# Project Report

## *Water Quality Explanatory Data Analysis and ML Modelling*


**Submitted by**

**GOPAL SINGH**

PG DIPLOMA IN DATA SCIENCE AND TECHNOLOGY,
CCSD,
JC BOSE UNIVERSITY OF SCIENCE AND TECHNOLOGY,
FARIDABAD.

**Mentor**

**MR. RANJAN PRASAD**

CCSD,
JC BOSE UNIVERSITY OF SCIENCE AND TECHNOLOGY,
FARIDABAD.


A major Machine Learning project report is submitted in CCSD of JC BOSE University Of Science And Technology in partial fulfilment of the requirements for Post Graduation Diploma in Data Science And Analytics.

Date:

Place:

Approved by

Professor Name

# *Water Quality Explanatory Data Analysis and ML Modelling*

**By Gopal Singh**

Accepted in partial fulfillment of the requirements for the Degree of Post Graduation Diploma in Data Science and Analytics

| | |
|---|---|
| _____ | _____ |
| Professor name | External Examiner |
| Date: | Date: |

# Contents

# 1. Abstract

Building scalable machine learning as a service, or MLaaS, is critical to enterprise success. Key to translate machine learning project success into program success is to solve the evolving convoluted data engineering challenge, using local and global data. Enabling sharing of data features across a multitude of models within and across various line of business is pivotal to program success.

A feature is basically any input into an ML model. It is a set of variables that are incorporated into an ML model with the intention to improve model performance and accuracy. Features are derived values extracted from files and tables (a database) — and more importantly, computed from one or more tables. These are usually grouped together to minimize operational overhead and optimize storage. A feature, for example, can be any column with a calculated, flagged or one hot encoded value

A feature store is a central repository for storing documented, curated, and access-controlled features. It is a central place to store features that are properties of data, be it in the form of statistical derivations, piece of text, image pixel coordinates, aggregated value of purchase history, etc. This enables feature management to be uniform, reliable, reusable and governed. A feature store shouldn't be perceived as a type of new data store. In fact it should be recognized as a store of feature recipes with occasional time dependencies. For example, a feature like "number of login attempts in the last hour" is used in fraud models, customer service models and retention models. Each model will be computing it at a different historical point — the fraud model perhaps during a logon attempt, the customer service call at the point someone calls a call center and the retention model a certain date for the model to be built. But when the feature goes into production, it needs to run every single time a customer calls the call center

A feature store enables reusability of features across the enterprise, as existing features are visible to all potential users (e.g., business analysts, business intelligence developers, data scientists, etc.) across the business domain. The feature store supports feature enrichment, ranking, discovery, lineage (both data to feature and feature to model) and lifecycle management.

Both development and model serving teams need a diverse feature set, which can be met easily through the store. This will enable both teams to discover, store and manage features, while also decommissioning features that are no longer needed.

# 2. Acknowledgements

This work would not have been possible without the support of many people. I wish to express my sincere gratitude to all those who gave me the possibility to complete this thesis.

First of all, I would like to thank my supervisor, Mr Ranjan Prasad YMCA UST,Faridabad. for helping me realize this project and for his guidance during this work and giving me the opportunity to work in one of well-equipped lab of the institution. His valuable ideas and suggestions were immensely helpful which make this thesis work possible.

I would also like to thank Mr. Rajinder Chitoria (Data Scientist) and Co-Founder of Froyo ,Faridabad for granting me opportunity for Training at Froyo technology by Antrix Academy at Faridabad.

And Most importantly, I want to thank my family for providing unlimited support and helping me realize my dreams.

# 3. INTRODUCTION

- Access to safe drinking-water is essential to health, a basic human right and a component of effective policy for health protection. This is important as a health and development issue at a national, regional and local level. In some regions, it has been shown that investments in water supply and sanitation can yield a net economic benefit, since the reductions in adverse health effects and health care costs outweigh the costs of undertaking the interventions.
- Drinking water and staying hydrated is associated with a reduced incidence of urinary tract infections (UTIs), lower blood pressure and heart disease. Therefore, drinking water is essential for good heart health.
- Water is the most important nutrient for the body. It has many benefits for your health and helps to protect you from illness and disease. Water is also an essential part of a healthy lifestyle.

