Water Quality Analysis

Drinking water potability

Context

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.

Content

The water_potability.csv file contains water quality metrics for 3276 different water bodies.

1. 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.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.

7. 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.

8. 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.

9. 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.

10. Potability: Indicates if water is safe for human consumption where 1 means Potable and 0 means Not potable.

STEPS COVERED :

FEATURE ENGINEERING - Treating the Missing Values

1)Random Sample Imputation
2)For Handling Categorical missing values like Cabin

3)Frequent Category Imputation
4)Label Encoding

Normalization And Standardisation

Robust Scaler

DISTRIBUTION OF THE COLUMNS

Guassian Transformation

boxcox transformation - For coverting Skewed Cloumns to Normal Distribution

FEATURE SELECTION Univariate Selection

ExtraTreesClassifier/Kbest Selector

Importance Of The Features Wrt, Label/Target Variable

Correlation - To Check Multicollinearity

Treating the multicollinearity with Threshold values

Information Gain

Checking for Outliers

PIPELINE CREATION

Hyper Parameter Tuning For Logistic Regression

Randomized Search Cross Validation

GridSearch CV

==

Automated Hyperparameter Tuning

==

Bayesian Optimization

Genetic Algorithms - TPOT Classifier

==

Optimize hyperparameters of the model using Optuna

In [2]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [3]:
```
# Suppressing Warnings
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
# Importing Pandas and NumPy
import pandas as pd, numpy as np
```

In [5]:
```
# Importing all datasets
water_portability = pd.read_csv("C:/Users/HP/Desktop/Upgrad Case Study/Water Quality/water_potabil
water_portability.head(4)
```

Out[5]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Tur
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.9
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180013	56.329076	4.5
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.868637	66.420093	3.0
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.341674	4.6

In [6]:

```
water_portability.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               2785 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3114 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

In [7]:

```
water_portability.shape
```

Out[7]:
```
(3276, 10)
```

FEATURE ENGINEERING

Treating the `Missing` Values

Data That can be missing can be of two types :

1) Continuous Data

2) Discreate Or Categorical Data

The Types of `missing` can be of mentioned types :

1) **MCAR** - Missing Completely At Random

If the probability of being `missing` is same for all the observations.

2) **MNAR** - Missing Not At Random

There is some relationship between the missing data

3) **MAR** - Missing At Random

Random Sample Imputation

Aim: Random sample imputation consists of taking random observation from the dataset and we use this observation to replace the nan values

When should it be used? It assumes that the data are missing completely at `random(MCAR)`

In [8]:

```
water_portability.isnull().sum()
```

Out[8]:
```
ph                 491
Hardness             0
Solids               0
Chloramines          0
Sulfate            781
Conductivity         0
Organic_carbon       0
Trihalomethanes    162
Turbidity            0
Potability           0
dtype: int64
```

In [9]:
```
def impute_nan(df,variable,median):
    df[variable+"_median"]=df[variable].fillna(median)
    df[variable+"_random"]=df[variable]
    ##It will have the random sample to fill the na
    random_sample=df[variable].dropna().sample(df[variable].isnull().sum(),random_state=0)
    ##pandas need to have same index in order to merge the dataset
    random_sample.index=df[df[variable].isnull()].index
    df.loc[df[variable].isnull(),variable+'_random']=random_sample
```

In [10]:
```
median=water_portability.ph.median()
```

In [11]:
```
median
```

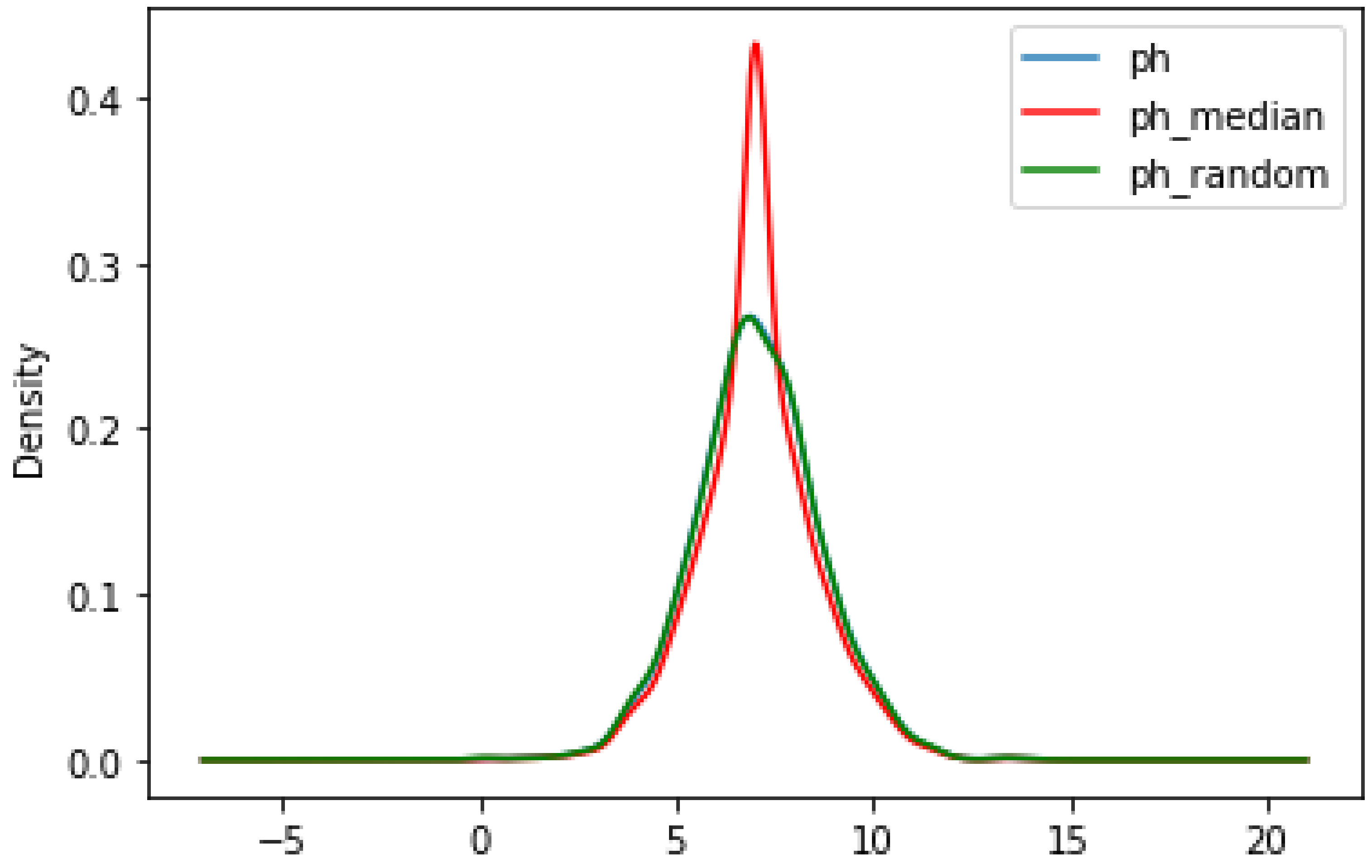
Out[11]:
```
7.036752103833548
```

In [12]:
```
impute_nan(water_portability,"ph",median)
```

In [13]:
```

import matplotlib.pyplot as plt
%matplotlib inline
```

In [14]:
```

fig = plt.figure()
ax = fig.add_subplot(111)

