School of Computer Science and Engineering

J Component report

Programme **:** B Tech in CSE

Course Title **:** Machine Learning

Course Code **:** CSE4020

Slot **:** D2

Title **:** Water Quality Analysis

Team Members : 20BCE1182
Sheral Simon Waskar

20BCE1320
Anjali Jain

Faculty **:** Dr. TulasiPrasad Sariki

ABSTRACT

Water quality analysis is a critical task for ensuring safe and sustainable water supply for human consumption and ecological preservation. Traditional water quality monitoring methods involve time-consuming and labor-intensive manual sampling and laboratory analysis, which can delay the detection of contaminants and health risks. This method takes in multiple inputs to measure the Water Quality Index (WQI) and potability of the water bodies. The inputs taken into consideration for the estimation of potability are pH, Hardness, Solids, Chloramines, Sulfate, Conductivity, Organic carbon, Trihalomethanes and Turbidity.Outlier analysis is done and the values are capped according to the values of upper limit and lower limit respectively. Various machine learning algorithms were applied out of which random forest and XG Boost provided the best accuracy on test data.

KEYWORDS

Outliers;XG Boost;Water Quality Index (WQI); Decision Tree; Random Forest; Potability;

INTRODUCTION

Water is used in multiple practices such as drinking, farming, household works and various industries and the quality of water is one of the crucial factors to maintain the hygiene of both public and the environment. Among the available water resources, rivers and ponds are easy access to the public. There were several investigations made related to rivers around the world to study the changes in terms of water quality since it is important in the development of civilization. Using ground water without suitable minerals for daily use or seawater for irrigation purposes results in bad productivity. The study of water is often very common in the earth sciences and it generally evaluates the quality that enhances the productivity of human societies. Even the governments have created projects to utilize the water bodies to create electricity.

Water quality analysis is a critical task for ensuring safe and sustainable water supply for human consumption and ecological preservation. Traditional water quality monitoring methods involve time-consuming and labor-intensive manual sampling and laboratory analysis, which can delay the detection of contaminants and health risks. In recent years, machine learning has emerged as a powerful tool for water quality analysis, enabling real-time monitoring and detection of water quality issues using a range of physical, chemical, and biological parameters. By analyzing large amounts of water quality data, machine learning algorithms can identify patterns and predict potential contaminants and health risks.

The water quality refers to the chemical, physical characteristics of water that are based on various standards of its usage. Various elements have different limitations of their presence in the water bodies and if in excess, can contaminate the water with bacteria, viruses, salts and other organic chemicals. Due to the industry's annual expansion in response to the demand's exponential growth and the discharge of hazardous waste into rivers and lakes by those industries, water pollution is a common problem. The surveys conducted world-wide on various levels have stated how even bottled water might contain a portion of bacteria with the increase in water pollutants. Excessive chemicals result in acid rain which in-turn damages the groundwater and the soil that would be useless for future irrigation. Though traditional methods like manually collecting water samples and analyzing them in a lab to test the quality of the water is practiced, it can be costly and time-consuming.Even sensors can be installed in water bodies but using sensors to test every aspect of water quality is expensive. So to monitor the water quality we here, use various machine learning techniques. This project deals with the potability of water that shows the possibility of water being used for drinking and other purposes.Water quality metrics such as pH, solids, chloramines, sulfate, conductivity, turbidity etc were considered.

LITERATURE SURVEY

The paper “Efficient Water Quality Prediction Using Supervised Machine Learning” by Umair Ahmed, Rafia Mumtaz, Hirra Anwar, Asad A. Shah, Rabia Irfan and José Garcia-Nieto shows how certain supervised machine learning algorithms can be used to estimate the water quality index (WQI) and the water quality class (WQC) which is a distinctive class that is defined based on the estimated WQI. Over four parameters were taken in their study i.e., temperature, pH, turbidity and total amount of dissolved solids. The most efficient WQI was predicted using the gradient boosting, with a learning rate of 0.1 and polynomial regression, with a degree of 2. The multi-layer perceptron classified the WQC most efficiently with an accuracy of over 0.8507. This methodology presented reasonable accuracy with the usage of minimal parameters that can be used to validate their use in real time applications of water body quality tests.

In “Water quality prediction and classification based on principal component regression and gradient boosting classifier approach” by Md. Saikat Islam Khan, Nazrul Islam, Jia Uddin, Sifatul Islama, and Mostofa Kamal Nasir, the water quality prediction model was developed such that it utilizes the principal component regression technique. In their paper, the WQI was initially calculated using the weighted arithmetic index method and the principal component analysis (PCA) was applied to their dataset that extracted the parameter with dominant WQI. The Gradient Boosting classifier was utilized to classify the water quality status and it resulted in the 95% prediction accuracy for the principal component regression model and a 100% classification accuracy. This presented credible performance over state-of-art models in the past.

In the study, ”Water Quality Prediction Using Artificial Intelligence Algorithms”, the researchers Theyazn H. H Aldhyani, Mohammed Al-Yaari, Hasan Alkahtani and Mashael Maashi have developed advanced artificial intelligence (AI) algorithms to predict the water quality index (WQI) and the water classification (WQC). Artificial neural networks such as nonlinear auto regressive neural networks (NARNET) and long short-term memory (LSTM) were developed to predict the WQI. Over three machine learning algorithms, namely, support vector machine (SVM), K-nearest neighbor (K-NN), and Naïve Bayes for WQS forecasting. According to prediction findings, the LSTM performed marginally better than the NARNET model in predicting WQI values, and the SVM method had the highest accuracy (97.01%) in predicting WQC. Furthermore, with only a minor difference in the regression coefficient, the NARNET and LSTM models both achieved equivalent accuracy for the testing phase.

The pattern applications in the paper “Water quality prediction using machine learning methods” proposed by Amir Hamzeh Haghiabi; Ali Heidar Nasrolahi; Abbas Parsaie presents the use of group method to data handling and support vector machine to predict the water quality at the local water bodies of southwest of iran. Different transfer and kernel function types were investigated in order to construct the ANN and SVM, respectively. Reviewing the ANN and SVM findings revealed that both models perform well at predicting the various elements of water quality. The comparison of the outcomes of other models to the GMDH model shows the acceptance performance for predicting the components for water quality. The outcome accuracy was less than what was observed in ANN and SVM. By examining the results, it was evident of the presence of over estimation properties in the bodies.

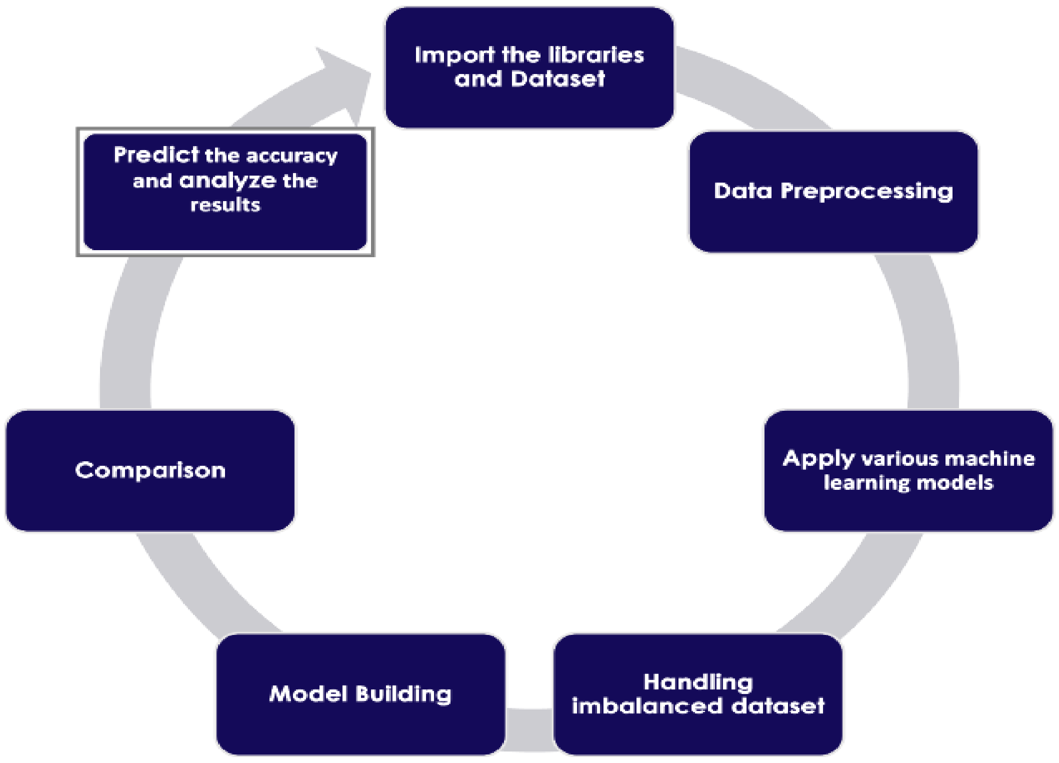
In the research blog, “Urban Water Quality Prediction based on Multi-task Multi- view Learning” issued by Ye Liu, Yu Zheng, Yuxuan Liang, Shuming Liu, David
S. Rosenblum shows the prediction of water quality to control the water pollution and help human health. In this method, by fusing several datasets from various domains, they have used a multi-task multi-view learning method to predict the water quality of a station over the next few hours. Their working model comprised of two alignments that is the spatio-temporal view alignment, which integrates the local spatial and temporal information of each station, is the initial alignment followed by second alignment that is the station prediction alignment, which captures the spatial correlations between the stations and conducts co-predictions by taking these correlations into account. The effectiveness of the approach has been demonstrated using the real time dataset samples in their paper.

PROBLEM STATEMENT

Develop a model that can accurately predict the quality of water based on various physical, chemical, and biological parameters. The model should be trained on historical data of water quality samples and should be able to identify potential contaminants and health risks in real-time to ensure safe and clean water supply for human consumption and ecological sustainability.

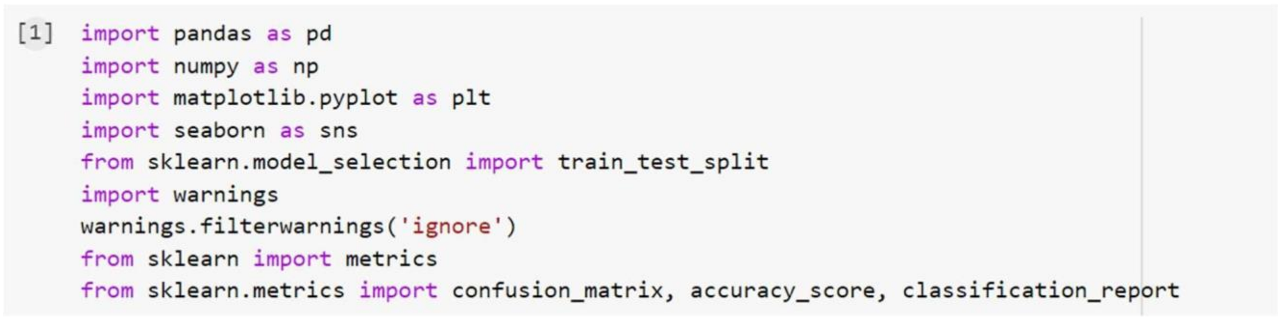
PROPOSED METHODOLOGY

The previous approaches had an implementation of Naive Bayes for WQS forecasting and to predict the Water Quality Index of the water bodies. The application of the multi-task multi-view learning method was used to predict the water quality over time with their two alignments i.e., the spatio-temporal view alignment and station prediction alignment. The correlations were used to conduct the co-predictions of the parameters. In our methodology, there were many missing values in each parameter, so to avoid a high count of outliers, we have taken the mean value imputation for the missing values. The capping has been initiated with the help of IQR where the values above the maximum count were capped to maximum value and values less than the minimum value were capped to minimum value. The data is then taken into account for the exploratory data analysis of each parameter to have an overall visualization for easy understanding. Different machine learning models such as Random Forest,Logistic Regression,KNN, Decision Tree,Gaussian Naive Bayes and Bernoulli Naive Bayes have been applied to find the best possible accuracy in predicting the water quality through the provided parameters. It is found that the Random Forest and XGBoost are the best machine learning model that provides an efficient accuracy of 0.79.