# 4. What will you learn from this project

- Bivariate and multivariate data analysis
- Correlation analysis
- Preprocessing: missing value, train-test split and normalization
- Modelling: Decision Tree and Random Forest Classifiers
- Visualize Decision Tree
- Random Forest Hyperparameter Tuning

# 5. Python Libraries

```
# This Python 3 environment comes with many helpful analytics libraries installed

# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python

# For example, here's several helpful packages to load


import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import seaborn as sns

import matplotlib.pyplot as plt

import plotly.express as px

import missingno as msno

# Input data files are available in the read-only "../input/" directory
```

```
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session

# ML
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV, RepeatedStratifiedKFold, train_test_split
from sklearn.metrics import precision_score, confusion_matrix

from sklearn import tree
```

# 6. Data Content

- **pH value:** PH 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. WHO has recommended maximum permissible limit of pH from 6.5 to 8.5. The current investigation ranges were 6.52–6.83 which are in the range of WHO standards.
- **Hardness:** Hardness is 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 (Total dissolved solids - TDS):** 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 un-wanted taste and diluted color in appearance of water. This is the important parameter for the use of water. The water with high TDS value indicates that water is highly mineralized. Desirable limit for TDS is 500 mg/l and maximum limit is 1000 mg/l which prescribed for drinking purpose.

- **Chloramines:** 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. Chlorine levels up to 4 milligrams per liter (mg/L or 4 parts per million (ppm)) are considered safe in 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. Sulfate concentration in seawater is about 2,700 milligrams per liter (mg/L). It ranges from 3 to 30 mg/L in most freshwater supplies, although much higher concentrations (1000 mg/L) are found in some geographic locations.
- **Conductivity:** Pure water is not a good conductor of electric current rather'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. Electrical conductivity (EC) actually measures the ionic process of a solution that enables it to transmit current. According to WHO standards, EC value should not exceeded 400 μS/cm.
- **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. According to US EPA < 2 mg/L as TOC in treated / drinking water, and < 4 mg/Lit in source water which is use for treatment.
- **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. THM levels up to 80 ppm is considered safe in drinking water.
- **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. The mean turbidity value obtained for Wondo Genet Campus (0.98 NTU) is lower than the WHO recommended value of 5.00 NTU.
- **Potability:** Indicates if water is safe for human consumption where 1 means Potable and 0 means Not potable.

# 7. Read And Analysis of Data

In [6]:
```
df = pd.read_csv("water_potability.csv")
```

In [7]:
```
df.head()
```

Out[7]:

| | 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 |

In [8]:
```
# describe
df.describe()
```

Out[8]:

| | 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 |

# 8. Dependent Variable Analysis

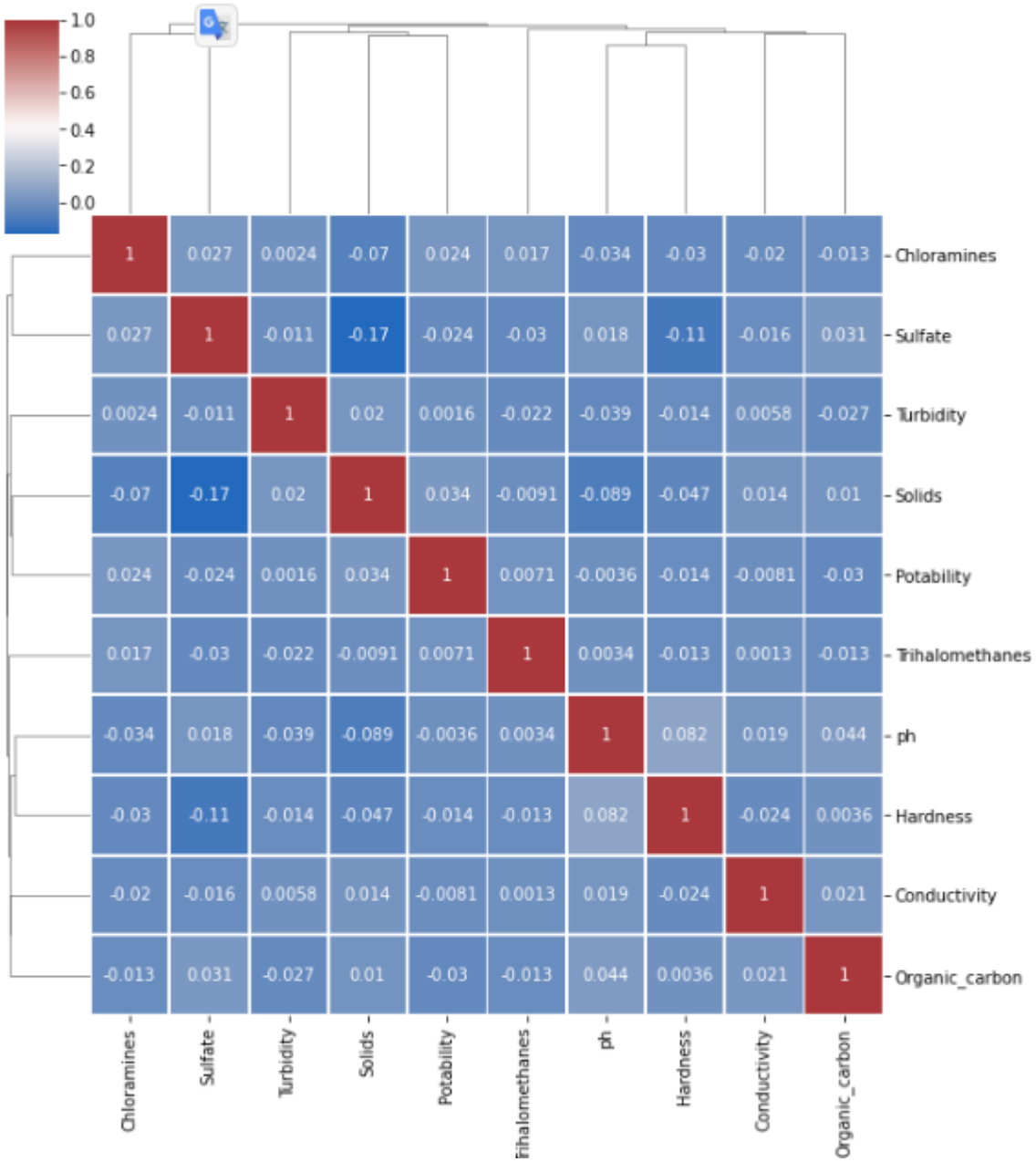
```
d = pd.DataFrame(df["Potability"].value_counts())
fig = px.pie(d, values = "Potability", names = ["Not Potable", "Potable"], hole = 0.35, opacity = 0.8,
             labels = {"label" :"Potability","Potability":"Number of Samples"})
fig.update_layout(title = dict(text = "Pie Chart of Potability Feature"))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

# 9. Correlation Between Features