water_portability.ph.plot(kind='kde', ax=ax)
water_portability.ph_median.plot(kind='kde', ax=ax, color='red')
water_portability.ph_random.plot(kind='kde', ax=ax, color='green')
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
```

Out[14]:
```
<matplotlib.legend.Legend at 0x2cf936992b0>
```



From the Above `Observation`,

In [15]:
```
water_portability = water_portability.drop(columns=["ph","ph_median"])
```

In [16]:
```
water_portability = water_portability.rename(columns={"ph_random": "ph"})
```

In [17]:
```
median=water_portability.Sulfate.median()
```

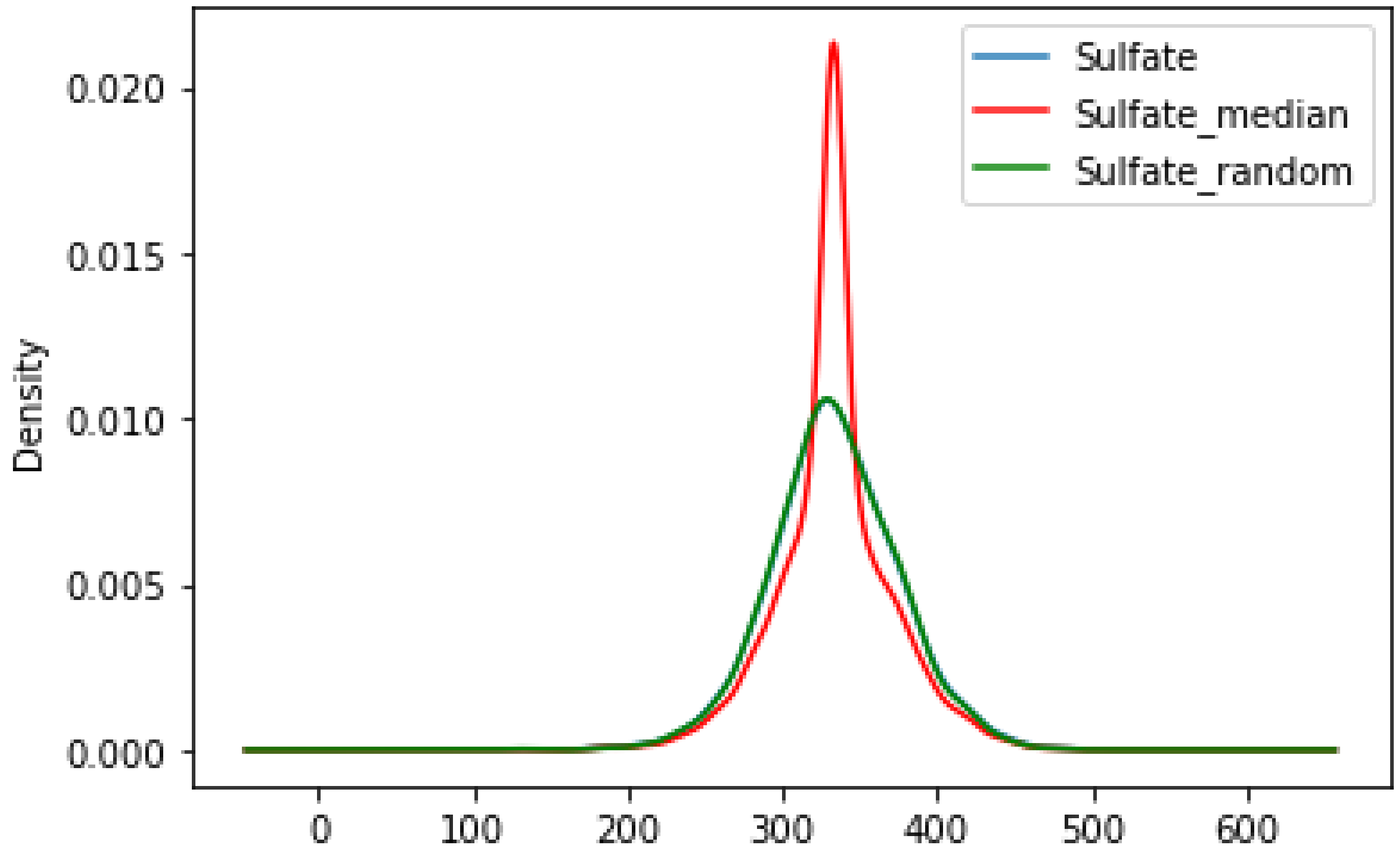
In [18]:
```
median
```

Out[18]:
```
333.073545745888
```

```
In [19]: impute_nan(water_portability,"Sulfate",median)
```

```
In [20]: fig = plt.figure()
         ax = fig.add_subplot(111)

         water_portability.Sulfate.plot(kind='kde', ax=ax)
         water_portability.Sulfate_median.plot(kind='kde', ax=ax, color='red')
         water_portability.Sulfate_random.plot(kind='kde', ax=ax, color='green')
         lines, labels = ax.get_legend_handles_labels()
         ax.legend(lines, labels, loc='best')
```

```
Out[20]: <matplotlib.legend.Legend at 0x2cf93791130>
```



From the Above `Observation`, for "Sulfate"

In [21]:
```
water_portability = water_portability.drop(columns=["Sulfate","Sulfate_median"])
```

In [22]:
```
water_portability = water_portability.rename(columns={"Sulfate_random": "Sulfate"})
```

In [23]:
```
median=water_portability.Trihalomethanes.median()
```

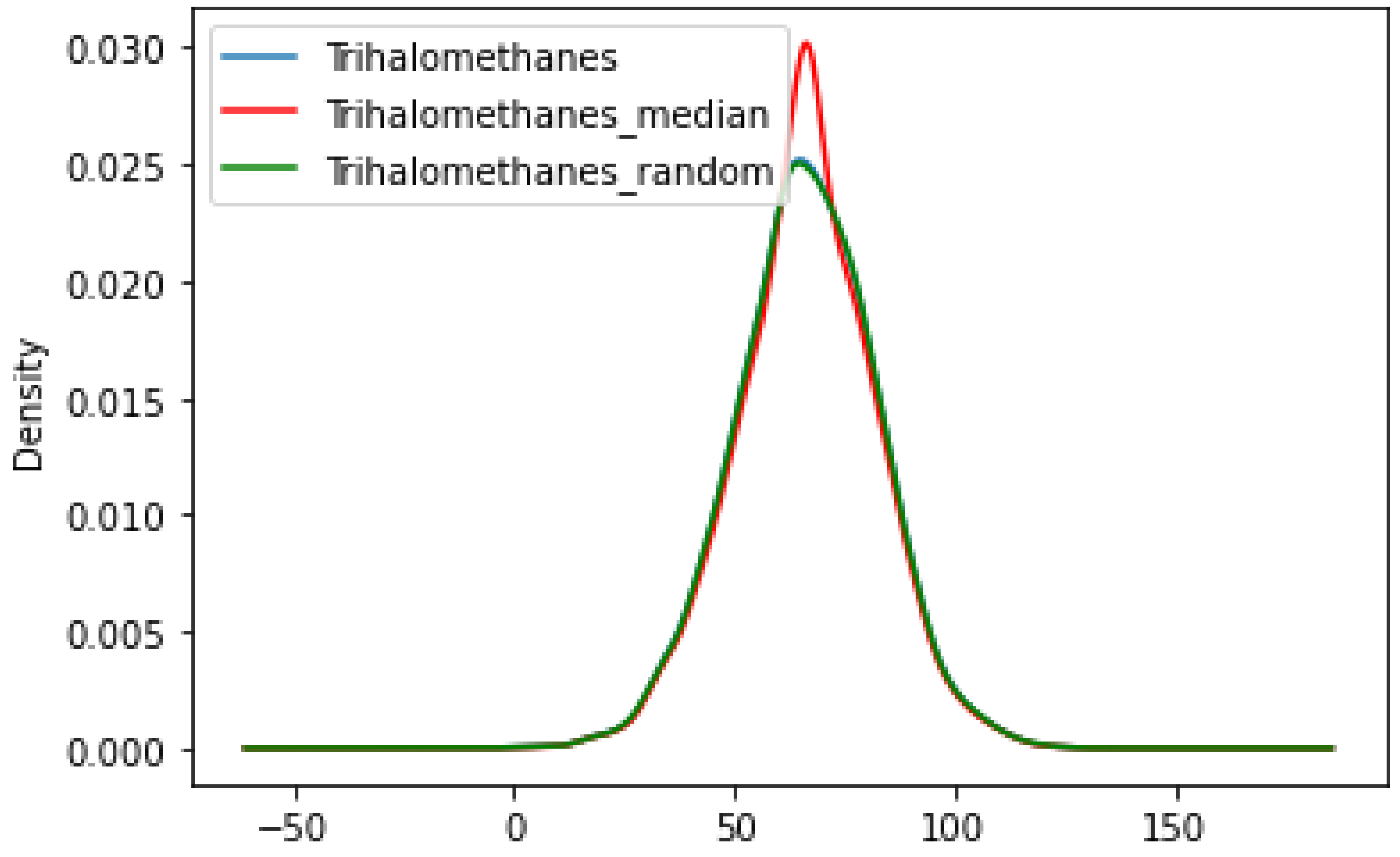
In [24]:
```
median
```

Out[24]:
```
66.62248509808484
```

In [25]:
```
impute_nan(water_portability,"Trihalomethanes",median)
```

In [26]:
```
fig = plt.figure()
ax = fig.add_subplot(111)

water_portability.Trihalomethanes.plot(kind='kde', ax=ax)
water_portability.Trihalomethanes_median.plot(kind='kde', ax=ax, color='red')
water_portability.Trihalomethanes_random.plot(kind='kde', ax=ax, color='green')
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
```

Out[26]:
```
<matplotlib.legend.Legend at 0x2cf93816220>
```



From the Above `Observation`, for "Trihalomethanes":