Import the libraries and Dataset
Data Preprocessing
Apply various machine learning models
Handling imbalanced dataset
Model Building
Comparison
Predict the accuracy and analyze the results

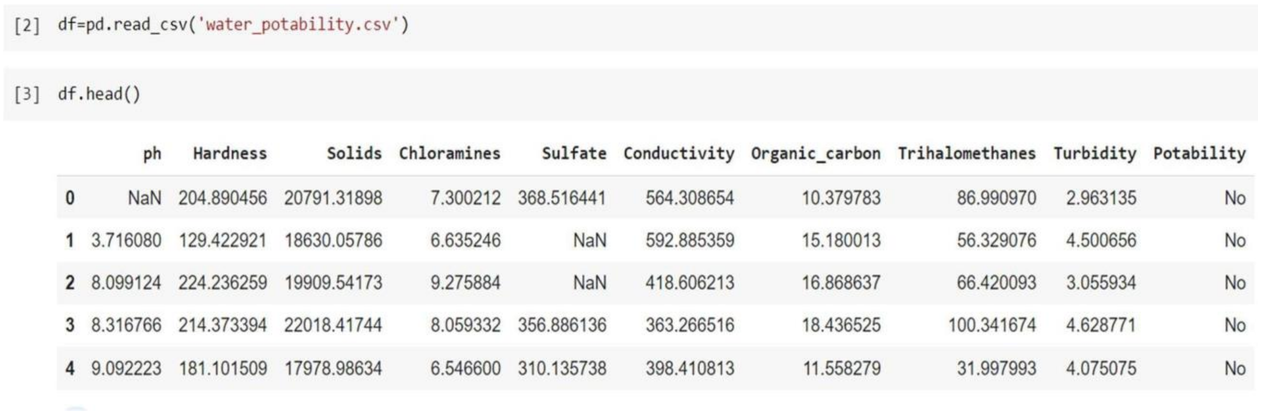
IMPLEMENTATION

Importing the libraries



```
[1] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    from sklearn.model_selection import train_test_split
    import warnings
    warnings.filterwarnings('ignore')
    from sklearn import metrics
    from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
```

Importing the dataset



```
[2] df=pd.read_csv('water_potability.csv')
```

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

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	NaN	204.890456	20791.31898	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	No
1	3.716080	129.422921	18630.05786	6.635246	NaN	592.885359	15.180013	56.329076	4.500656	No
2	8.099124	224.236259	19909.54173	9.275884	NaN	418.606213	16.868637	66.420093	3.055934	No
3	8.316766	214.373394	22018.41744	8.059332	356.886136	363.266516	18.436525	100.341674	4.628771	No
4	9.092223	181.101509	17978.98634	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	No

<u>Information about the dataset</u>

Column description:

1. pH: A measure of how acidic/basic water is.

2. Hardness: Capacity of water to precipitate soap in mg/L.

3. Solids: Total dissolved solids in ppm.

4. Chloramines: Amount of Chloramines in ppm.

5. Sulfate: Amount of Sulfates dissolved in mg/L.

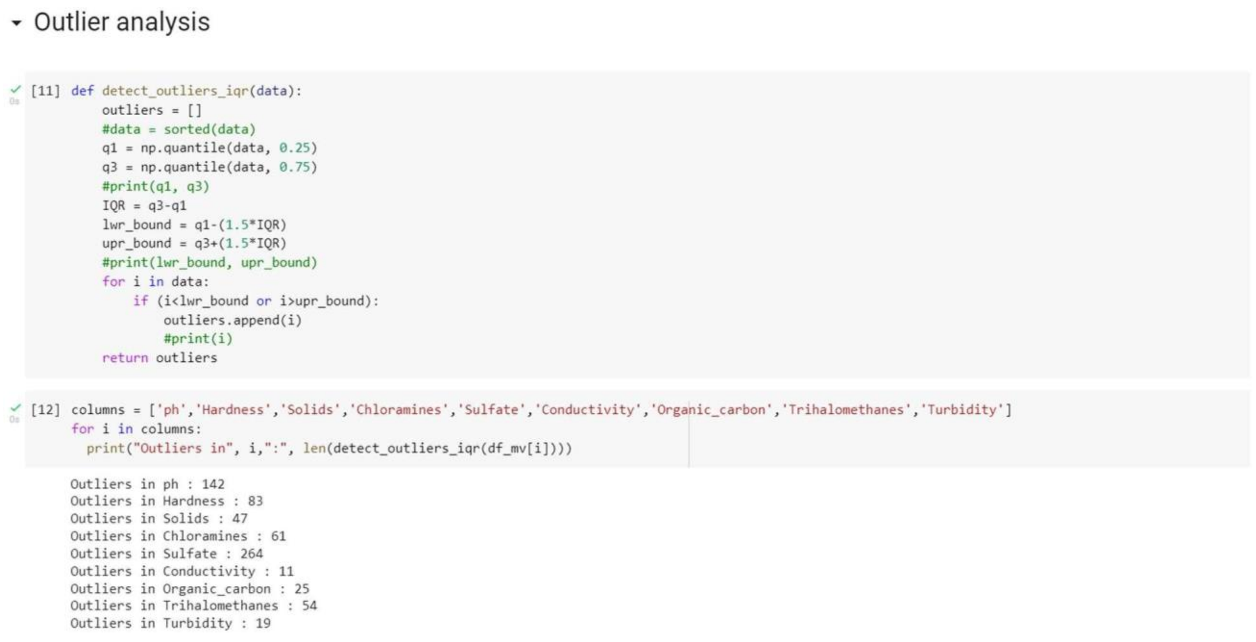
6. Conductivity: Electrical conductivity of water in μS/cm.

7. Organic carbon: Amount of organic carbon in ppm.

8. Trihalomethanes: Amount of Trihalomethanes in μg/L.

9. Turbidity: Measure of light emitting property of water in NTU.

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



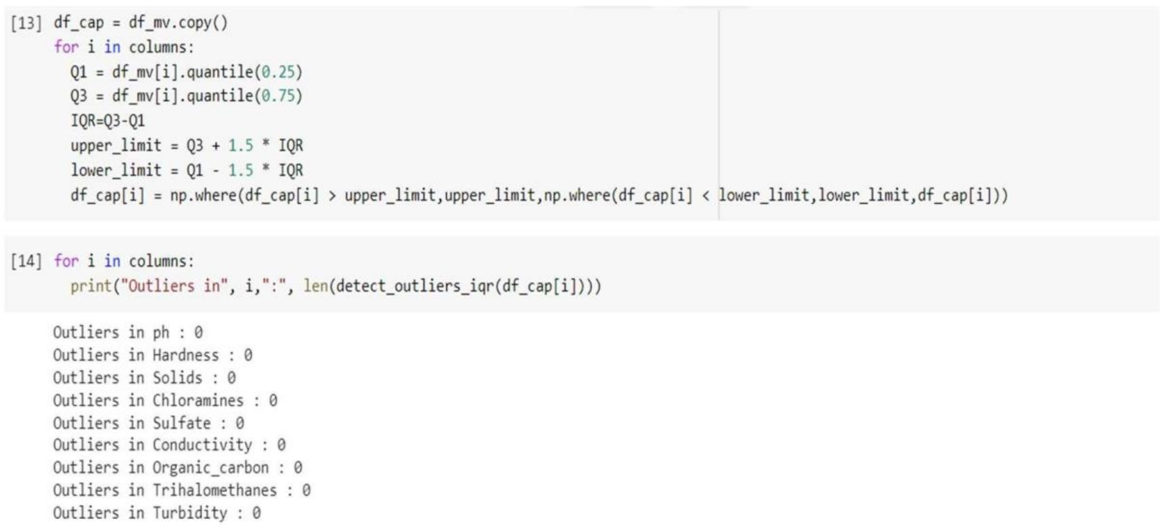
Outlier analysis

```
[11] def detect_outliers_iqr(data):
         outliers = []
         #data = sorted(data)
         q1 = np.quantile(data, 0.25)
         q3 = np.quantile(data, 0.75)
         #print(q1, q3)
         IQR = q3-q1
         lwr_bound = q1-(1.5*IQR)
         upr_bound = q3+(1.5*IQR)
         #print(lwr_bound, upr_bound)
         for i in data:
             if (i<lwr_bound or i>upr_bound):
                 outliers.append(i)
                 #print(i)
         return outliers
```

```
[12] columns = ['ph','Hardness','Solids','Chloramines','Sulfate','Conductivity','Organic_carbon','Trihalomethanes','Turbidity']
     for i in columns:
       print("Outliers in", i,":", len(detect_outliers_iqr(df_mv[i])))
```

```
Outliers in ph : 142
Outliers in Hardness : 83
Outliers in Solids : 47
Outliers in Chloramines : 61
Outliers in Sulfate : 264
Outliers in Conductivity : 11
Outliers in Organic_carbon : 25
Outliers in Trihalomethanes : 54
Outliers in Turbidity : 19
```

Outliers for each of the input columns have been identified using the IQR method. We can create a "fence" outside of Q1 and Q3 using the IQR method for spotting outliers. Outliers are any values that lie outside of this range. To construct this fence, we multiply the IQR by 1.5, subtract Q1 from this result, and add Q3 to the result.



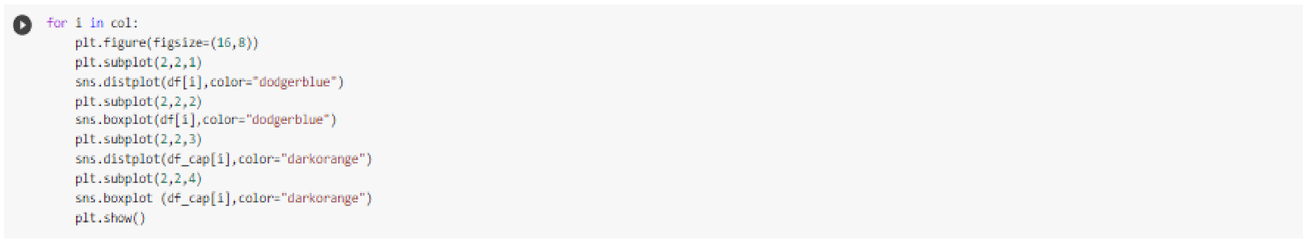
```
[13] df_cap = df_mv.copy()
     for i in columns:
       Q1 = df_mv[i].quantile(0.25)
       Q3 = df_mv[i].quantile(0.75)
       IQR=Q3-Q1
       upper_limit = Q3 + 1.5 * IQR
       lower_limit = Q1 - 1.5 * IQR
       df_cap[i] = np.where(df_cap[i] > upper_limit,upper_limit,np.where(df_cap[i] < lower_limit,lower_limit,df_cap[i]))
```

```
[14] for i in columns:
       print("Outliers in", i,":", len(detect_outliers_iqr(df_cap[i])))
```