```
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 |

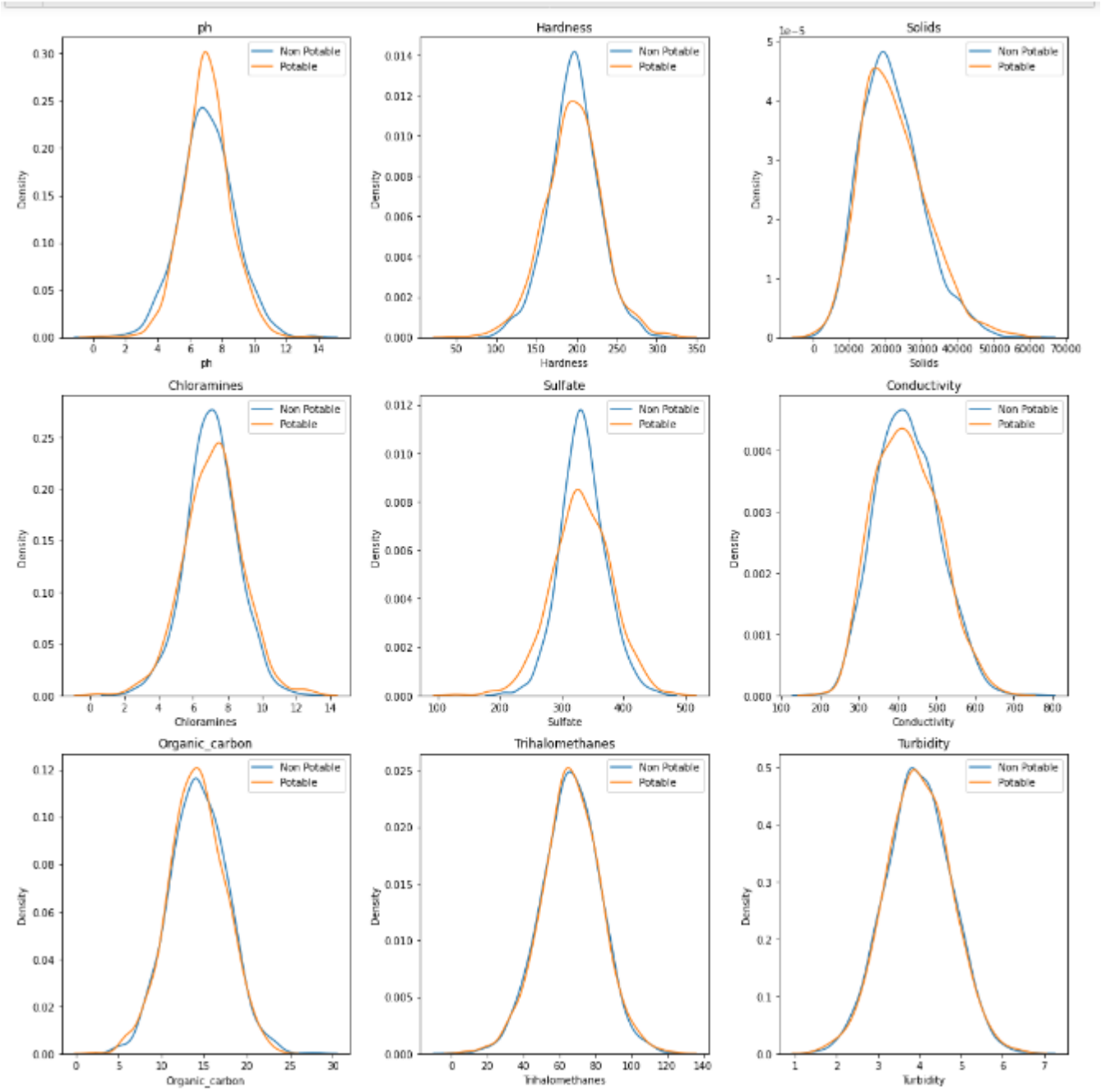
```
sns.clustermap(df.corr(), cmap = "vlag", dendrogram_ratio = (0.1, 0.2), annot = True, linewidths = .8, figsize = (9,10))
plt.show()
```