```
In [27]:  water_portability = water_portability.drop(columns=["Trihalomethanes","Trihalomethanes_median"])
```

```
In [28]:  water_portability = water_portability.rename(columns={"Trihalomethanes_random": "Trihalomethanes"}
```

```
In [29]:  water_portability.isnull().sum()
```

```
Out[29]: Hardness           0
         Solids             0
         Chloramines        0
         Conductivity       0
         Organic_carbon     0
```

```
Turbidity          0
Potability         0
ph                 0
Sulfate            0
Trihalomethanes    0
dtype: int64
```

In [30]:
```
water_portability.head(3)
```

Out[30]:

	Hardness	Solids	Chloramines	Conductivity	Organic_carbon	Turbidity	Potability	ph	Sulfate
0	204.890455	20791.318981	7.300212	564.308654	10.379783	2.963135	0	9.074923	368.516441
1	129.422921	18630.057858	6.635246	592.885359	15.180013	4.500656	0	3.716080	298.082462
2	224.236259	19909.541732	9.275884	418.606213	16.868637	3.055934	0	8.099124	367.224297

In [31]:
```
water_portability.Potability.value_counts()
```

Out[31]:
```
0    1998
1    1278
Name: Potability, dtype: int64
```

Label Encoding

- Since, All the values are `numerical` in nature, So actually `No Label Encoding` is Required.

Normalization And Standardisation

Robust Scaler

It is used to scale the feature to median and quantiles Scaling using median and quantiles consists of substracting the median to all the observations, and then dividing by the interquantile difference. The interquantile difference is the difference between the 75th and 25th quantile:

IQR = 75th quantile - 25th quantile

X_scaled = (X - X.median) / IQR

0,1,2,3,4,5,6,7,8,9,10

9-90 percentile ---90% of all values in this group is less than 9
1-10 precentile ---10% of all values in this group is less than 1
4-40%

In [32]:
```
from sklearn.preprocessing import RobustScaler
scaler=RobustScaler()
water_portability_robust_scaler=pd.DataFrame(scaler.fit_transform(water_portability),columns=water
water_portability_robust_scaler.head()
```

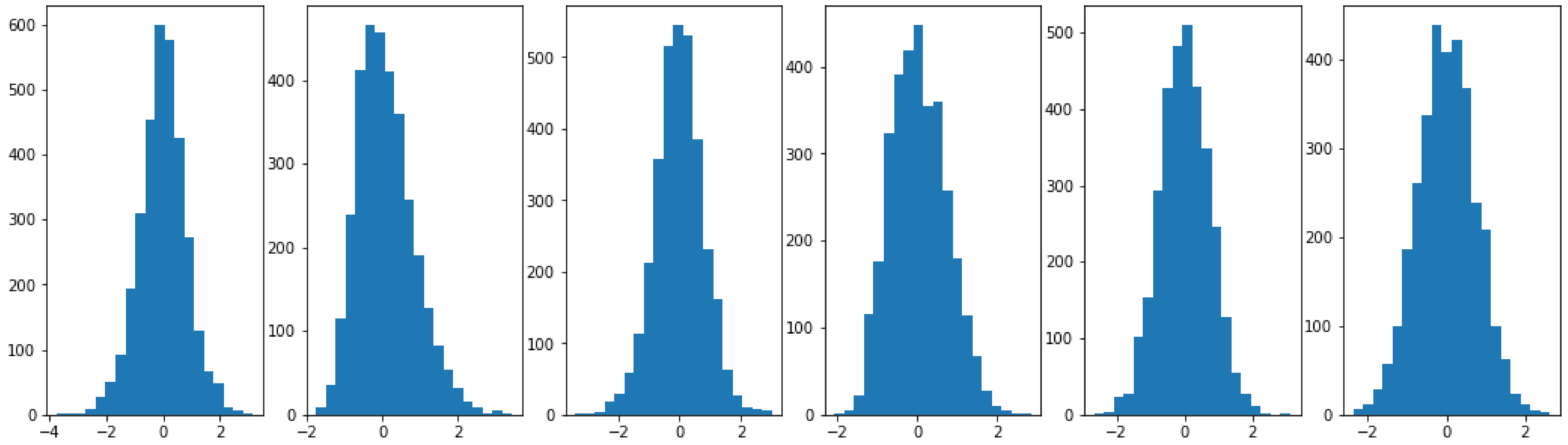
Out[32]:

	Hardness	Solids	Chloramines	Conductivity	Organic_carbon	Turbidity	Potability	ph	Sulfate	Triha
0	0.198981	-0.011702	0.085492	1.227178	-0.854560	-0.935210	0.0	1.032167	0.682864	
1	-1.696382	-0.196962	-0.249088	1.473406	0.214093	0.514449	0.0	-1.672015	-0.666773	
2	0.684850	-0.087287	1.079558	-0.028251	0.590024	-0.847715	0.0	0.539759	0.658104	

	Hardness	Solids	Chloramines	Conductivity	Organic_carbon	Turbidity	Potability	ph	Sulfate	Triha
3	0.437145	0.093483	0.467446	-0.505079	0.939076	0.635242	0.0	0.649586	0.460007	
4	-0.398477	-0.252771	-0.293690	-0.202262	-0.592197	0.113188	0.0	1.040897	-0.435811	

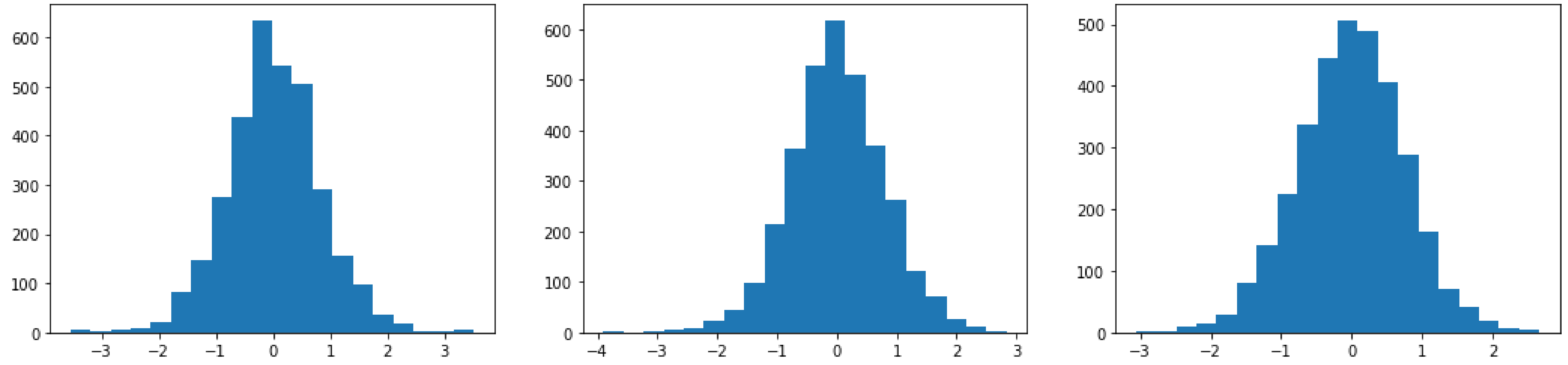
DISTRIBUTION OF THE `COLUMNS`

In [33]:
```
plt.figure(figsize=(18,5))
plt.subplot(1,6,1)
plt.hist(water_portability_robust_scaler['Hardness'],bins=20)
plt.subplot(1,6,2)
plt.hist(water_portability_robust_scaler['Solids'],bins=20)
plt.subplot(1,6,3)
plt.hist(water_portability_robust_scaler['Chloramines'],bins=20)
plt.subplot(1,6,4)
plt.hist(water_portability_robust_scaler['Conductivity'],bins=20)
plt.subplot(1,6,5)
plt.hist(water_portability_robust_scaler['Organic_carbon'],bins=20)
plt.subplot(1,6,6)
plt.hist(water_portability_robust_scaler['Turbidity'],bins=20)
plt.show()
```



```
In [34]:
plt.figure(figsize=(18,4))
plt.subplot(1,3,1)
plt.hist(water_portability_robust_scaler['ph'],bins=20)
plt.subplot(1,3,2)
plt.hist(water_portability_robust_scaler['Sulfate'],bins=20)
plt.subplot(1,3,3)
plt.hist(water_portability_robust_scaler['Trihalomethanes'],bins=20)
```