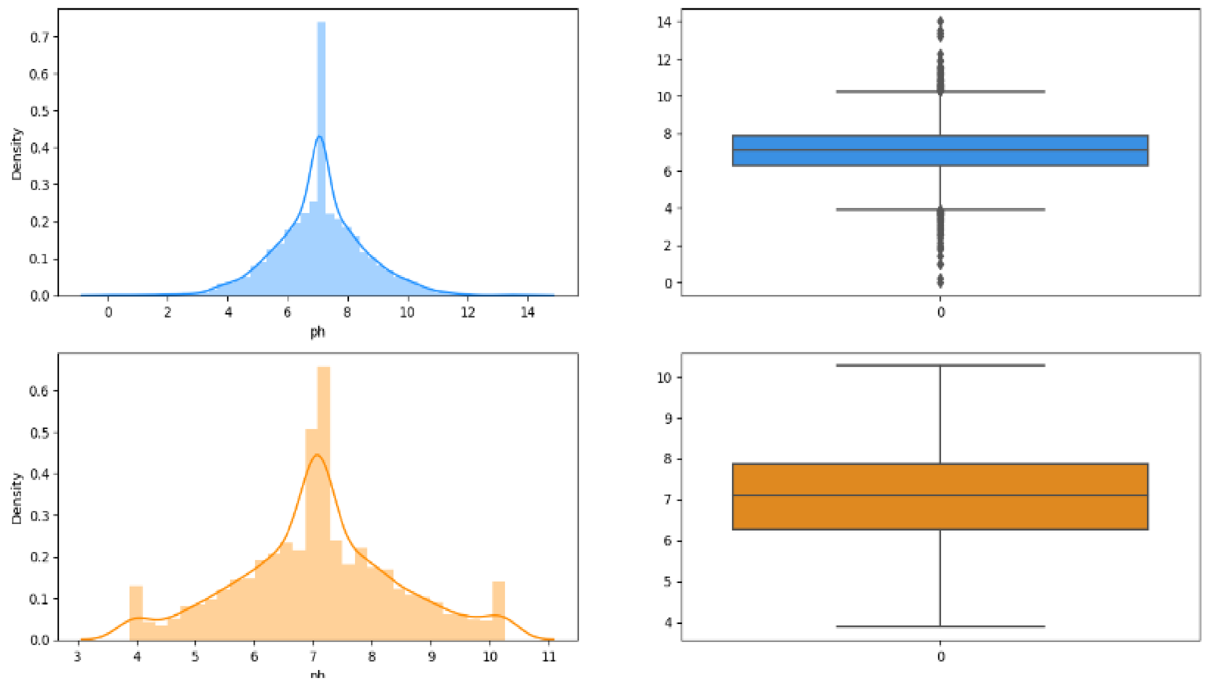
```
Outliers in ph : 0
Outliers in Hardness : 0
Outliers in Solids : 0
Outliers in Chloramines : 0
Outliers in Sulfate : 0
Outliers in Conductivity : 0
Outliers in Organic_carbon : 0
Outliers in Trihalomethanes : 0
Outliers in Turbidity : 0
```

In this method, we cap our outliers to the respective limit value, meaning that any value above upper limit will be replaced by the upper limit and below lower limit value will be replaced by the lower limit. Here outliers are not eliminated to maintain the dimensionality of the dataset.

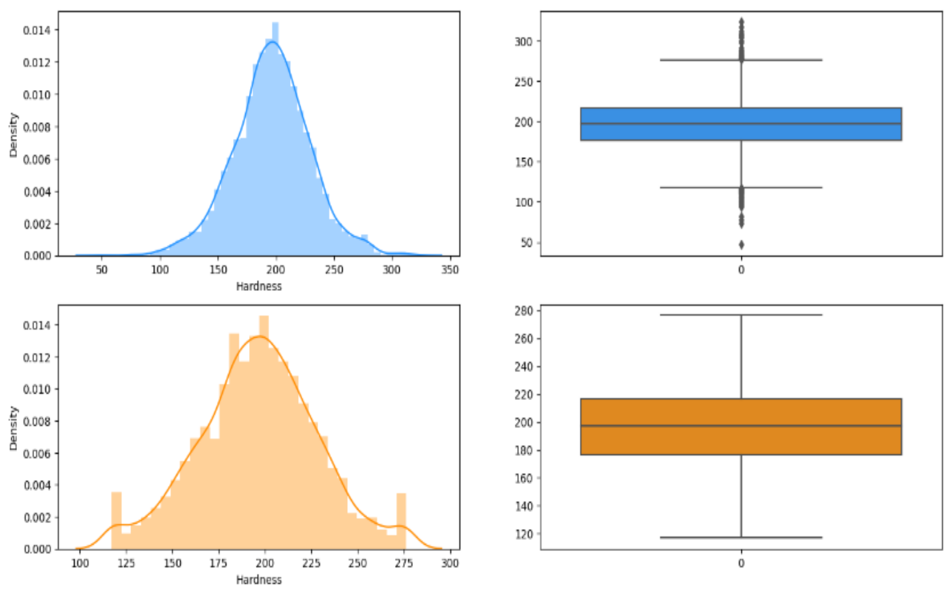
Ph:



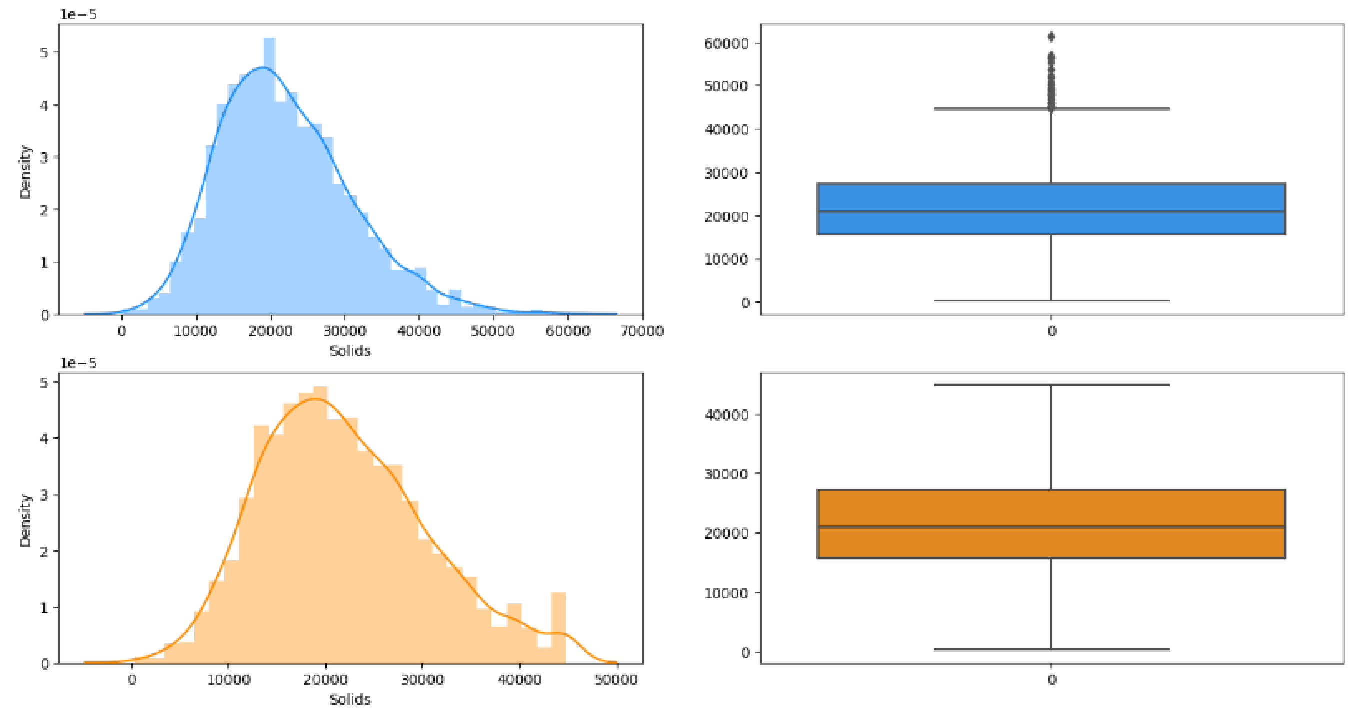
```
for i in col:
    plt.figure(figsize=(16,8))
    plt.subplot(2,2,1)
    sns.distplot(df[i],color="dodgerblue")
    plt.subplot(2,2,2)
    sns.boxplot(df[i],color="dodgerblue")
    plt.subplot(2,2,3)
    sns.distplot(df_cap[i],color="darkorange")
    plt.subplot(2,2,4)
    sns.boxplot (df_cap[i],color="darkorange")
    plt.show()
```



Hardness:

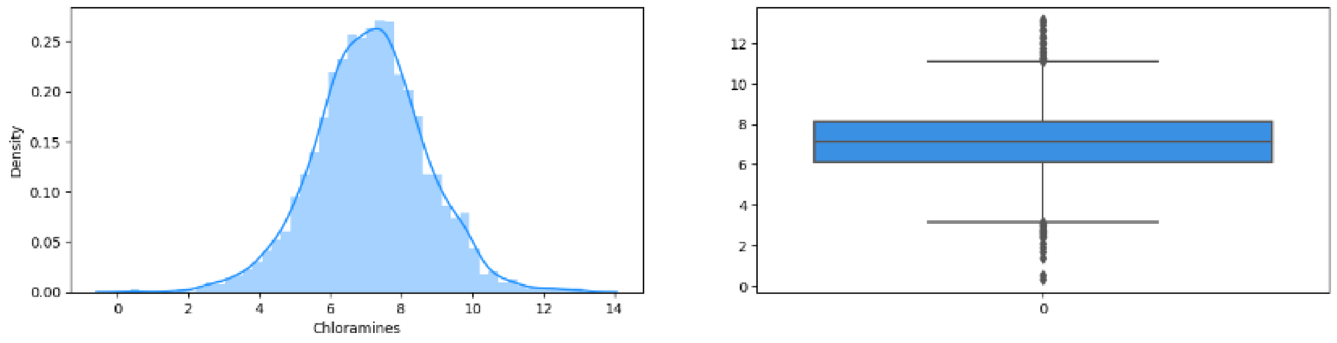


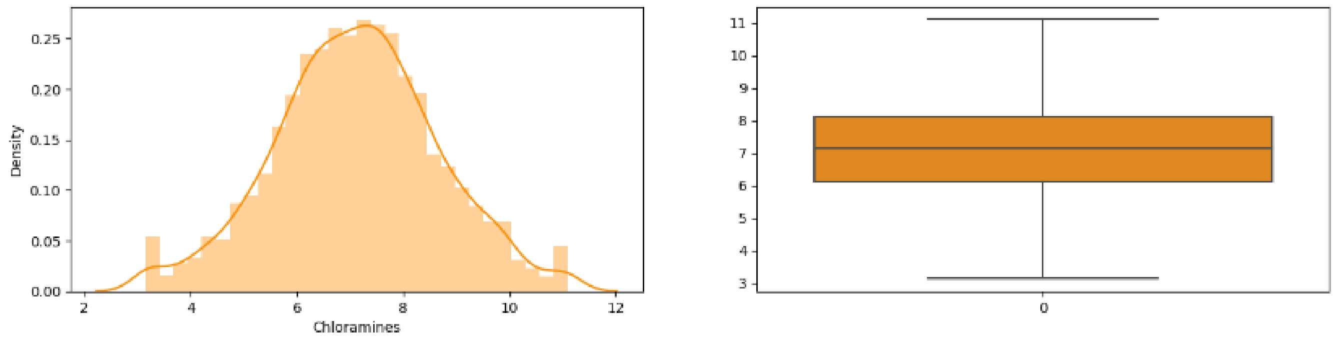
Solids:



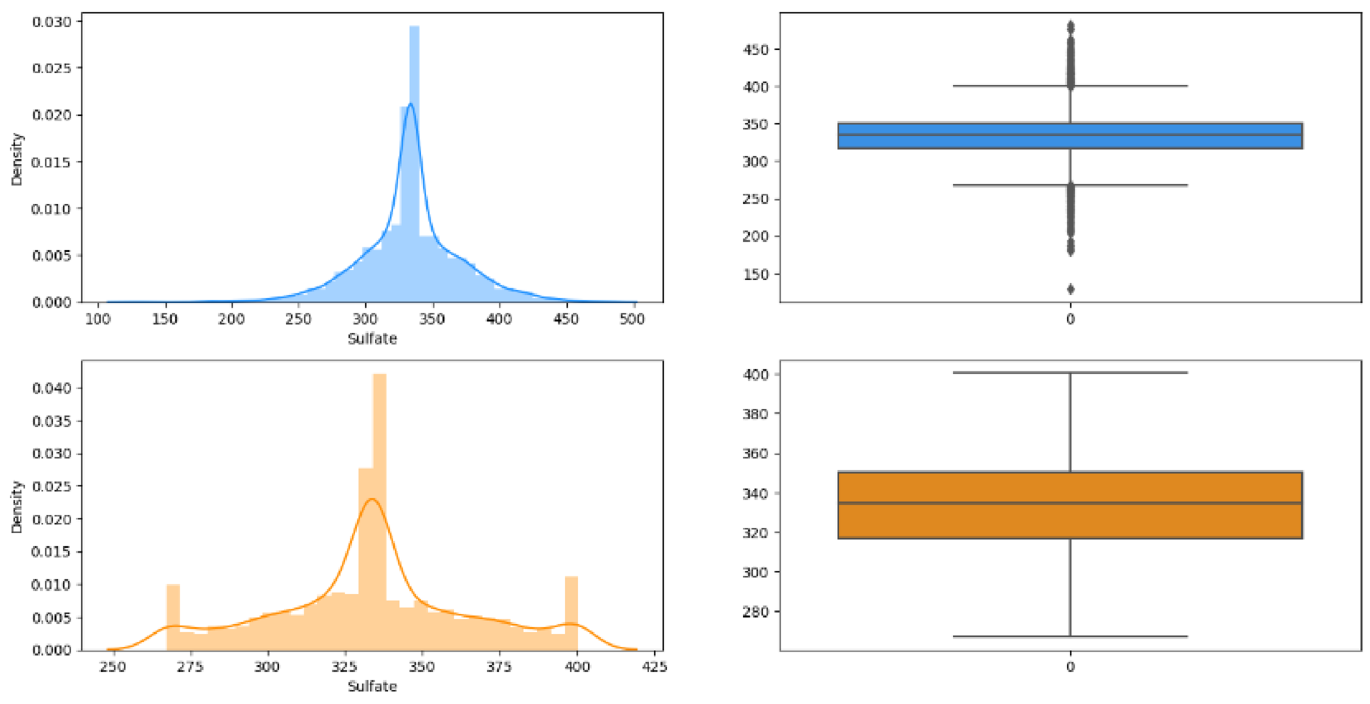
Chloramines:

Solids

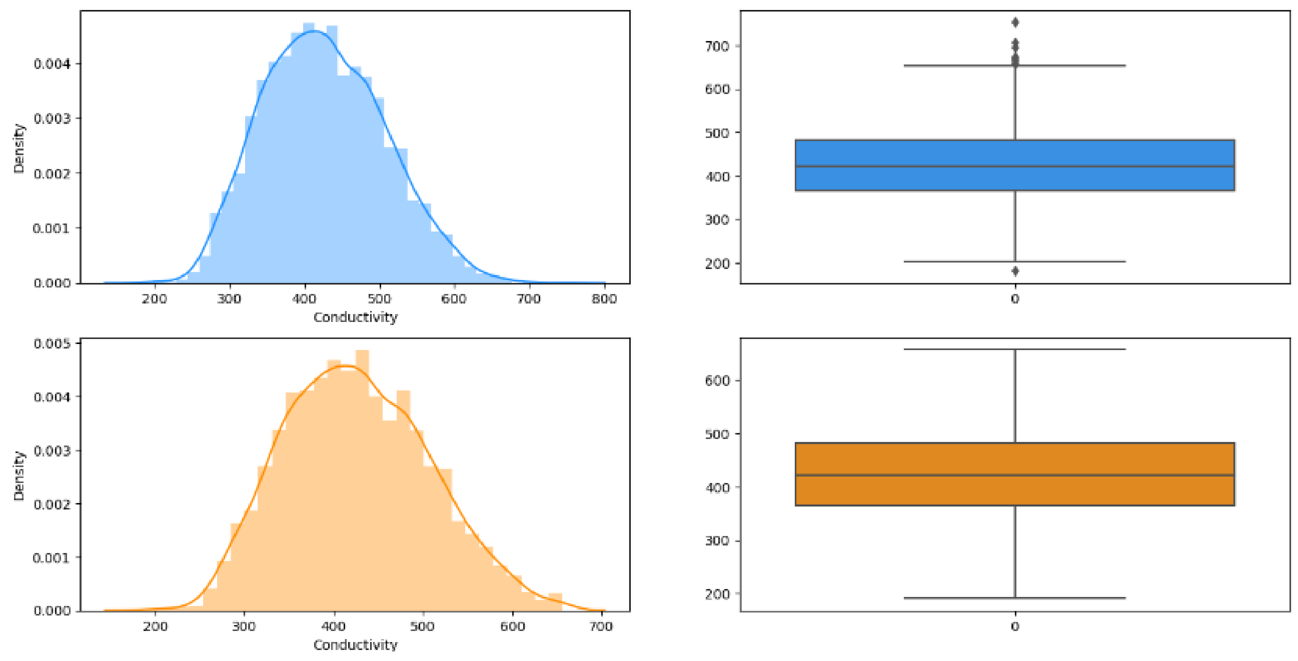

Density
0.00
0.05
0.10
0.15
0.20
0.25
0
2
4
6
8
10
12
14
Chloramines
0
2
4
6
8
10
12
0


Density
0.00
0.05
0.10
0.15
0.20
0.25
2
4
6
8
10
12
Chloramines
3
4
5
6
7
8
9
10
11
0

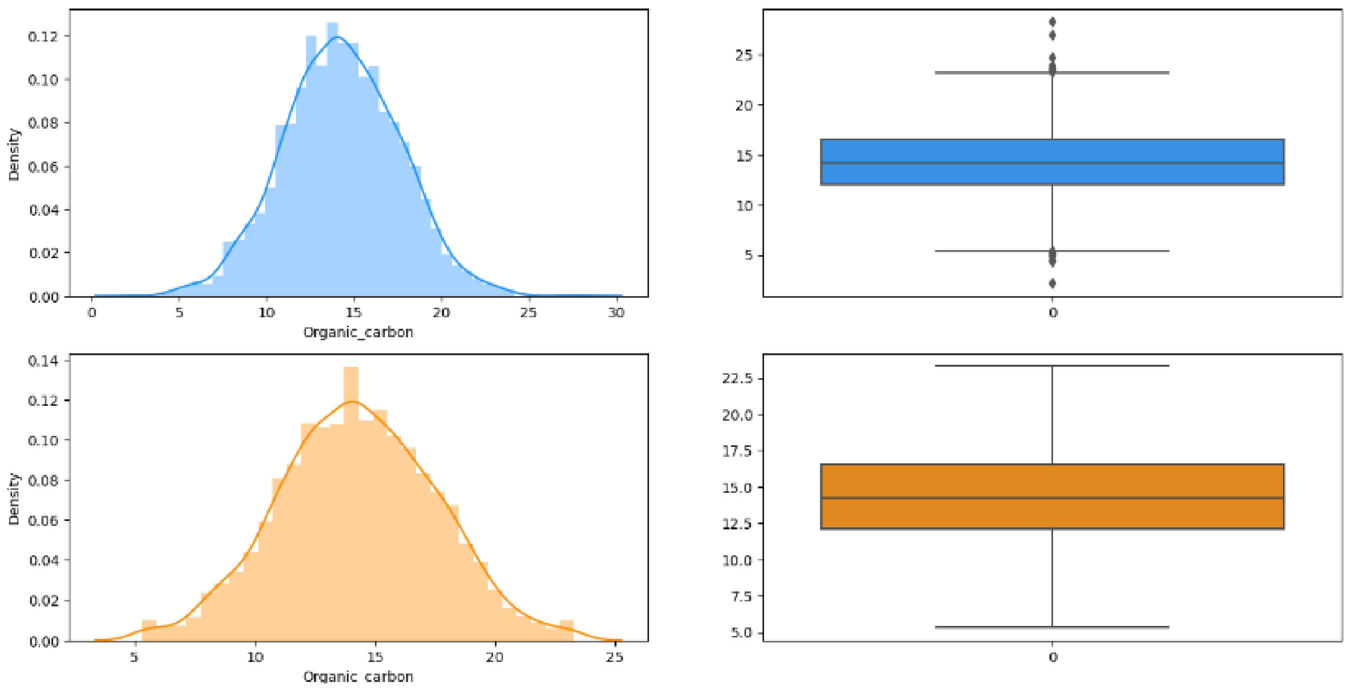
Sulfates:



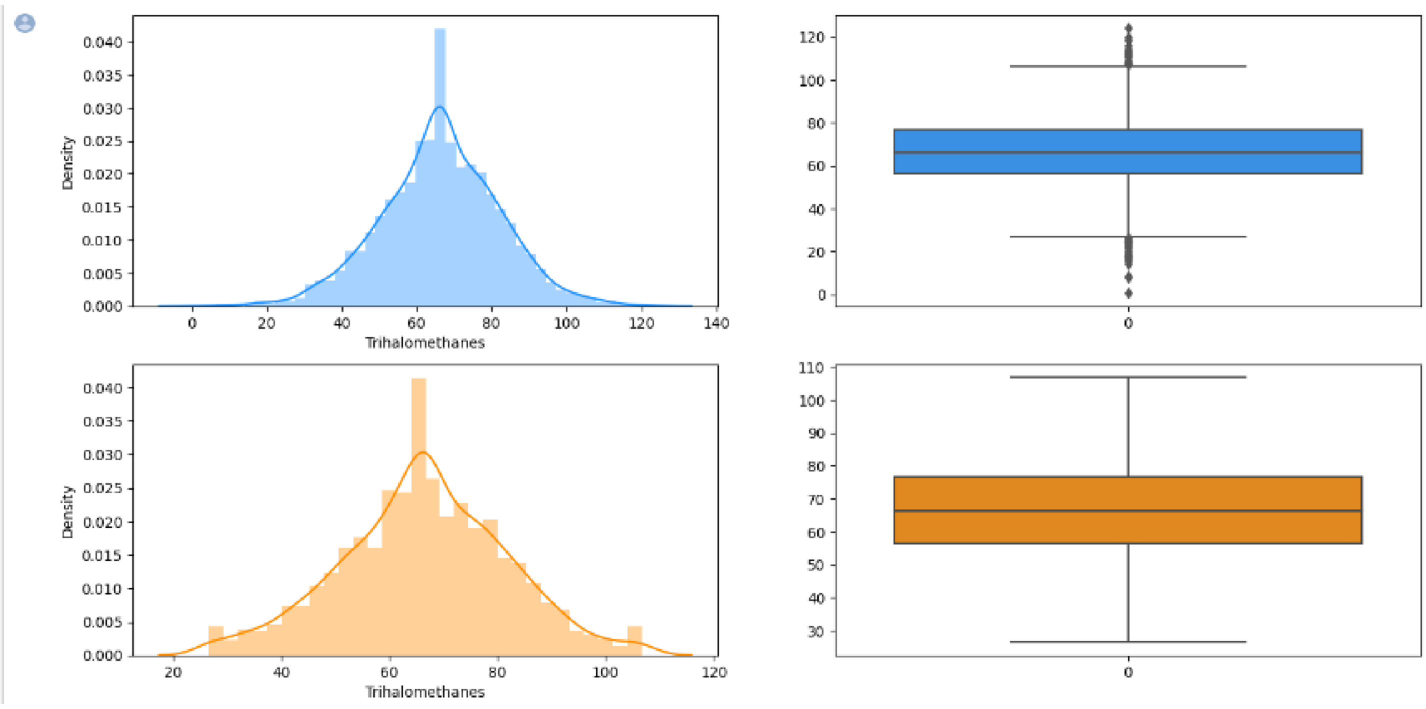
Conductivity:



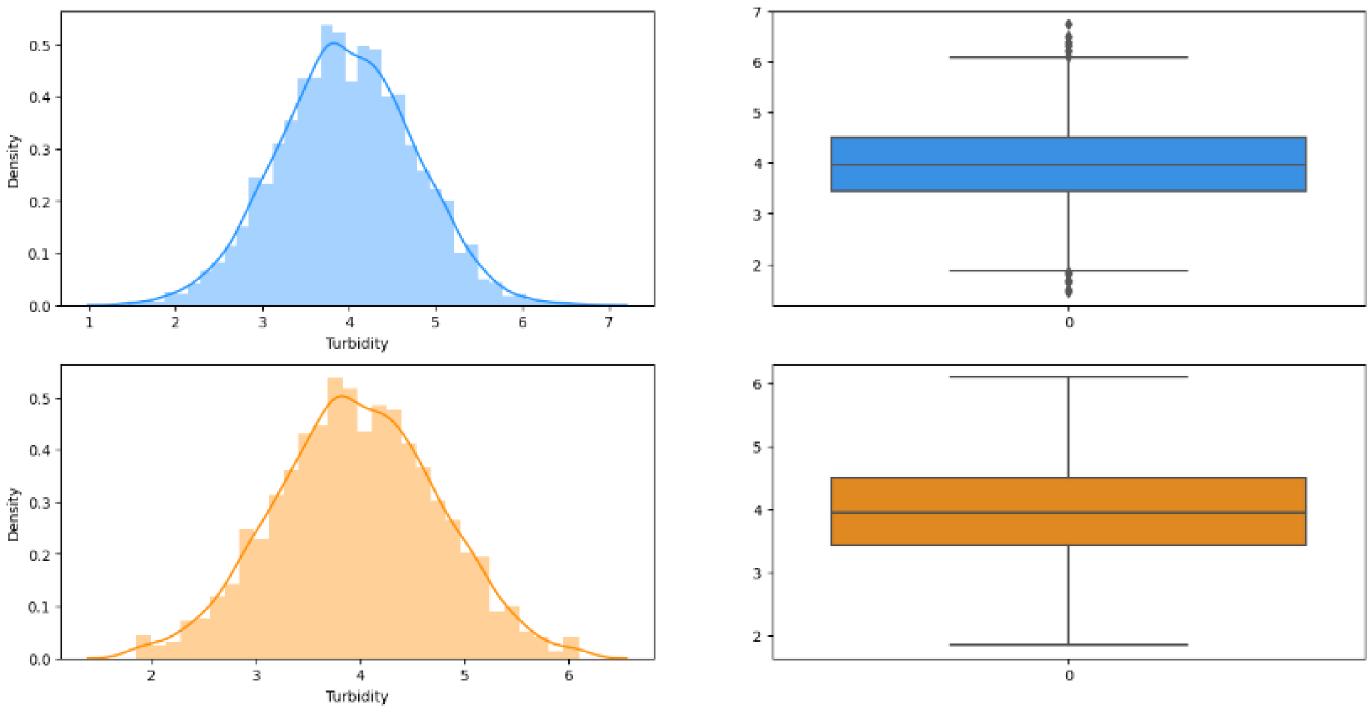
Organic carbon:


Density
Organic_carbon
Density
Organic_carbon

Trihalomethanes:



Turbidity:



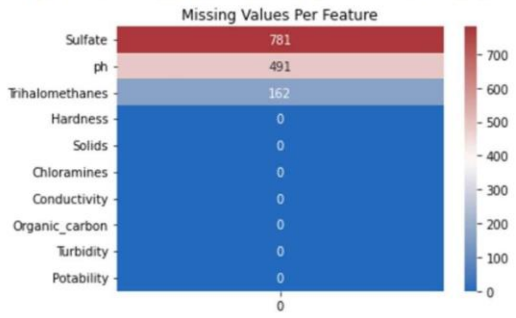
All the outliers are capped to the upper or lower bounds accordingly and no outliers are detected anymore in the dataset and are ready for the analysis of the data. Now, each box plot of the input variable depicts the distribution before and after capping of the data.

```
[6] 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
```

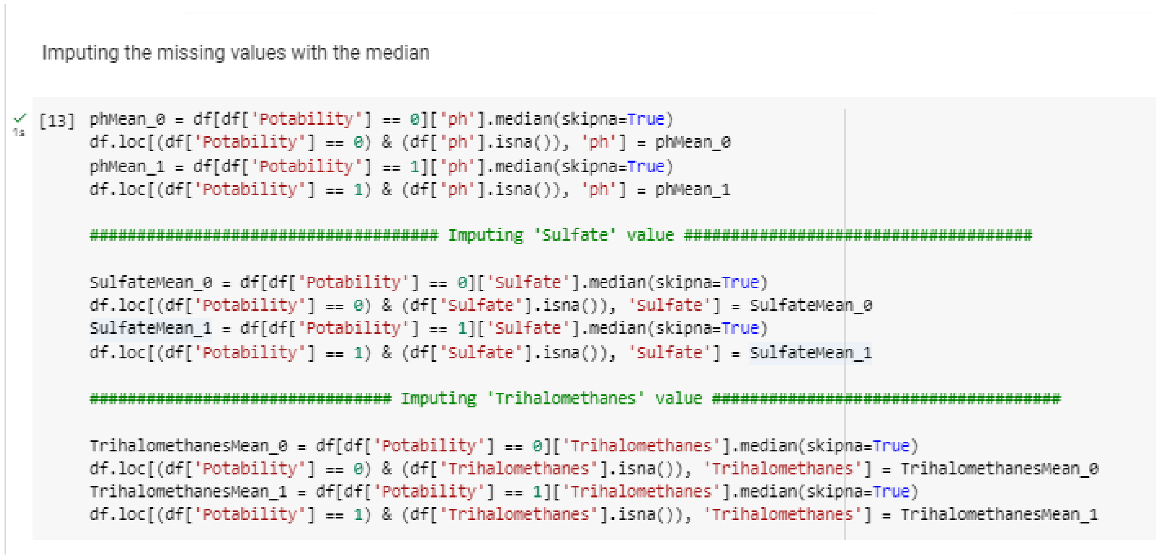
```
[7] plt.title('Missing Values Per Feature')
    nans = df.isna().sum().sort_values(ascending=False).to_frame()
    sns.heatmap(nans,annot=True,fmt='d',cmap='vlag')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f530b3d0e50>
```



```
[9] print('Checking to see any more missing data \n')
    df.isna().sum()
```

```
Checking to see any more missing data

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


Imputing the missing values with the median

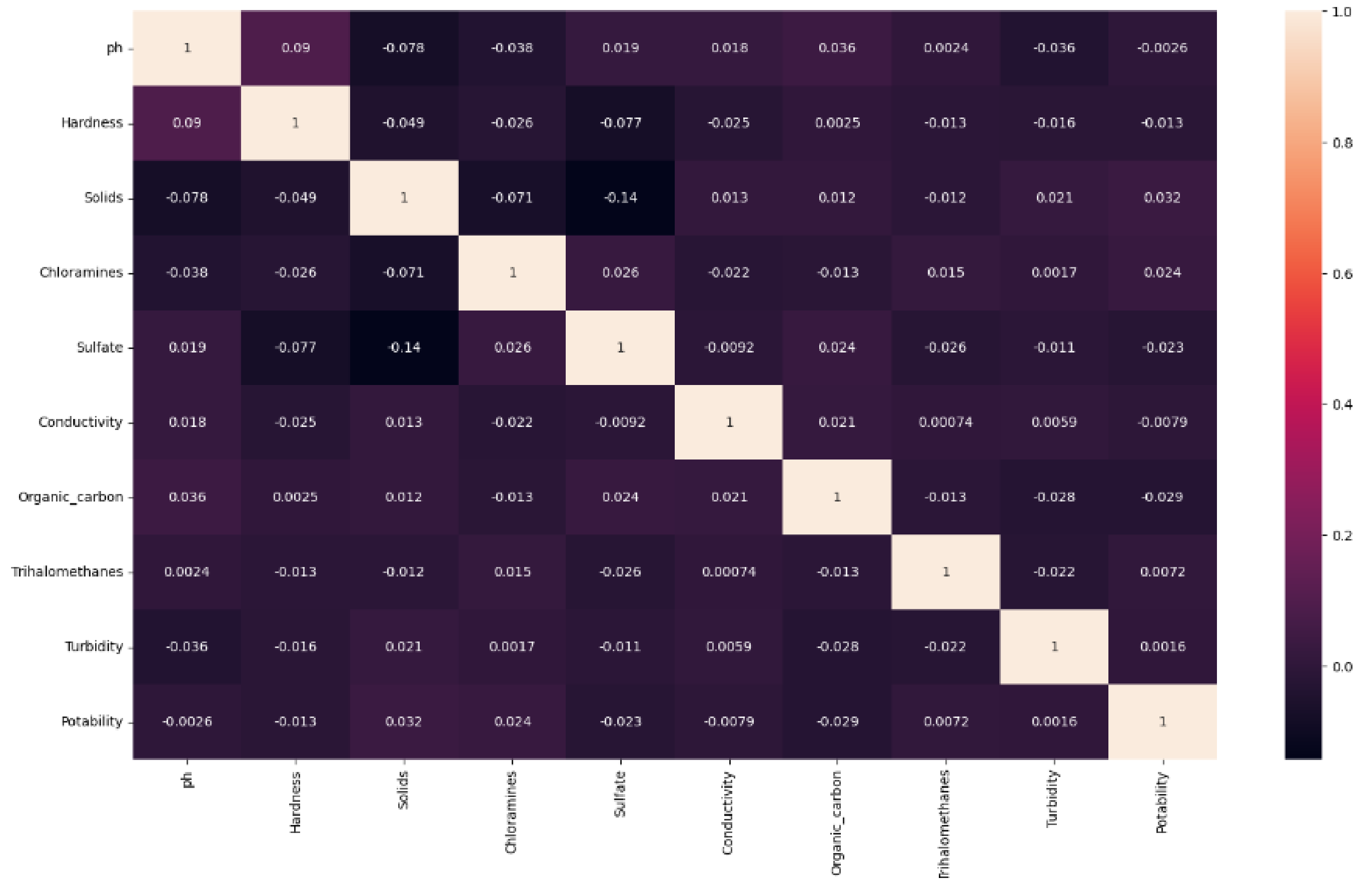
```
[13] phMean_0 = df[df['Potability'] == 0]['ph'].median(skipna=True)
     df.loc[(df['Potability'] == 0) & (df['ph'].isna()), 'ph'] = phMean_0
     phMean_1 = df[df['Potability'] == 1]['ph'].median(skipna=True)
     df.loc[(df['Potability'] == 1) & (df['ph'].isna()), 'ph'] = phMean_1

     ##################################### Imputing 'Sulfate' value #####################################

     SulfateMean_0 = df[df['Potability'] == 0]['Sulfate'].median(skipna=True)
     df.loc[(df['Potability'] == 0) & (df['Sulfate'].isna()), 'Sulfate'] = SulfateMean_0
     SulfateMean_1 = df[df['Potability'] == 1]['Sulfate'].median(skipna=True)
     df.loc[(df['Potability'] == 1) & (df['Sulfate'].isna()), 'Sulfate'] = SulfateMean_1

     ################################ Imputing 'Trihalomethanes' value #####################################

     TrihalomethanesMean_0 = df[df['Potability'] == 0]['Trihalomethanes'].median(skipna=True)
     df.loc[(df['Potability'] == 0) & (df['Trihalomethanes'].isna()), 'Trihalomethanes'] = TrihalomethanesMean_0
     TrihalomethanesMean_1 = df[df['Potability'] == 1]['Trihalomethanes'].median(skipna=True)
     df.loc[(df['Potability'] == 1) & (df['Trihalomethanes'].isna()), 'Trihalomethanes'] = TrihalomethanesMean_1
```

All the missing values in the columns are replaced with the median value of the same column i.e., replacing all occurrences of missing values (NA) within a variable by the median value has been done.

EXPLORATORY DATA ANALYSIS

```
plt.figure(figsize=(18,10))
sns.heatmap(df_cap.corr(),cbar=True,annot=True)
```

```
<Axes: >
```



	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
ph	1	0.09	-0.078	-0.038	0.019	0.018	0.036	0.0024	-0.036	-0.0026
Hardness	0.09	1	-0.049	-0.026	-0.077	-0.025	0.0025	-0.013	-0.016	-0.013
Solids	-0.078	-0.049	1	-0.071	-0.14	0.013	0.012	-0.012	0.021	0.032
Chloramines	-0.038	-0.026	-0.071	1	0.026	-0.022	-0.013	0.015	0.0017	0.024
Sulfate	0.019	-0.077	-0.14	0.026	1	-0.0092	0.024	-0.026	-0.011	-0.023
Conductivity	0.018	-0.025	0.013	-0.022	-0.0092	1	0.021	0.00074	0.0059	-0.0079
Organic_carbon	0.036	0.0025	0.012	-0.013	0.024	0.021	1	-0.013	-0.028	-0.029
Trihalomethanes	0.0024	-0.013	-0.012	0.015	-0.026	0.00074	-0.013	1	-0.022	0.0072
Turbidity	-0.036	-0.016	0.021	0.0017	-0.011	0.0059	-0.028	-0.022	1	0.0016
Potability	-0.0026	-0.013	0.032	0.024	-0.023	-0.0079	-0.029	0.0072	0.0016	1

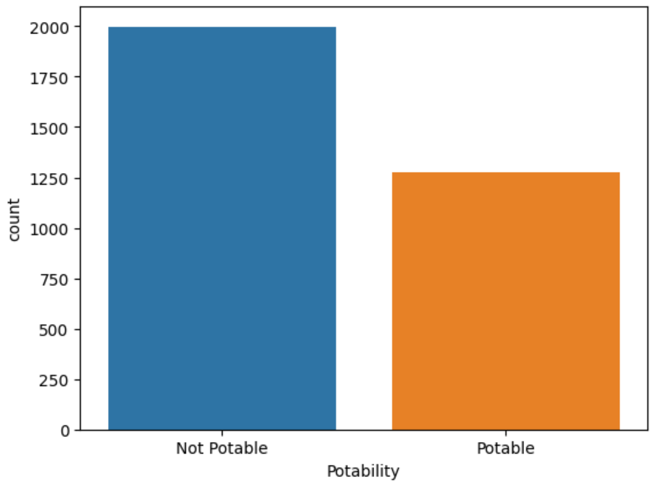
]

It is inferred from the heatmap that there is absence of multicollinearity which depicts that the input variables are not interdependent to get the result.