# 10. Distribution of Features

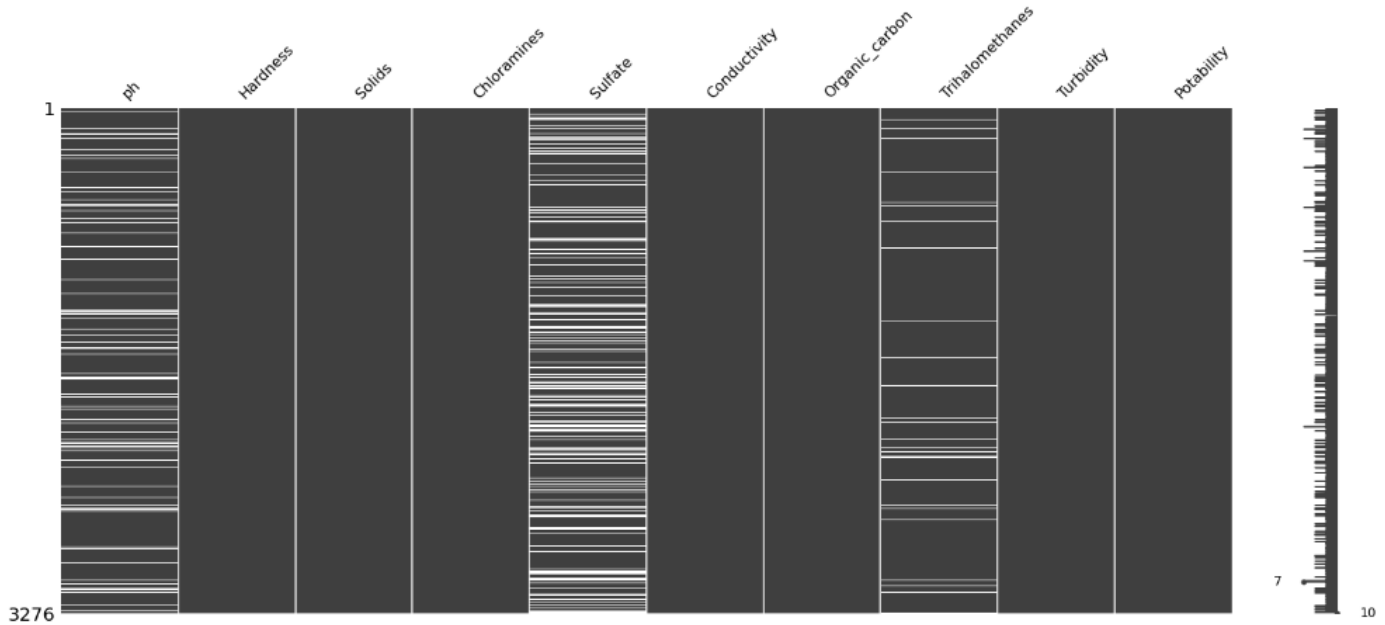
```
non_potable = df.query("Potability == 0")
potable = df.query("Potability == 1")

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()
```




# 11. Processing Missing Values Problem

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




```
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
```

```
# handle missing value with average of features
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)
```

# 12. Processing Train_Test_Split and Normalization

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

```
# train test split
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,)
```

```
# min-max normalization
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)
```

# 13. Modelling Decision Tree and Random Forest Classifiers

Precision Score: The precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.

```
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
```