```
Out[34]: (array([  1.,   2.,  10.,  13.,  29.,  80., 142., 225., 338., 444., 507.,
        490., 405., 288., 163.,  70.,  40.,  18.,   7.,   4.]),
 array([-3.06196299, -2.77560672, -2.48925045, -2.20289418, -1.91653791,
        -1.63018164, -1.34382537, -1.0574691 , -0.77111283, -0.48475656,
        -0.19840029,  0.08795598,  0.37431225,  0.66066852,  0.94702479,
         1.23338106,  1.51973733,  1.8060936 ,  2.09244987,  2.37880614,
         2.66516241]),
 <BarContainer object of 20 artists>)
```



Guassian Transformation

Some machine learning algorithms like linear and logistic assume that the features are normally distributed - Accuracy -Performance

- logarithmic transformation
- reciprocal transformation
- square root transformation
- exponential transformation (more general, you can use any exponent)
- boxcox transformation

- Since , The Data are `Normally Distributed` , thus , Guassian Transformation is actually Not Required.

In [35]:
```
# choosing all the numerical variables as independent variables (classifier can only take numerica
# dropping two variable funded_amnt as we have created new variable transformation based on it
X = water_portability_robust_scaler.drop(columns = "Potability")
Y = water_portability_robust_scaler["Potability"]
```

```
#splitting the dataset in train and test datasets using a split ratio of 70:30

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=10)
```

In [36]:
```
X_train.shape
```

Out[36]:
```
(2293, 9)
```

In [37]:
```
X_test.shape
```

Out[37]:
```
(983, 9)
```

In [38]:
```
y_train.shape
```

Out[38]:
```
(2293,)
```

In [39]:
```
y_test.shape
```

Out[39]:
```
(983,)
```

Data Analysis - Univariate And Bivariate Analysis

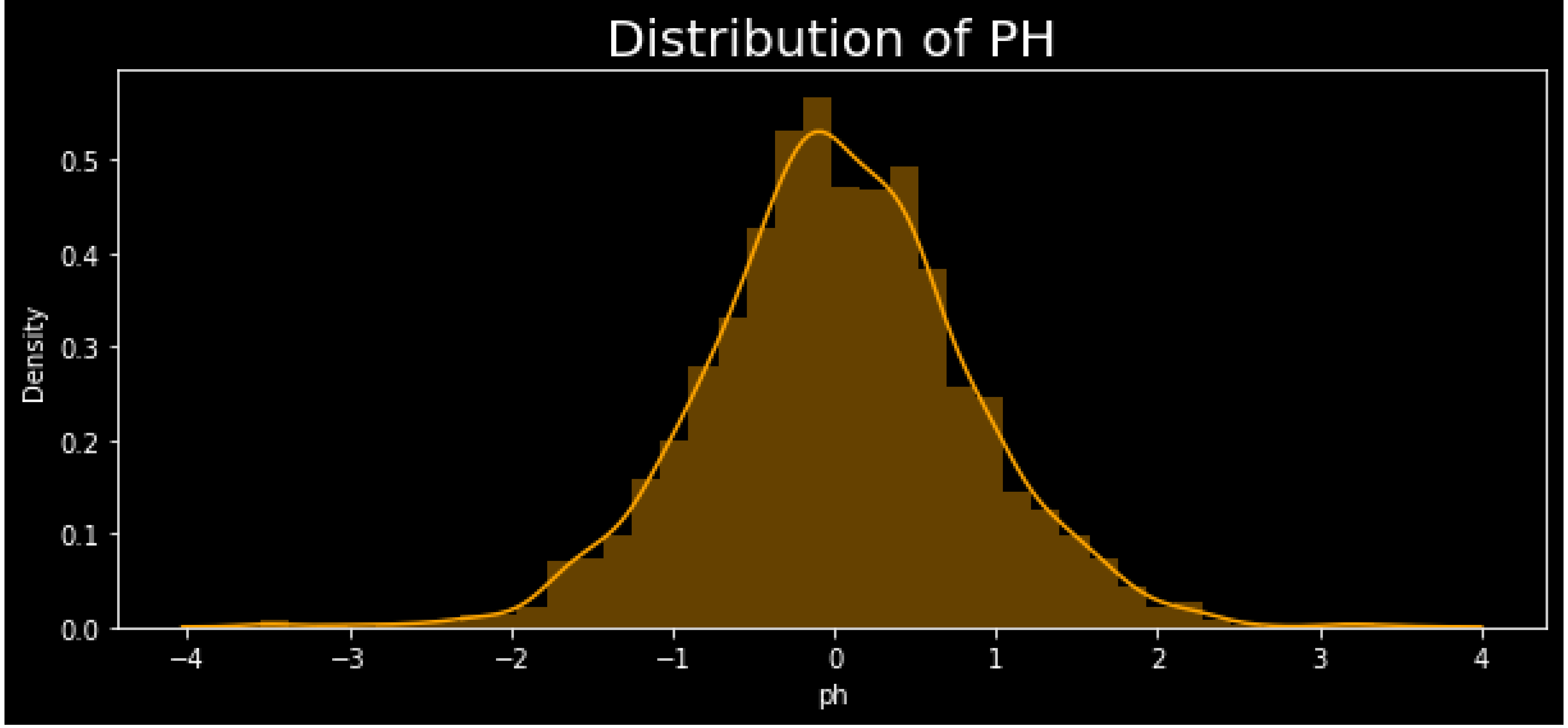
Univariate Analysis

In [40]:

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use("dark_background")
```

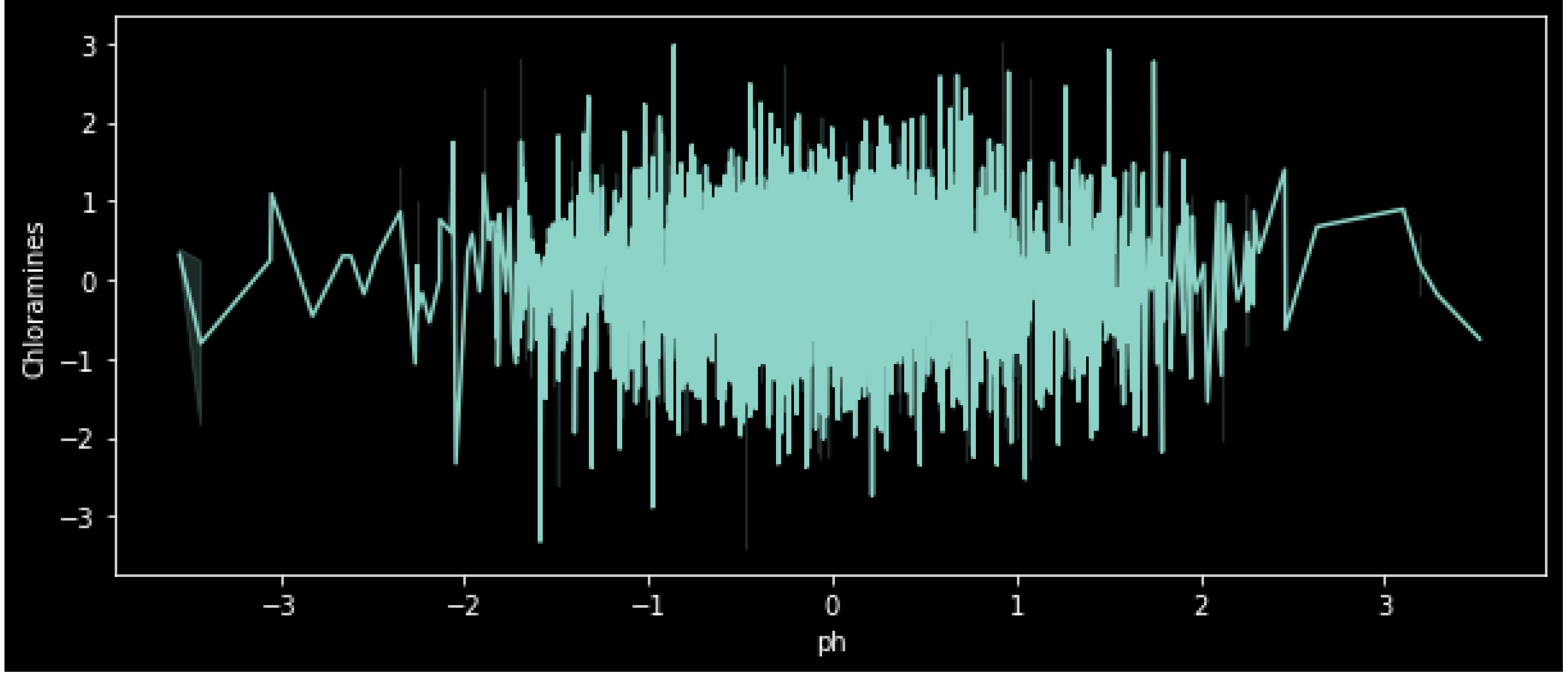
In [41]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

import seaborn as sns
plt.figure(figsize = [10,4])
sns.distplot(water_portability_robust_scaler.ph,  bins = 40, color = "orange")
plt.title("Distribution of PH", fontsize = 20, fontweight = 10, verticalalignment = 'baseline')

plt.show()
```



```
In [42]:
plt.figure(figsize = [10,4])
sns.lineplot(water_portability_robust_scaler.ph,water_portability_robust_scaler.Chloramines)
```