```
[15] #unstacking the correlation matrix to see the values more clearly
     corr = df_cap.corr()
     c1 = corr.abs().unstack()
     c1.sort_values(ascending=False)[12:24:2]
```

```
Hardness     ph             0.089525
Solids       ph             0.077905
Sulfate      Hardness       0.076747
Chloramines  Solids         0.070861
Hardness     Solids         0.048503
ph           Chloramines    0.037994
dtype: float64
```

```
#visualising the count of potable and non potable water bodies
ax = sns.countplot(x='Potability',data=df_cap,saturation=0.8)
plt.xticks(ticks=[0,1],labels=["Not Potable","Potable"])
plt.show()
```

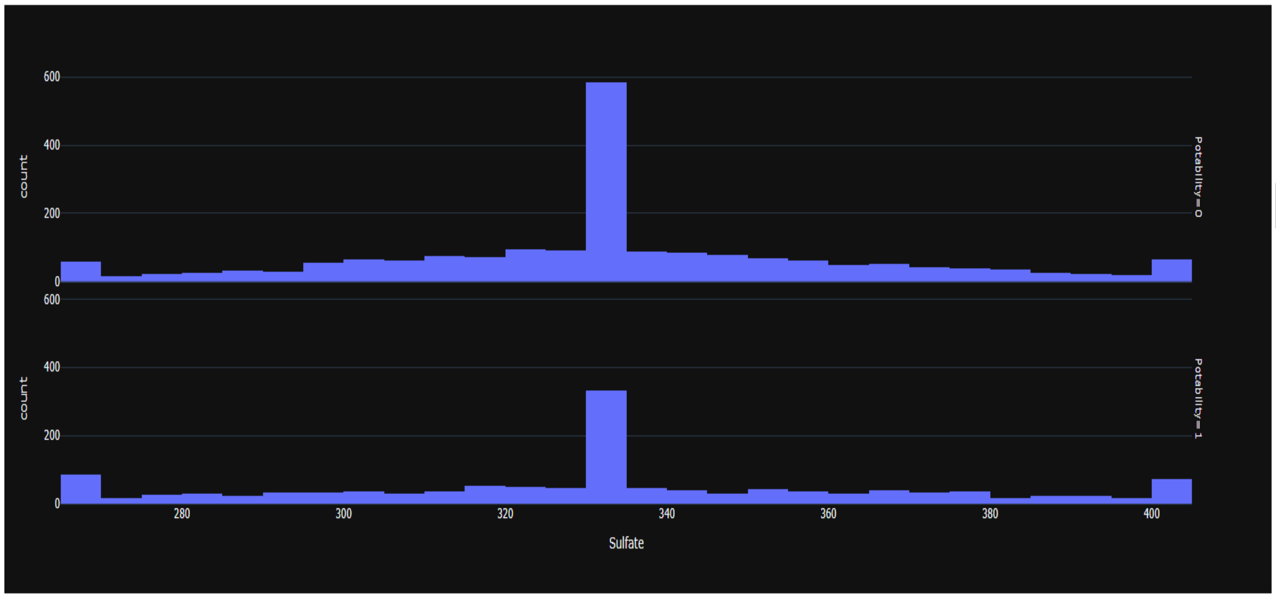


This plot depicts the no of values in each class of the output variable.

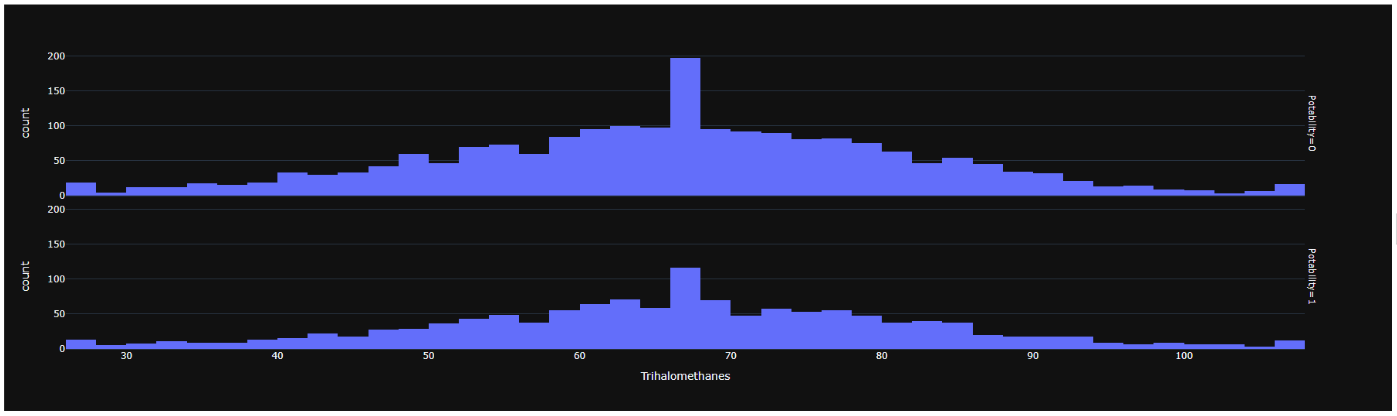
```
[17] #interpreting the results obtained from the above visualization
     x=df_cap.Potability.value_counts()
     label=[0,1]
     print(x)
```

```
0    1998
1    1278
Name: Potability, dtype: int64
```

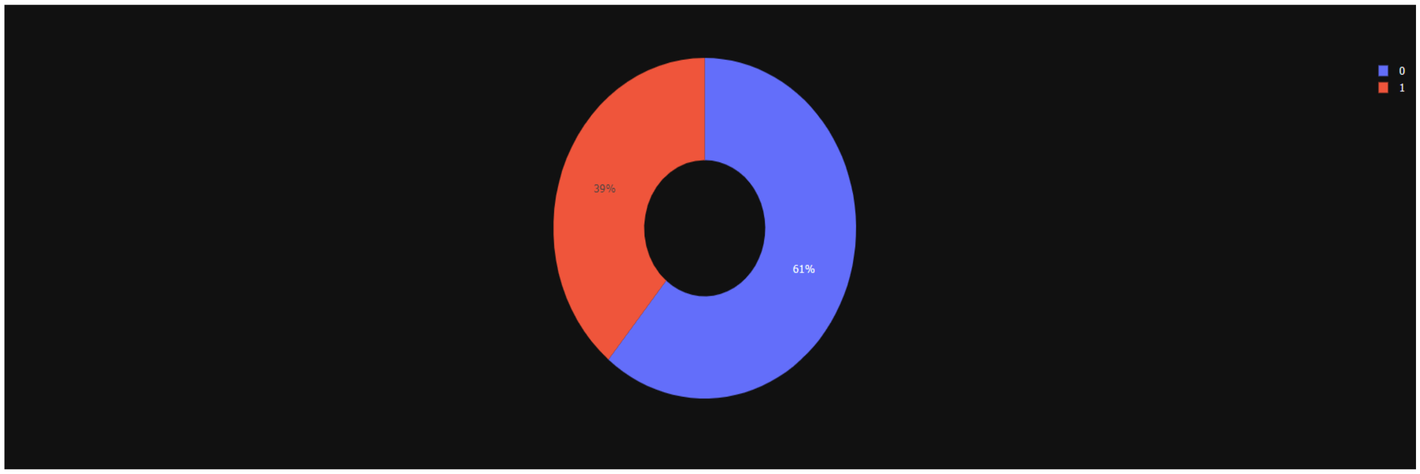
```
[18] fig = px.histogram(df_cap,x='Sulfate',facet_row='Potability',template='plotly_dark')
     fig.show()
```



```
fig = px.histogram(df_cap,x='Trihalomethanes',facet_row='Potability',template='plotly_dark')
fig.show()
```



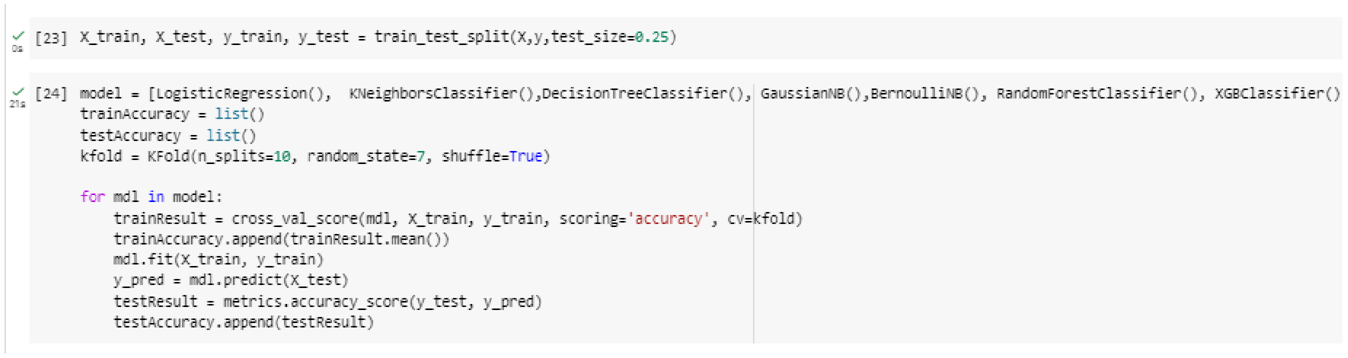
```
fig = px.pie(df_cap,names="Potability",hole=0.4,template="plotly_dark")
fig.show()
```



The above chart describes that 39% of the water bodies have potable water whereas 61% of water bodies have not potable water.

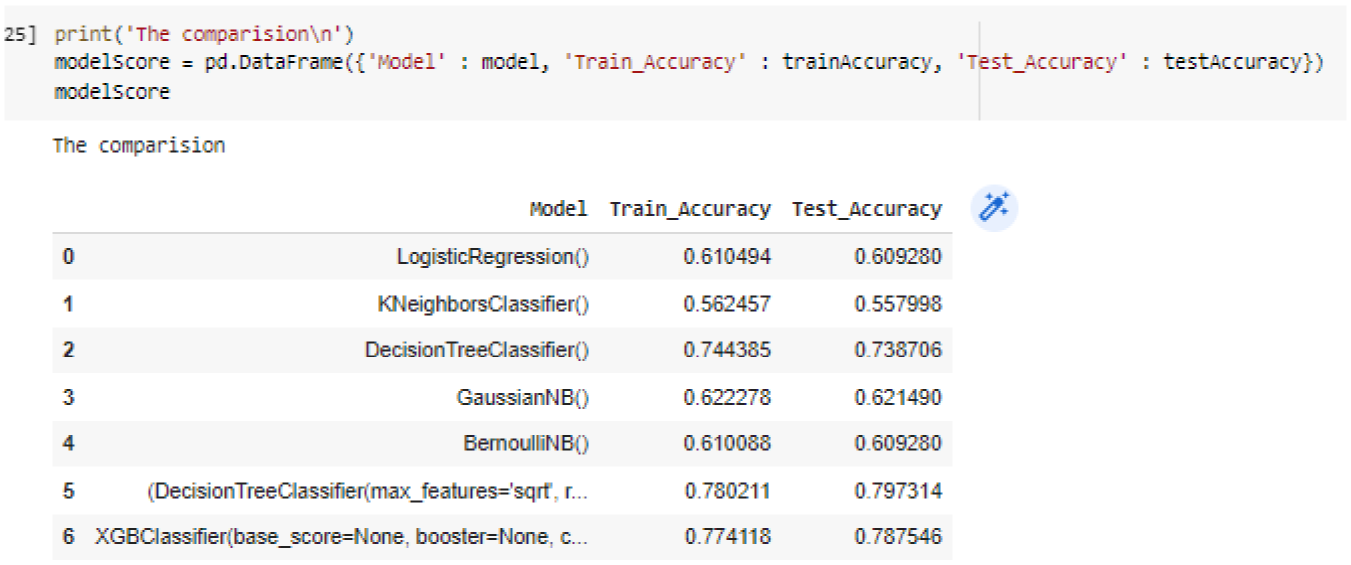
WITHOUT OVERSAMPLING BUILDING MODELS USING K FOLD CROSS VALIDATION