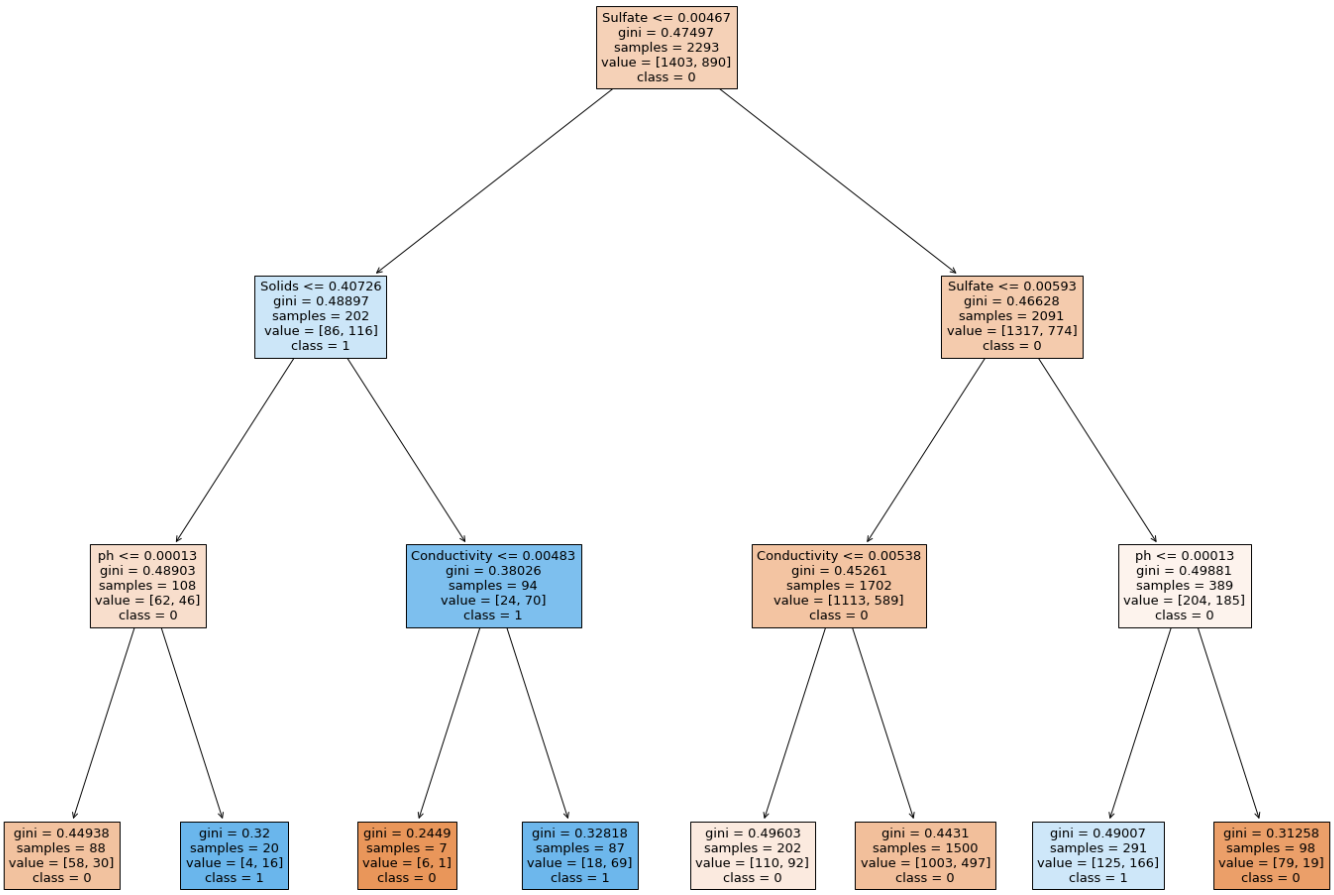
```
[('DTC', 0.5652173913043478), ('RF', 0.6346153846153846)]
```

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

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

```
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()
```




# 14. Random Forest HyperParameter Tuning

```
model_params = {
    "Random Forest":
    {
        "model":RandomForestClassifier(),
        "params":
        {
            "n_estimators":[10, 50, 100],
            "max_features":["auto","sqrt","log2"],
            "max_depth":list(range(1,21,3))
        }
    }

}
model_params
```

```
{'Random Forest': {'model': RandomForestClassifier(),
  'params': {'n_estimators': [10, 50, 100],
   'max_features': ['auto', 'sqrt', 'log2'],
   'max_depth': [1, 4, 7, 10, 13, 16, 19]}}}
```

```
cv = RepeatedStratifiedKFold(n_splits = 5, n_repeats = 2)
scores = []
for model_name, params in model_params.items():
    rs = RandomizedSearchCV(params["model"], params["params"], cv = cv, n_iter = 10)
    rs.fit(X,y)
    scores.append([model_name, dict(rs.best_params_),rs.best_score_])
scores
```

```
[['Random Forest',
  {'n_estimators': 50, 'max_features': 'log2', 'max_depth': 13},
  0.667434369763545]]
```