```
Out[42]: <AxesSubplot:xlabel='ph', ylabel='Chloramines'>
```



FEATURE SELECTION

Univariate Selection

In [43]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

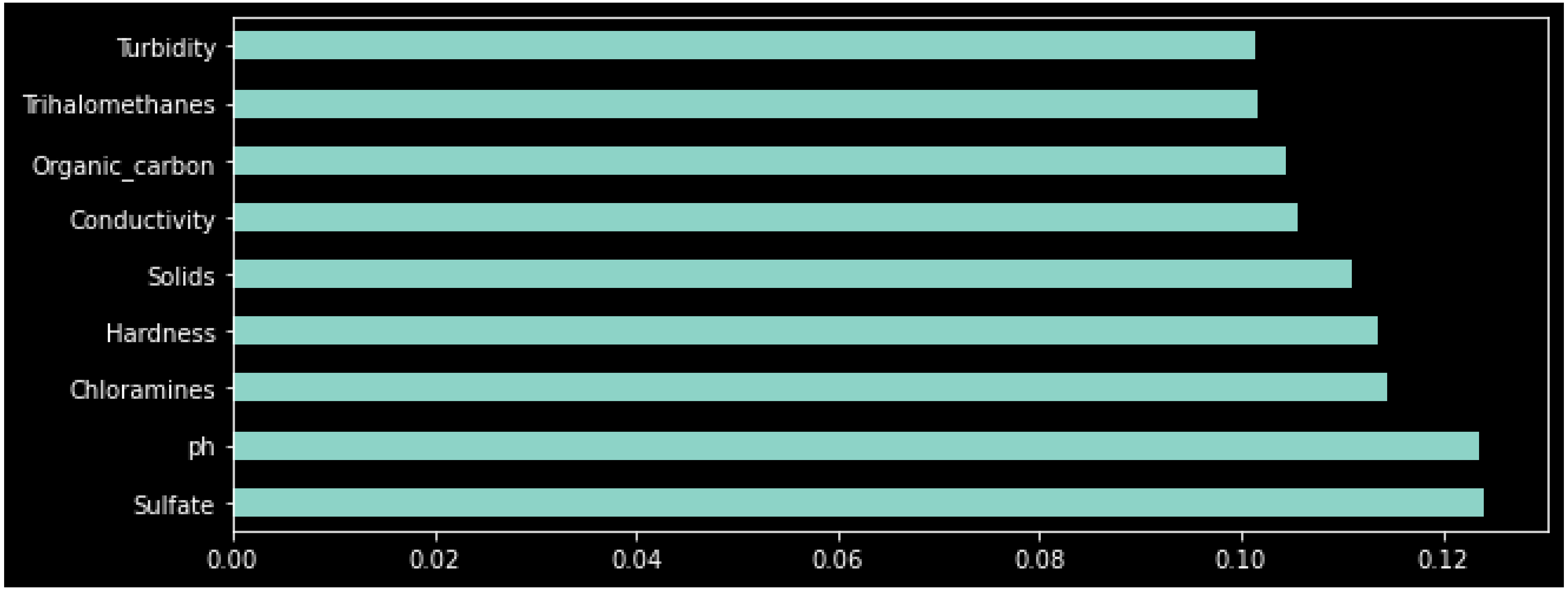
In [44]:
```
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt
model=ExtraTreesClassifier()
model.fit(X_train,y_train)
```

```
Out[44]: ExtraTreesClassifier()
```

```
In [45]: print(model.feature_importances_)
```

```
[0.11359884 0.11104449 0.11445582 0.10561082 0.10437789 0.1015234
 0.12366303 0.12416452 0.10156119]
```

```
In [46]: plt.figure(figsize = [10,4])
         ranked_features=pd.Series(model.feature_importances_,index=X_train.columns)
         ranked_features.nlargest(10).plot(kind='barh')
         plt.show()
```



Importance Of The Features Wrt, `Label/Target Variable`

In [47]:
```
ranked_features.nlargest(10, keep='all')
```

Out[47]:
```
Sulfate            0.124165
ph                 0.123663
Chloramines        0.114456
Hardness           0.113599
Solids             0.111044
Conductivity       0.105611
Organic_carbon     0.104378
Trihalomethanes    0.101561
Turbidity          0.101523
dtype: float64
```

- Here , From the above observation , all the features are important wrt, the `target` , So we can not drop `any`

Correlation - To Check `Multicollinearity`

In [48]:
```
X_train.corr()
```

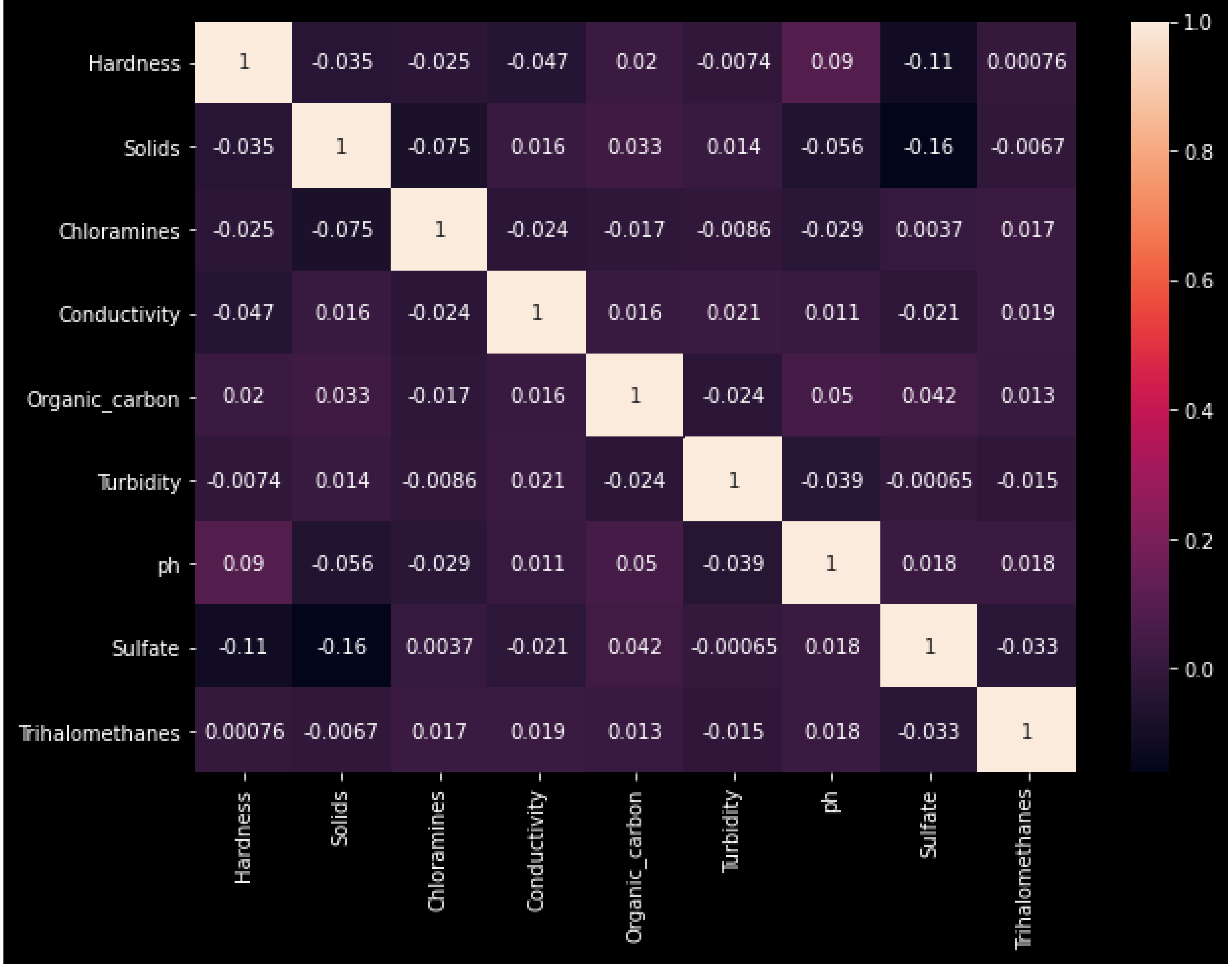
Out[48]:

	Hardness	Solids	Chloramines	Conductivity	Organic_carbon	Turbidity	ph	Sulfate	T
Hardness	1.000000	-0.034806	-0.025428	-0.046913	0.019890	-0.007436	0.089940	-0.106678	
Solids	-0.034806	1.000000	-0.075316	0.016267	0.032627	0.013813	-0.055973	-0.160251	
Chloramines	-0.025428	-0.075316	1.000000	-0.024329	-0.016638	-0.008555	-0.029079	0.003724	
Conductivity	-0.046913	0.016267	-0.024329	1.000000	0.016295	0.020973	0.010788	-0.021495	

	Hardness	Solids	Chloramines	Conductivity	Organic_carbon	Turbidity	ph	Sulfate	T
Organic_carbon	0.019890	0.032627	-0.016638	0.016295	1.000000	-0.024333	0.049783	0.041937	
Turbidity	-0.007436	0.013813	-0.008555	0.020973	-0.024333	1.000000	-0.038699	-0.000653	
ph	0.089940	-0.055973	-0.029079	0.010788	0.049783	-0.038699	1.000000	0.017882	
Sulfate	-0.106678	-0.160251	0.003724	-0.021495	0.041937	-0.000653	0.017882	1.000000	
Trihalomethanes	0.000758	-0.006719	0.017276	0.018741	0.013133	-0.014935	0.018010	-0.033332	

In [49]:
```
import seaborn as sns
corr=X_train.corr()
top_features=corr.index
plt.figure(figsize=(10,7))
sns.heatmap(X_train[top_features].corr(),annot=True)
```

Out[49]:
```
<AxesSubplot:>
```


Hardness
Solids
Chloramines
Conductivity
Organic_carbon
Turbidity
ph
Sulfate
Trihalomethanes
1 -0.035 -0.025 -0.047 0.02 -0.0074 0.09 -0.11 0.00076
-0.035 1 -0.075 0.016 0.033 0.014 -0.056 -0.16 -0.0067
-0.025 -0.075 1 -0.024 -0.017 -0.0086 -0.029 0.0037 0.017
-0.047 0.016 -0.024 1 0.016 0.021 0.011 -0.021 0.019
0.02 0.033 -0.017 0.016 1 -0.024 0.05 0.042 0.013
-0.0074 0.014 -0.0086 0.021 -0.024 1 -0.039 -0.00065 -0.015
0.09 -0.056 -0.029 0.011 0.05 -0.039 1 0.018 0.018
-0.11 -0.16 0.0037 -0.021 0.042 -0.00065 0.018 1 -0.033
0.00076 -0.0067 0.017 0.019 0.013 -0.015 0.018 -0.033 1
1.0
0.8
0.6
0.4
0.2
0.0

Reduction Of `Multi Collinearity`

In [50]:
```
threshold=0.6
```

In [51]:
```
# find and remove correlated features
def correlation(dataset, threshold):
    col_corr = set()  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff valu
                colname = corr_matrix.columns[i]  # getting the name of column
                col_corr.add(colname)
    return col_corr
```

In [52]:
```
correlation(X_train,threshold)
```

Out[52]:
```
set()
```

Information Gain

In [53]:
```
from sklearn.feature_selection import mutual_info_classif
```

In [54]:
```
mutual_info=mutual_info_classif(X_train,y_train)
```

