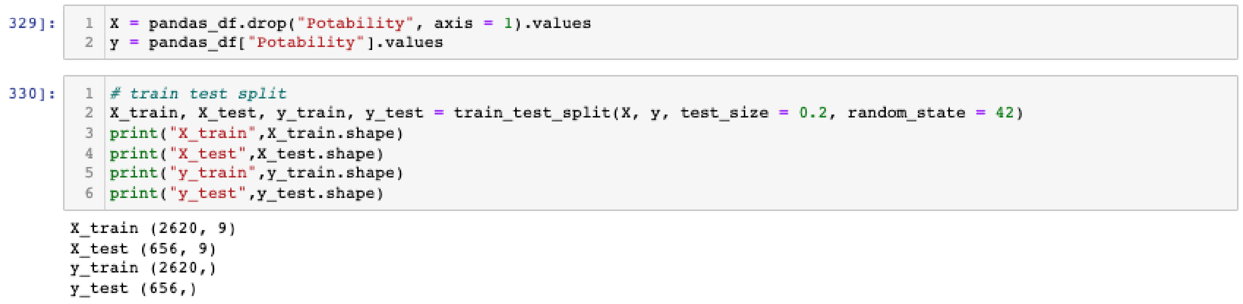

Distribution of Features
ph
Hardness
Solids
Chloramines
Sulfate
Conductivity
Organic_carbon
Trihalomethanes
Turbidity
Non Potable
Potable
Density
1e−5


Water Quality
ph
Hardness
Solids
Chloramines
Sulfate
Conductivity
Organic_carbon
Trihalomethanes
Turbidity
Potability
0.0
1.0



8. MODELIZATION:

I will 80% for training and 20% for testing the data

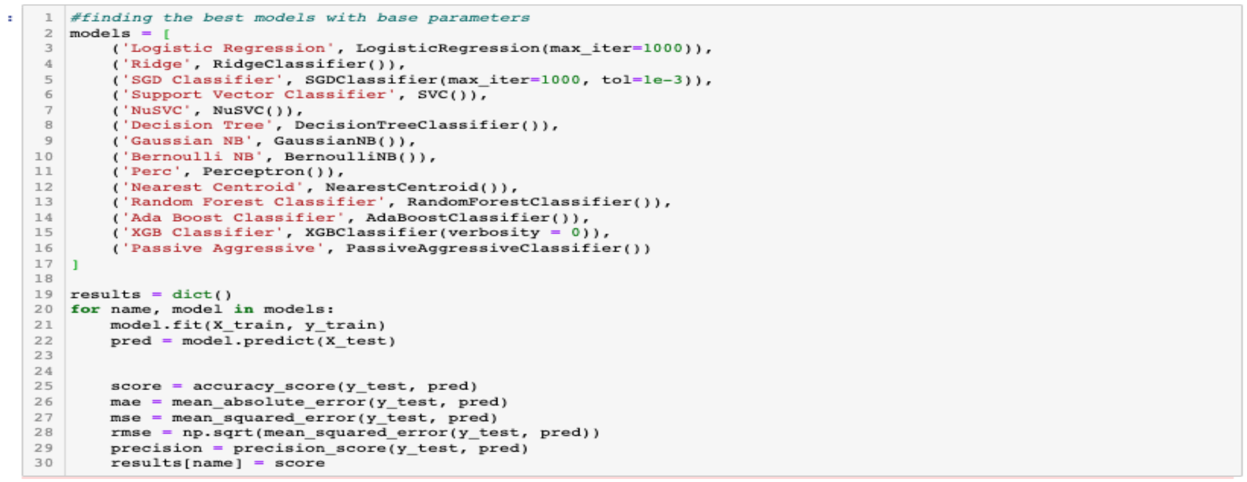


```
329]:
X = pandas_df.drop("Potability", axis = 1).values
y = pandas_df["Potability"].values
```

```
330]:
# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
print("X_train",X_train.shape)
print("X_test",X_test.shape)
print("y_train",y_train.shape)
print("y_test",y_test.shape)
```