```
[22] import pandas as pd
     import numpy as np
     from sklearn.model_selection import KFold
     from sklearn.linear_model import LogisticRegression
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.naive_bayes import GaussianNB
     from sklearn.svm import SVC
     from sklearn.metrics import accuracy_score
     from xgboost import XGBClassifier
     from sklearn.naive_bayes import BernoulliNB
     from sklearn.neural_network import MLPClassifier
```



```
[23] X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25)
```

```
[24] model = [LogisticRegression(),  KNeighborsClassifier(),DecisionTreeClassifier(), GaussianNB(),BernoulliNB(), RandomForestClassifier(), XGBClassifier()
     trainAccuracy = list()
     testAccuracy = list()
     kfold = KFold(n_splits=10, random_state=7, shuffle=True)

     for mdl in model:
         trainResult = cross_val_score(mdl, X_train, y_train, scoring='accuracy', cv=kfold)
         trainAccuracy.append(trainResult.mean())
         mdl.fit(X_train, y_train)
         y_pred = mdl.predict(X_test)
         testResult = metrics.accuracy_score(y_test, y_pred)
         testAccuracy.append(testResult)
```

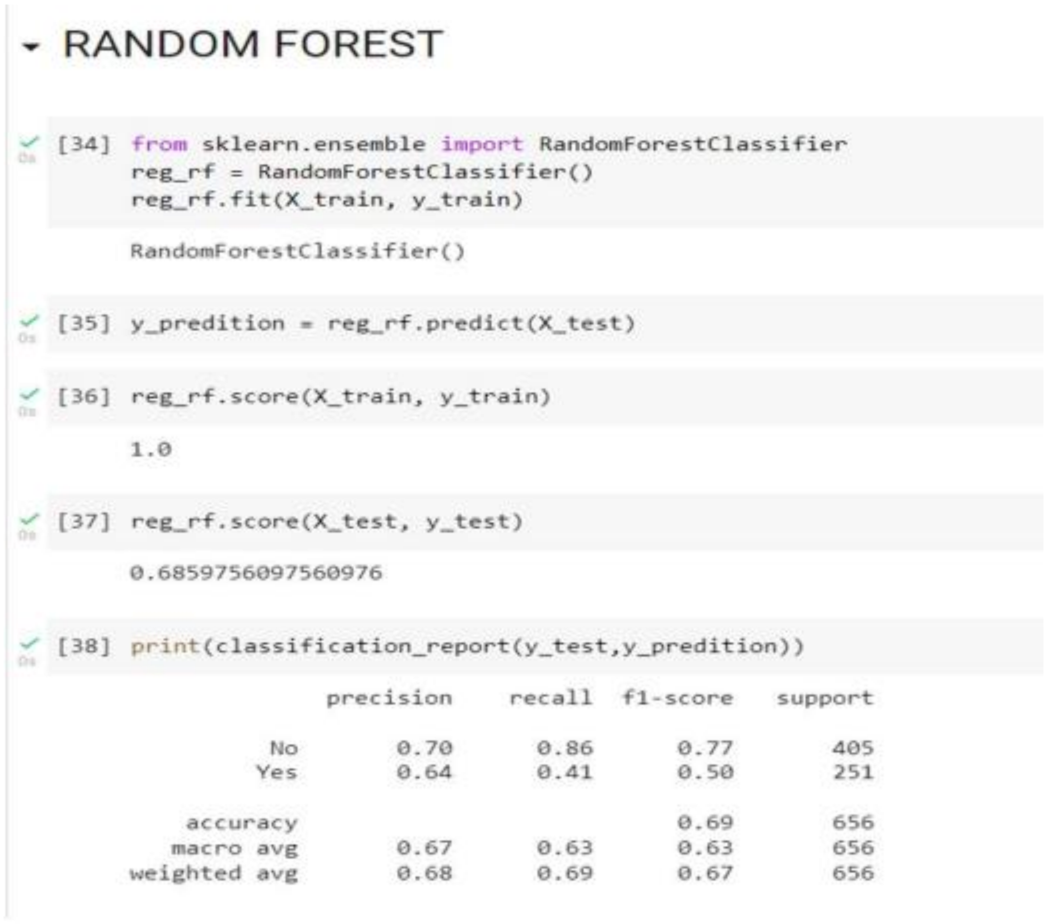


```
25] print('The comparision\n')
    modelScore = pd.DataFrame({'Model' : model, 'Train_Accuracy' : trainAccuracy, 'Test_Accuracy' : testAccuracy})
    modelScore
```

```
The comparision
```

	Model	Train_Accuracy	Test_Accuracy
0	LogisticRegression()	0.610494	0.609280
1	KNeighborsClassifier()	0.562457	0.557998
2	DecisionTreeClassifier()	0.744385	0.738706
3	GaussianNB()	0.622278	0.621490
4	BernoulliNB()	0.610088	0.609280
5	(DecisionTreeClassifier(max_features='sqrt', r...	0.780211	0.797314
6	XGBClassifier(base_score=None, booster=None, c...	0.774118	0.787546

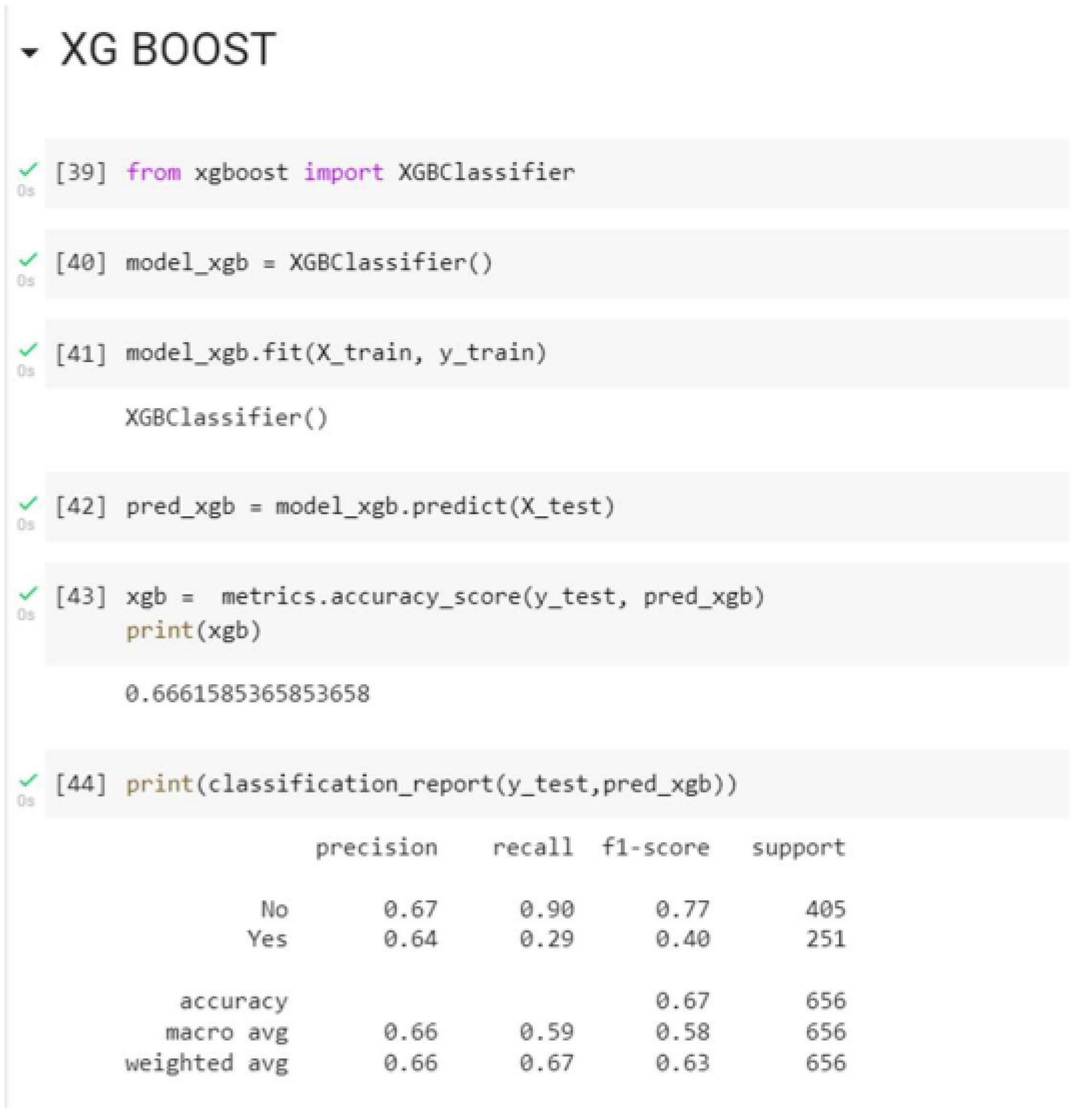
We can infer from the above obtained results that the test accuracy obtained is higher than train accuracy . Random Forest and XGBClassifier have shown better accuracy of about 79.7% and 78.7% respectively.



RANDOM FOREST

```
[34] from sklearn.ensemble import RandomForestClassifier
     reg_rf = RandomForestClassifier()
     reg_rf.fit(X_train, y_train)

     RandomForestClassifier()

[35] y_predition = reg_rf.predict(X_test)

[36] reg_rf.score(X_train, y_train)

     1.0

[37] reg_rf.score(X_test, y_test)

     0.6859756097560976

[38] print(classification_report(y_test,y_predition))

                   precision    recall  f1-score   support

              No       0.70      0.86      0.77       405
             Yes       0.64      0.41      0.50       251

        accuracy                           0.69       656
       macro avg       0.67      0.63      0.63       656
    weighted avg       0.68      0.69      0.67       656
```

The Random Forest approach is a supervised machine learning algorithm that grows through implementation and combines multiple decision trees to create a section of what is called a 'forest'. Random forest is an extension of bagging that also randomly selects subsets of features used in each data sample.



XG BOOST