```
In [55]: mutual_data=pd.Series(mutual_info,index=X_train.columns)
         mutual_data.sort_values(ascending=False)
```

```
Out[55]: Turbidity          0.019079
         Sulfate            0.012078
         ph                 0.011437
         Hardness           0.006109
         Conductivity       0.000936
         Organic_carbon     0.000755
         Solids             0.000000
         Chloramines        0.000000
         Trihalomethanes    0.000000
         dtype: float64
```

From the above Information,

let us take the mentioned below Feature wrt, target variable

- Solids
- Chloramines
- Trihalomethanes

```
In [56]: X_train = X_train.drop(columns = ['Solids', 'Chloramines', 'Trihalomethanes'])
```

```
In [57]: X_test = X_test.drop(columns = ['Solids', 'Chloramines', 'Trihalomethanes'])
```

Final Dimensions - After All `Features Engineering` and `Selection Is completed`

```
In [58]: X_test.shape
```

```
Out[58]: (983, 6)
```

```
In [59]: X_train.shape
```

```
Out[59]: (2293, 6)
```

Checking for `Outliers`

Which Machine LEarning Models Are Sensitive To Outliers?

1. Naivye Bayes Classifier--- Not Sensitive To Outliers
2. SVM-------- Not Sensitive To Outliers
3. Linear Regression---------- Sensitive To Outliers
4. Logistic Regression------- Sensitive To Outliers
5. Decision Tree Regressor or Classifier---- Not Sensitive
6. Ensemble(RF,XGboost,GB)------- Not Sensitive
7. KNN--------------------------- Not Sensitive
8. Kmeans------------------------ Sensitive
9. Hierarichal------------------- Sensitive
10. PCA-------------------------- Sensitive
11. Neural Networks-------------- Sensitive

```
In [60]: X_train.columns
```

```
Out[60]: Index(['Hardness', 'Conductivity', 'Organic_carbon', 'Turbidity', 'ph',
               'Sulfate'],
              dtype='object')
```

```
In [61]:
# Checking for outliers in the continuous variables
num_X_train = X_train[['Hardness', 'Conductivity', 'Organic_carbon', 'Turbidity', 'ph','Sulfate']]
```

```
In [62]:
# Checking outliers at 25%, 50%, 75%, 90%, 95% and 99%
num_X_train.describe(percentiles=[.25, .5, .75, .90, .95, .99])
```

Out[62]:

	Hardness	Conductivity	Organic_carbon	Turbidity	ph	Sulfate
count	2293.000000	2293.000000	2293.000000	2293.000000	2293.000000	2293.000000
mean	-0.002888	0.035332	0.015238	-0.000296	0.033365	0.004578
std	0.831040	0.689261	0.734216	0.730168	0.813348	0.790693
min	-2.583734	-2.071391	-2.675587	-2.361877	-3.547225	-3.906685
25%	-0.507013	-0.478332	-0.475566	-0.490746	-0.468916	-0.488943
50%	0.008026	0.001331	-0.007636	-0.015101	0.012503	-0.017732
75%	0.508631	0.516009	0.519653	0.500800	0.548452	0.518344
90%	1.004722	0.949998	0.944848	0.942614	1.058468	0.981380
95%	1.348951	1.211064	1.196574	1.180106	1.402574	1.319435
99%	2.037302	1.639084	1.719739	1.648406	1.970427	1.927068
max	3.168411	2.855968	2.847016	2.394588	3.517462	2.838830

In [63]:
```python
Q1 = X_train.quantile(0.25)
Q3 = X_train.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
Hardness          1.015644
Conductivity      0.994342
Organic_carbon    0.995219
Turbidity         0.991546
ph                1.017368
Sulfate           1.007287
dtype: float64
```

Outlier Treatment

Perhaps the most important hyperparameter in the model is the “contamination” argument, which is used to help estimate the number of outliers in the dataset. This is a value between 0.0 and 0.5 and by default is set to 0.1.

Isolation Forest

`Isolation Forest`, or iForest for short, is a tree-based anomaly detection algorithm.

It is based on modeling the normal data in such a way as to isolate anomalies that are both few in number and different in the feature space.

for reference, https://machinelearningmastery.com/model-based-outlier-detection-and-removal-in-python/

In [64]:
```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import IsolationForest
from sklearn.metrics import mean_absolute_error
```

In [65]:
```
# identify outliers in the training dataset
iso = IsolationForest(contamination=0.1)
yhat = iso.fit_predict(X_train)
```

In [66]:
```
# select all rows that are not outliers
mask = yhat != -1
```

In [67]:
```
X_train = X_train[mask]
```

In [68]:
```
y_train = y_train[mask]
```

In [69]:
```
# summarize the shape of the updated training dataset
print(X_train.shape, y_train.shape)
```

```
(2063, 6) (2063,)
```

PIPELINE CREATION

In [70]:
```
## Pipelines Creation
## 1. Data Preprocessing by using Standard Scaler
## 2. Reduce Dimension using PCA
## 3. Apply  Classifier
```

In [71]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
```