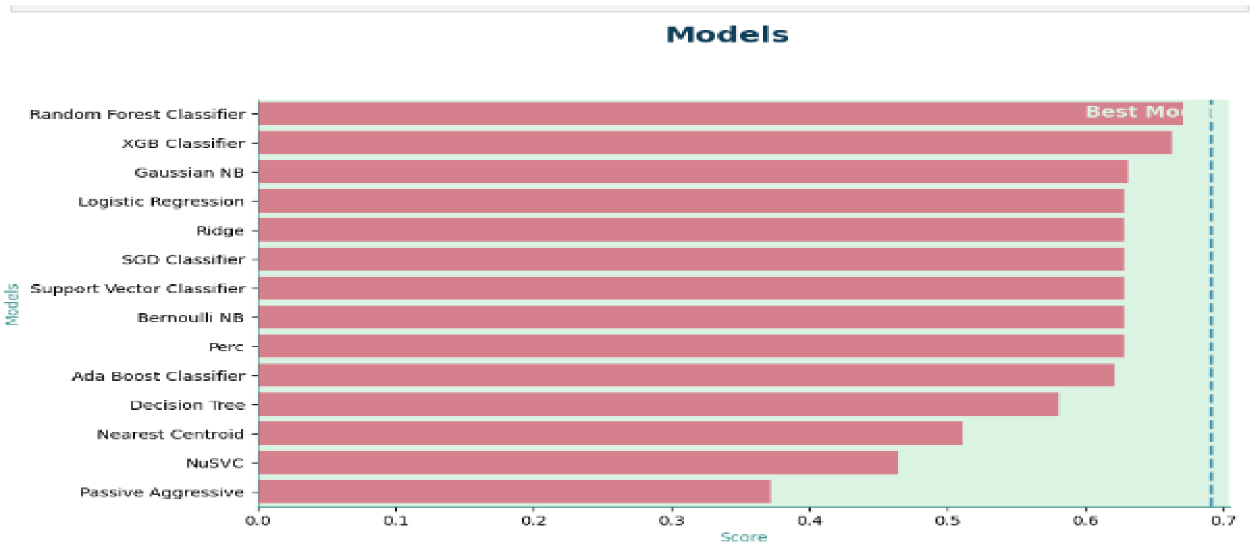
```
X_train (2620, 9)
X_test (656, 9)
y_train (2620,)
y_test (656,)
```

Now, by training on a couple models, I see that the random forest performs better, followed by the XGB Classifier.