In [72]:
```
pipeline_lr=Pipeline([('scalar1',RobustScaler()),
                     ('pca1',PCA(n_components=2)),
                     ('lr_classifier',LogisticRegression(random_state=0))])
```

In [73]:
```
pipeline_dt=Pipeline([('scalar2',RobustScaler()),
                     ('pca2',PCA(n_components=2)),
                     ('dt_classifier',DecisionTreeClassifier())])
```

In [74]:
```
pipeline_randomforest=Pipeline([('scalar3',RobustScaler()),
                     ('pca3',PCA(n_components=2)),
                     ('rf_classifier',RandomForestClassifier())])
```

```
In [75]: pipeline_gradient_boost=Pipeline([('scalar4',RobustScaler()),
                     ('pca4',PCA(n_components=2)),
                     ('gb_classifier',GradientBoostingClassifier())])
```

```
In [76]: pipeline_XGboost=Pipeline([('scalar5',RobustScaler()),
                     ('pca5',PCA(n_components=2)),
                     ('xgb_classifier',XGBClassifier())])
```

```
In [77]: ## LEts make the list of pipelines
         pipelines = [pipeline_lr, pipeline_dt, pipeline_randomforest,pipeline_gradient_boost,pipeline_XGbc
```

```
In [78]: best_accuracy=0.0
         best_classifier=0
         best_pipeline=""
```

```
In [79]: # Dictionary of pipelines and classifier types for ease of reference
         pipe_dict = {0: 'Logistic Regression', 1: 'Decision Tree', 2: 'RandomForest', 3: 'Gradient Boost',

         # Fit the pipelines
         for pipe in pipelines:
                 pipe.fit(X_train, y_train)
```

```
[18:15:57] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:10
95: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logis
```

```
tic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore th
e old behavior.
```

In [80]:
```
for i,model in enumerate(pipelines):
    print("{} Test Accuracy: {}".format(pipe_dict[i],model.score(X_test,y_test)))
```

```
Logistic Regression Test Accuracy: 0.6256358087487284
Decision Tree Test Accuracy: 0.5289928789420142
RandomForest Test Accuracy: 0.5666327568667345
Gradient Boost Test Accuracy: 0.6012207527975585
Extreme Gradient Boost Test Accuracy: 0.5584944048830112
```

In [81]:
```
for i,model in enumerate(pipelines):
    if model.score(X_test,y_test)>best_accuracy:
        best_accuracy=model.score(X_test,y_test)
        best_pipeline=model
        best_classifier=i
print('Classifier with best accuracy:{}'.format(pipe_dict[best_classifier]))
```

```
Classifier with best accuracy:Logistic Regression
```

In [82]:
```
y_test.value_counts()
```

Out[82]:
```
0.0    615
1.0    368
Name: Potability, dtype: int64
```

Let's Use XGBoost Classifier

In [83]:
```
import xgboost as xgb
```

In [84]:
```
xgb = XGBClassifier(n_estimators=100)
xgb.fit(X_train, y_train)
preds = xgb.predict(X_test)
acc_xgb = (preds == y_test).sum().astype(float) / len(preds)*100
print("XGBoost's prediction accuracy is: %3.2f" % (acc_xgb))
```

```
[18:15:59] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:10
95: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logis
tic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore th
e old behavior.
XGBoost's prediction accuracy is: 58.09
```

In [85]:
```
y_pred = xgb.predict(X_test)
```

In [86]:
```
y_pred[0:20]
```

Out[86]:
```
array([0., 1., 1., 0., 1., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
       0., 0., 0.])
```

In [87]:
```
param_test1 = {
 'max_depth':range(3,10,2),
 'min_child_weight':range(1,6,2)
}
```

In [88]:
```
from sklearn.model_selection import GridSearchCV
gsearch1 = GridSearchCV(estimator = XGBClassifier( learning_rate =0.1, n_estimators=140, max_depth
 min_child_weight=1, gamma=0, subsample=0.8, colsample_bytree=0.8,
 objective= 'binary:logistic', nthread=4, scale_pos_weight=1, seed=27),
 param_grid = param_test1,n_jobs=4, cv=5)
```

In [89]:

```
gsearch1.fit(X_train,y_train)
```

```
[18:17:02] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:10
95: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logis
tic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore th
e old behavior.
```

Out[89]:

```
GridSearchCV(cv=5,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=0.8, gamma=0, gpu_id=None,
                                     importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=0.1, max_delta_step=None,
                                     max_depth=5, min_child_weight=1,
                                     missing=nan, monotone_constraints=None,
                                     n_estimators=140, n_jobs=None, nthread=4,
                                     num_parallel_tree=None, random_state=None,
                                     reg_alpha=None, reg_lambda=None,
                                     scale_pos_weight=1, seed=27, subsample=0.8,
                                     tree_method=None, validate_parameters=None,
                                     verbosity=None),
             n_jobs=4,
             param_grid={'max_depth': range(3, 10, 2),
                         'min_child_weight': range(1, 6, 2)})
```

In [90]:

```
y_pred = gsearch1.predict(X_test)
```

In [91]:
```
from sklearn import metrics
```

Plotting the `Confusion Matrix`

In [92]:
```
# Confusion matrix
confusion = metrics.confusion_matrix(y_test, y_pred)
print(confusion)
```

```
[[520  95]
 [272  96]]
```

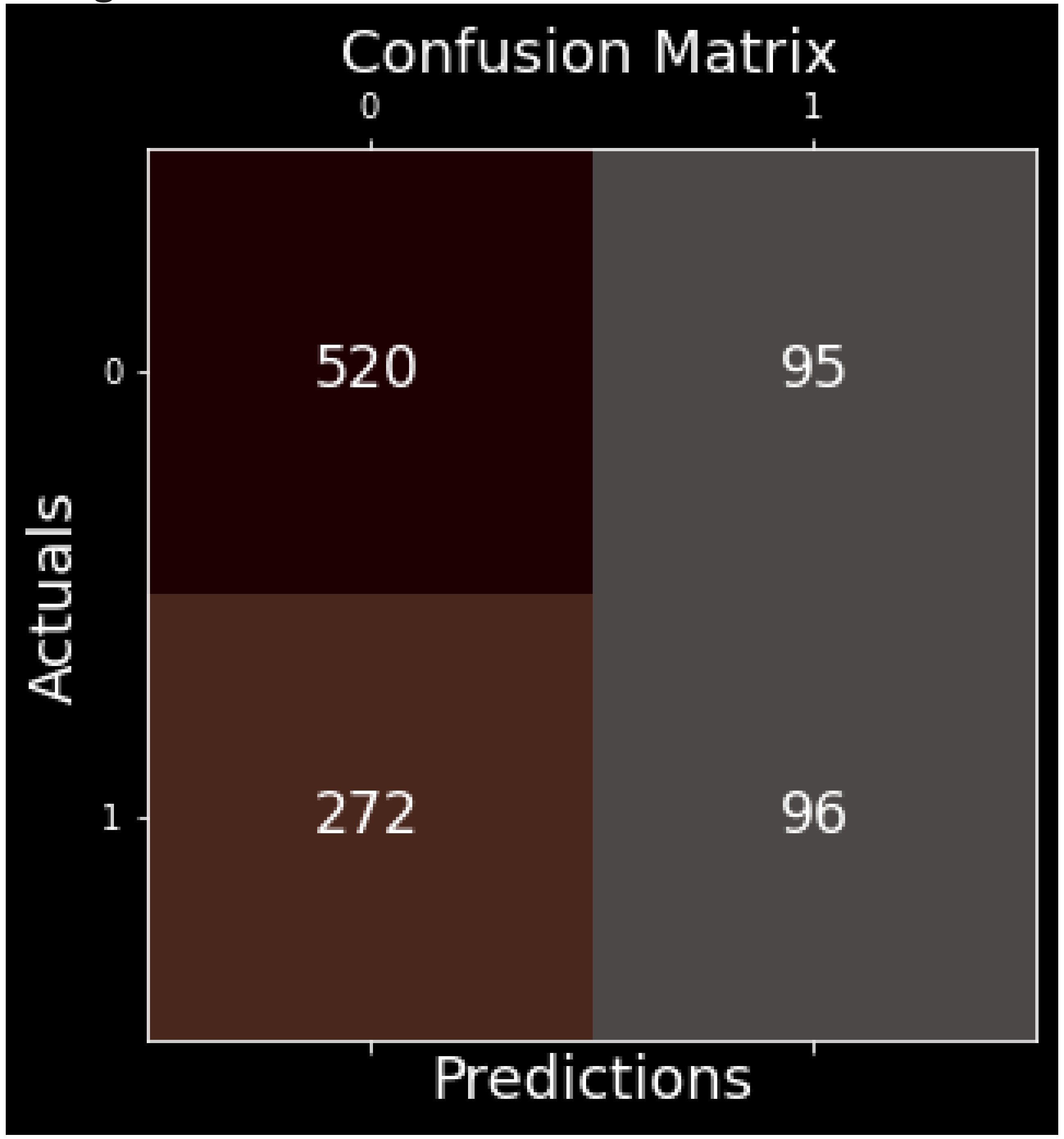
In [94]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.66      0.85      0.74       615
         1.0       0.50      0.26      0.34       368

    accuracy                           0.63       983
   macro avg       0.58      0.55      0.54       983
weighted avg       0.60      0.63      0.59       983
```

In [95]:
```
#
# Print the confusion matrix using Matplotlib
#
plt.figure(figsize = [5,5])
fig, ax = plt.subplots(figsize=(5, 5))
```