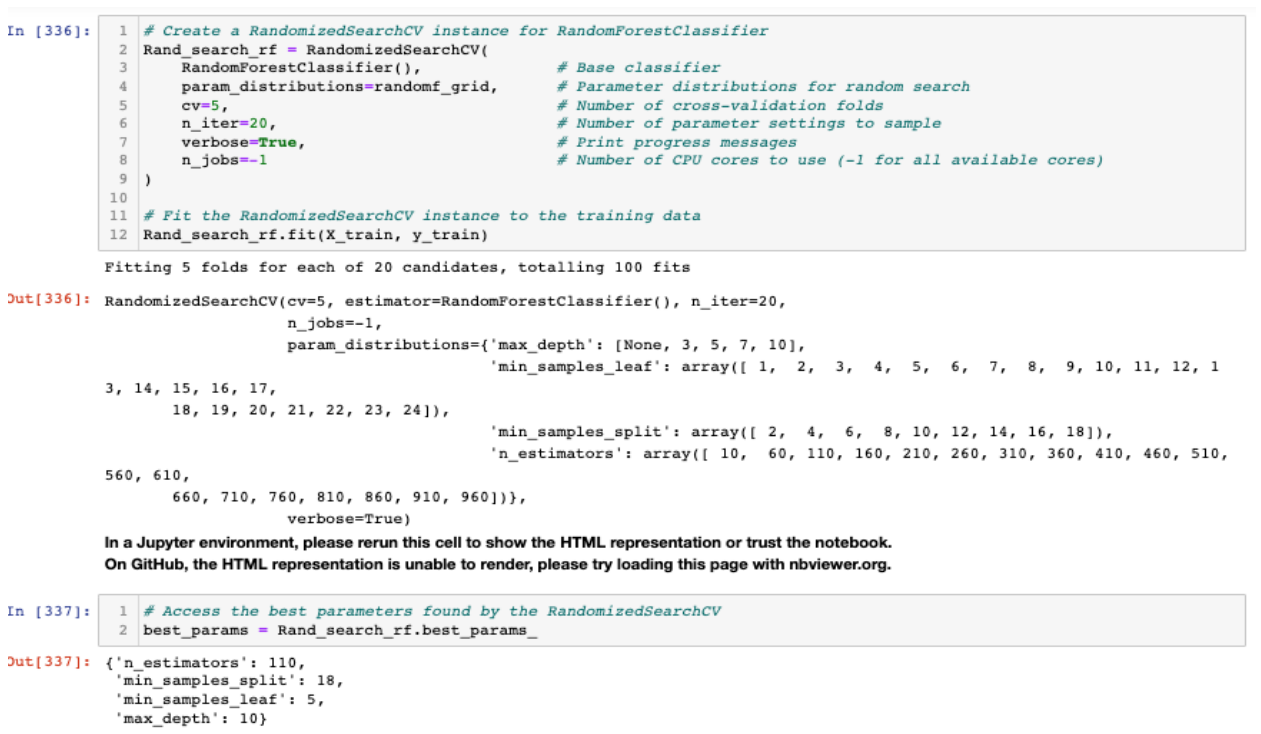
```
#finding the best models with base parameters
models = [
    ('Logistic Regression', LogisticRegression(max_iter=1000)),
    ('Ridge', RidgeClassifier()),
    ('SGD Classifier', SGDClassifier(max_iter=1000, tol=1e-3)),
    ('Support Vector Classifier', SVC()),
    ('NuSVC', NuSVC()),
    ('Decision Tree', DecisionTreeClassifier()),
    ('Gaussian NB', GaussianNB()),
    ('Bernoulli NB', BernoulliNB()),
    ('Perc', Perceptron()),
    ('Nearest Centroid', NearestCentroid()),
    ('Random Forest Classifier', RandomForestClassifier()),
    ('Ada Boost Classifier', AdaBoostClassifier()),
    ('XGB Classifier', XGBClassifier(verbosity = 0)),
    ('Passive Aggressive', PassiveAggressiveClassifier())
]

results = dict()
for name, model in models:
    model.fit(X_train, y_train)
    pred = model.predict(X_test)


    score = accuracy_score(y_test, pred)
    mae = mean_absolute_error(y_test, pred)
    mse = mean_squared_error(y_test, pred)
    rmse = np.sqrt(mean_squared_error(y_test, pred))
    precision = precision_score(y_test, pred)
    results[name] = score
```

Here is the result figure.



9. HYPERPARAMETER TUNING:

This is a crucial step in optimizing the performance of machine learning models. Here, we systematically search for the best combination of hyperparameters that lead to the highest model performance on a validation dataset. By using a random search, **RandomizedSearchCV**, I found the best performer is following:

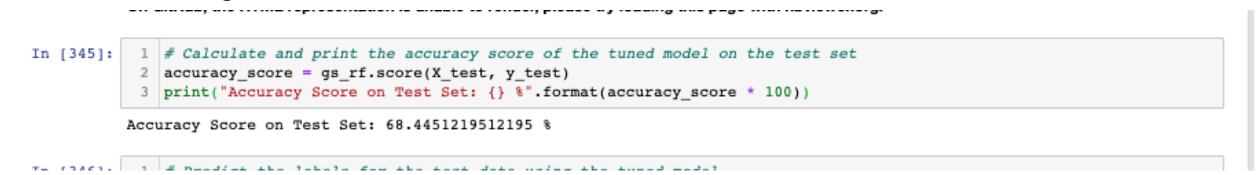


```
In [336]:
# Create a RandomizedSearchCV instance for RandomForestClassifier
Rand_search_rf = RandomizedSearchCV(
    RandomForestClassifier(),           # Base classifier
    param_distributions=randomf_grid,   # Parameter distributions for random search
    cv=5,                               # Number of cross-validation folds
    n_iter=20,                          # Number of parameter settings to sample
    verbose=True,                       # Print progress messages
    n_jobs=-1                           # Number of CPU cores to use (-1 for all available cores)
)

# Fit the RandomizedSearchCV instance to the training data
Rand_search_rf.fit(X_train, y_train)

Fitting 5 folds for each of 20 candidates, totalling 100 fits

Out[336]: RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_iter=20,
                   n_jobs=-1,
                   param_distributions={'max_depth': [None, 3, 5, 7, 10],
                                        'min_samples_leaf': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 1
3, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24]),
                                        'min_samples_split': array([ 2,  4,  6,  8, 10, 12, 14, 16, 18]),
                                        'n_estimators': array([ 10,  60, 110, 160, 210, 260, 310, 360, 410, 460, 510,
560, 610,
       660, 710, 760, 810, 860, 910, 960])},
                   verbose=True)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

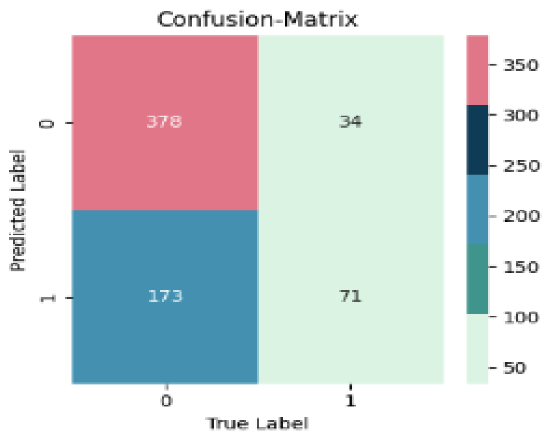
```
In [337]:
# Access the best parameters found by the RandomizedSearchCV
best_params = Rand_search_rf.best_params_

Out[337]: {'n_estimators': 110,
 'min_samples_split': 18,
 'min_samples_leaf': 5,
 'max_depth': 10}
```

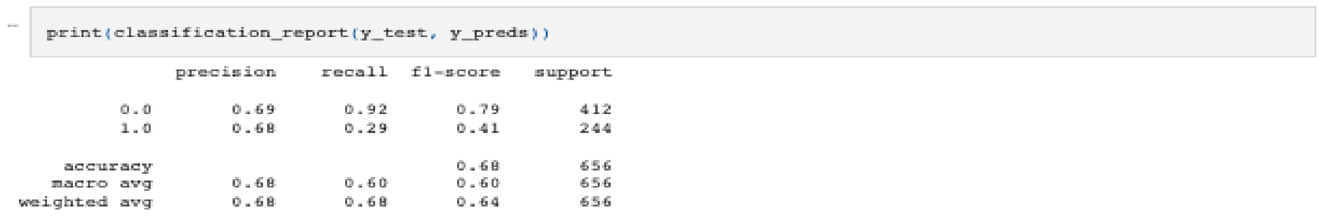
Also, by running a grid search on the **RandomForestClassifier,** and running the accuracy test on the test set, I found it working for 68.44%



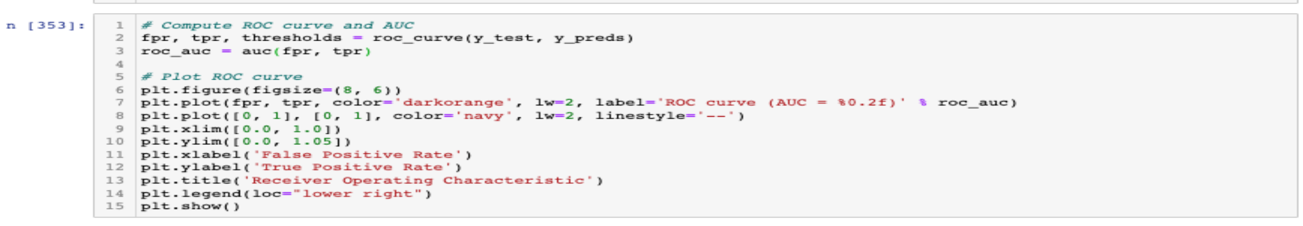
```
In [345]: # Calculate and print the accuracy score of the tuned model on the test set
          accuracy_score = gs_rf.score(X_test, y_test)
          print("Accuracy Score on Test Set: {} %".format(accuracy_score * 100))

Accuracy Score on Test Set: 68.4451219512195 %
```

And by running the confusion matrix, the accuracy is around 68%

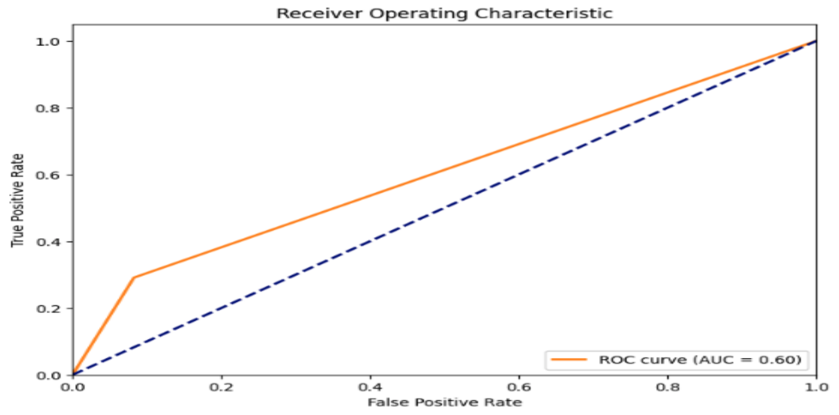


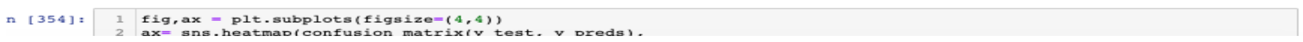


```
print(classification_report(y_test, y_preds))

              precision    recall  f1-score   support

         0.0       0.69      0.92      0.79       412
         1.0       0.68      0.29      0.41       244

    accuracy                           0.68       656
   macro avg       0.68      0.60      0.60       656
weighted avg       0.68      0.68      0.64       656
```



```
n [353]: # Compute ROC curve and AUC
         fpr, tpr, thresholds = roc_curve(y_test, y_preds)
         roc_auc = auc(fpr, tpr)

         # Plot ROC curve
         plt.figure(figsize=(8, 6))
         plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %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')
         plt.legend(loc="lower right")
         plt.show()
```





```
n [354]: fig,ax = plt.subplots(figsize=(4,4))
         ax= sns.heatmap(confusion_matrix(y_test, y_preds),
```

10. CONCLUSION:

We have explored the application of machine learning techniques to assess water quality based on its constituent elements. Among the constructed models, the random forest exhibited the most promising outcomes. However, an inherent imbalance between the quantities of safe and unsafe samples introduced bias towards the majority class (unsafe). Consequently, various remedies for this issue were investigated, with the utilization of sample weights proving to be the most effective strategy. Nevertheless, a more in-depth exploration of this predicament is warranted to cultivate a model that offers enhanced accuracy and dependability.

In summation, the analysis of the dataset reveals the following key observations:

- ❖ The data is predominantly clean, with minimal instances of inaccurate reports that were subsequently discarded.
- ❖ The investigation did not unveil concrete evidence suggesting the redundancy of particular features or the likelihood of generating biased solutions.
- ❖ The presence of a class imbalance adversely affects the performance of classifiers.
- ❖ The application of sample weights enhances classifier accuracy, although a comprehensive inquiry remains imperative for substantiated results.

REFERENCE:

- ❖ Water Quality
  Kadiwal
  https://www.kaggle.com/datasets/adityakadiwal/water-potability
- ❖ Google
  www.google.com