```
ax.matshow(confusion, cmap=plt.cm.Reds, alpha=0.3)
for i in range(confusion.shape[0]):
    for j in range(confusion.shape[1]):
        ax.text(x=j, y=i,s=confusion[i, j], va='center', ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```

```
<Figure size 360x360 with 0 Axes>
```



In [96]:

```
# Let's check the overall accuracy.
print(metrics.accuracy_score(y_test, y_pred))
```

```
0.6266531027466938
```

ROC CURVE

An ROC curve demonstrates several things:

- It shows the tradeoff between sensitivity and specificity (any increase in sensitivity will be accompanied by a decrease in specificity).
- The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test.
- The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

In [98]:

```
def draw_roc( actual, probs ):
    fpr, tpr, thresholds = metrics.roc_curve( actual, probs,
                                              drop_intermediate = False )
    auc_score = metrics.roc_auc_score( actual, probs )
    plt.figure(figsize=(5, 5))
    plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score )
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()

    return None
```

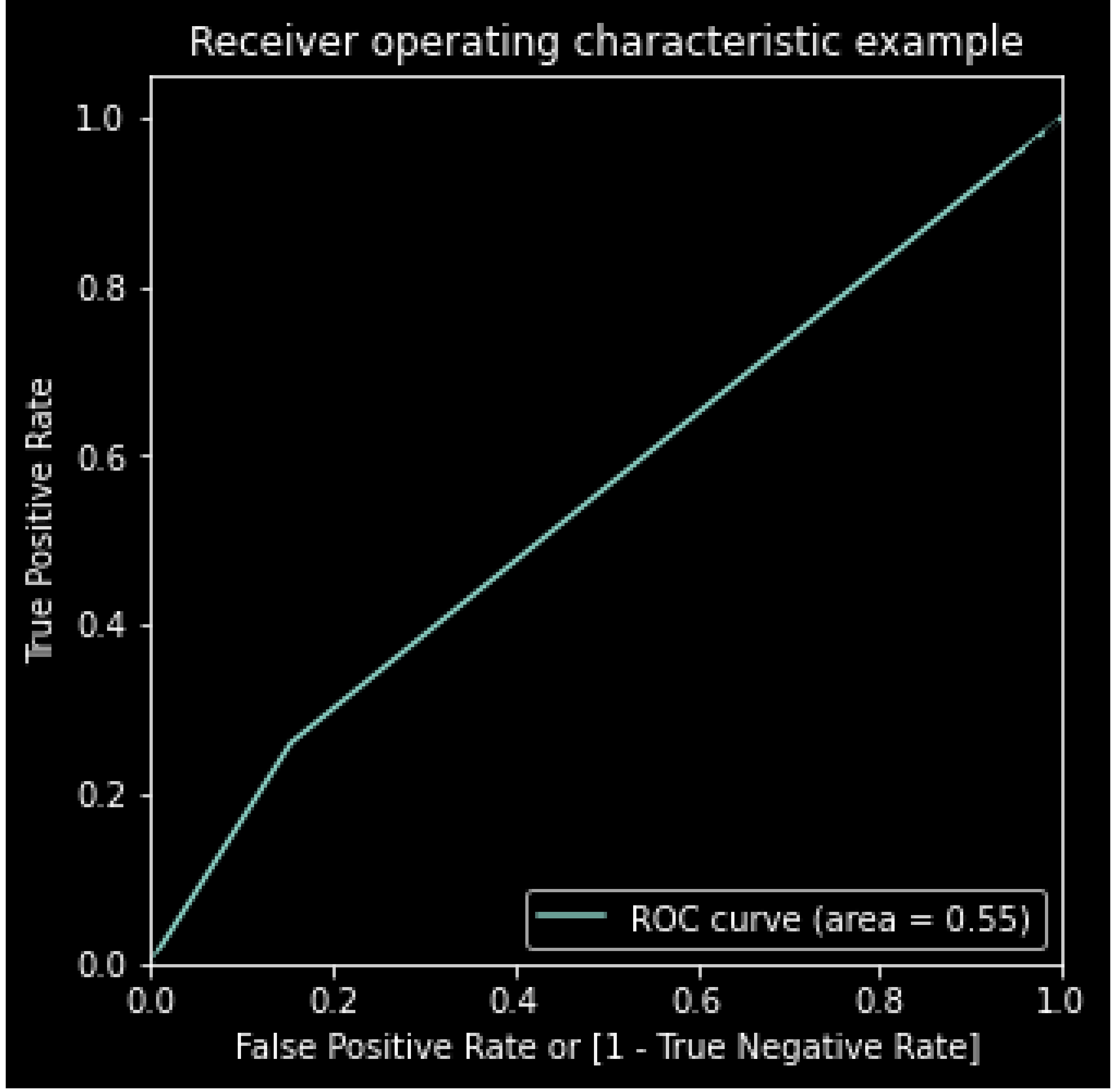
In [101…

```
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred, drop_intermediate = False )
```

In [102…

```
plt.figure(figsize= [8,8])
draw_roc(y_test, y_pred)
```

```
<Figure size 576x576 with 0 Axes>
```



In [106…

```python
# Converting y_pred to a dataframe which is an array
y_pred_1 = pd.DataFrame(y_pred)
```

In [107…

```python
# Let's see the head
y_pred_1.head()
```

Out[107…

```
    0
```

	0
0	0.0
1	0.0
2	1.0
3	0.0
4	0.0

```
In [108…
y_test_df = X_test
```

```
In [109…
# Putting CustID to index
y_test_df['ID'] = y_test_df.index
```

```
In [110…
# Removing index for both dataframes to append them side by side
y_pred_1.reset_index(drop=True, inplace=True)
y_test_df.reset_index(drop=True, inplace=True)
```

```
In [111…
# Appending y_test_df and y_pred_1
y_pred_final = pd.concat([y_test_df, y_pred_1],axis=1)
```

```
In [112…
y_pred_final.head()
```

Out[112…

	Hardness	Conductivity	Organic_carbon	Turbidity	ph	Sulfate	ID	0
0	0.353158	0.596203	0.065597	0.333012	0.497954	1.081856	1200	0.0
1	-0.755977	0.335014	-0.632167	-1.037612	-0.095916	0.704040	825	0.0
2	0.336765	-0.861790	-0.608839	0.171527	0.561325	1.003252	1781	1.0
3	-0.602744	1.290252	-0.594067	0.726300	-0.446209	0.003313	2596	0.0
4	1.296179	-0.690386	0.105650	-1.329702	0.759412	0.637447	454	0.0

In [113…

```
y_pred_final = y_pred_final.rename(columns={0:"Water_Quality_Pred"})
```

In [114…

```
y_pred_final.head(4)
```

Out[114…

	Hardness	Conductivity	Organic_carbon	Turbidity	ph	Sulfate	ID	Water_Quality_Pred
0	0.353158	0.596203	0.065597	0.333012	0.497954	1.081856	1200	0.0
1	-0.755977	0.335014	-0.632167	-1.037612	-0.095916	0.704040	825	0.0
2	0.336765	-0.861790	-0.608839	0.171527	0.561325	1.003252	1781	1.0
3	-0.602744	1.290252	-0.594067	0.726300	-0.446209	0.003313	2596	0.0

In [115…

```
y_pred_final.Water_Quality_Pred.value_counts()
```

```
0.0    792
```

```
Out[115…  1.0    191
          Name: Water_Quality_Pred, dtype: int64
```

```
In [124…  y_test.value_counts()
```

```
Out[124…  0.0    615
          1.0    368
          Name: Potability, dtype: int64
```

```
In [ ]:
```