```
[39] from xgboost import XGBClassifier
```

```
[40] model_xgb = XGBClassifier()
```

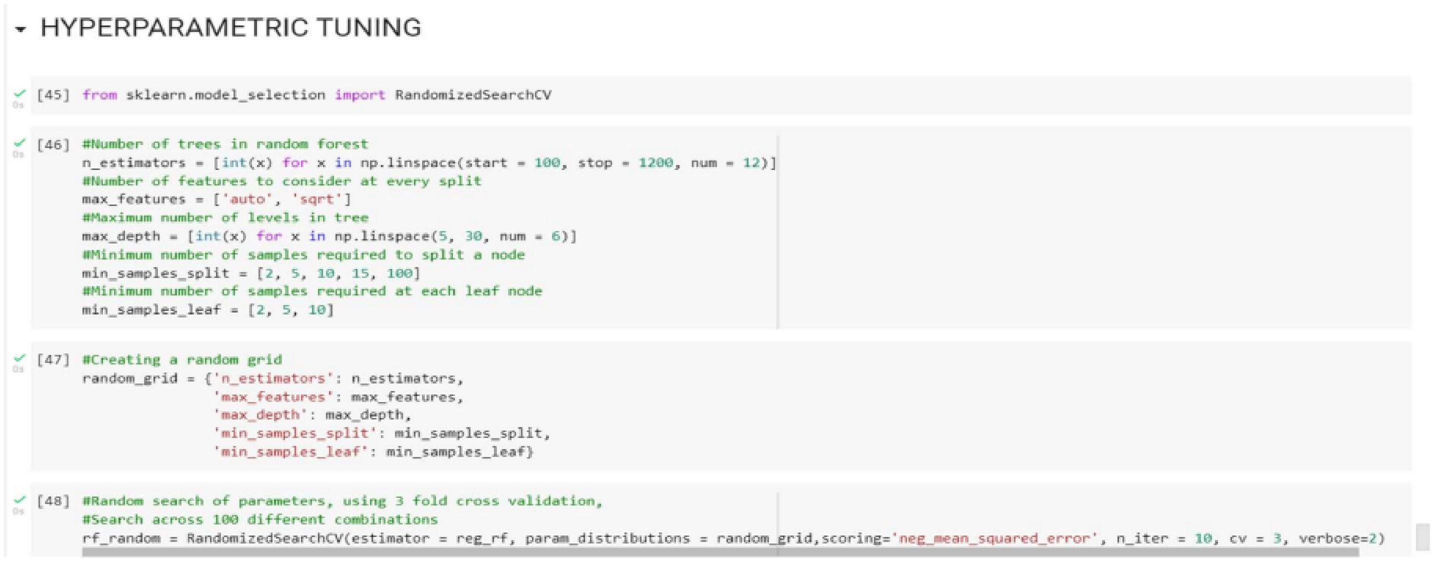
```
[41] model_xgb.fit(X_train, y_train)

     XGBClassifier()
```

```
[42] pred_xgb = model_xgb.predict(X_test)
```

```
[43] xgb =  metrics.accuracy_score(y_test, pred_xgb)
     print(xgb)

     0.6661585365853658
```

```
[44] print(classification_report(y_test,pred_xgb))

                   precision    recall  f1-score   support

              No       0.67      0.90      0.77       405
             Yes       0.64      0.29      0.40       251

        accuracy                           0.67       656
       macro avg       0.66      0.59      0.58       656
    weighted avg       0.66      0.67      0.63       656
```

The XGBoost is a supervised learning algorithm and is an approach of efficient implementation of the gradient boost tree algorithm. It tries to predict the target variable by combining an ensemble of the estimates from weak learners.



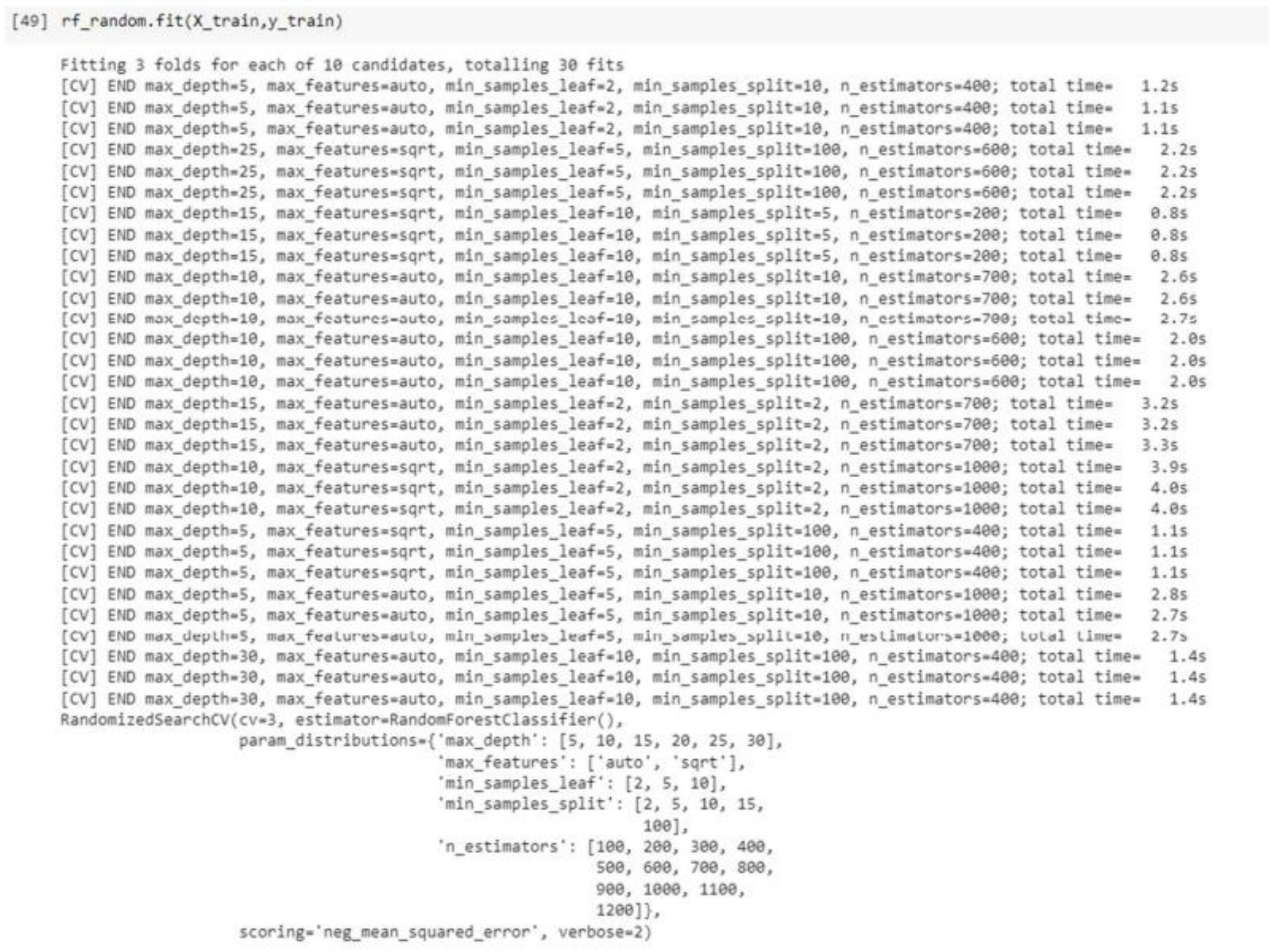
HYPERPARAMETRIC TUNING

```
[45] from sklearn.model_selection import RandomizedSearchCV
```

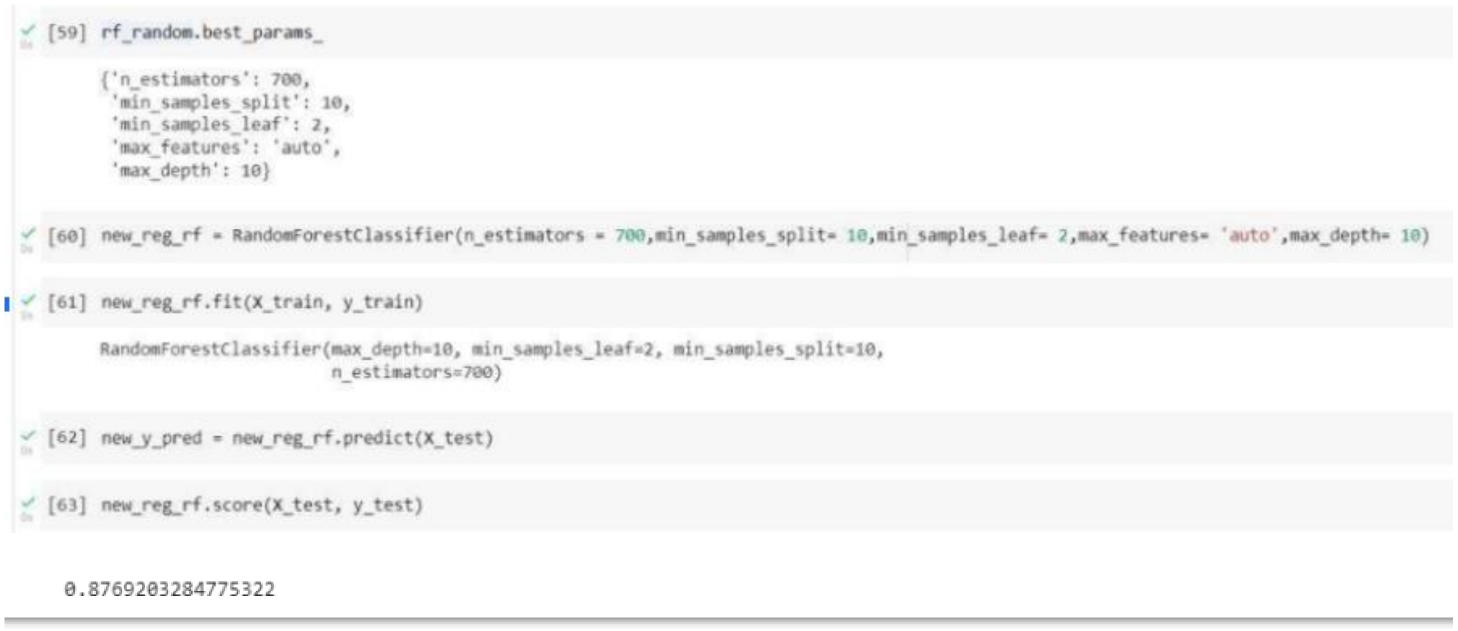
```
[46] #Number of trees in random forest
     n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
     #Number of features to consider at every split
     max_features = ['auto', 'sqrt']
     #Maximum number of levels in tree
     max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
     #Minimum number of samples required to split a node
     min_samples_split = [2, 5, 10, 15, 100]
     #Minimum number of samples required at each leaf node
     min_samples_leaf = [2, 5, 10]
```

```
[47] #Creating a random grid
     random_grid = {'n_estimators': n_estimators,
                    'max_features': max_features,
                    'max_depth': max_depth,
                    'min_samples_split': min_samples_split,
                    'min_samples_leaf': min_samples_leaf}
```

```
[48] #Random search of parameters, using 3 fold cross validation,
     #Search across 100 different combinations
     rf_random = RandomizedSearchCV(estimator = reg_rf, param_distributions = random_grid,scoring='neg_mean_squared_error', n_iter = 10, cv = 3, verbose=2)
```



```
[49] rf_random.fit(X_train,y_train)

     Fitting 3 folds for each of 10 candidates, totalling 30 fits
     [CV] END max_depth=5, max_features=auto, min_samples_leaf=2, min_samples_split=10, n_estimators=400; total time=   1.2s
     [CV] END max_depth=5, max_features=auto, min_samples_leaf=2, min_samples_split=10, n_estimators=400; total time=   1.1s
     [CV] END max_depth=5, max_features=auto, min_samples_leaf=2, min_samples_split=10, n_estimators=400; total time=   1.1s
     [CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5, min_samples_split=100, n_estimators=600; total time=   2.2s
     [CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5, min_samples_split=100, n_estimators=600; total time=   2.2s
     [CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5, min_samples_split=100, n_estimators=600; total time=   2.2s
     [CV] END max_depth=15, max_features=sqrt, min_samples_leaf=10, min_samples_split=5, n_estimators=200; total time=   0.8s
     [CV] END max_depth=15, max_features=sqrt, min_samples_leaf=10, min_samples_split=5, n_estimators=200; total time=   0.8s
     [CV] END max_depth=15, max_features=sqrt, min_samples_leaf=10, min_samples_split=5, n_estimators=200; total time=   0.8s
     [CV] END max_depth=10, max_features=auto, min_samples_leaf=10, min_samples_split=10, n_estimators=700; total time=   2.6s
     [CV] END max_depth=10, max_features=auto, min_samples_leaf=10, min_samples_split=10, n_estimators=700; total time=   2.6s
     [CV] END max_depth=10, max_features=auto, min_samples_leaf=10, min_samples_split=10, n_estimators=700; total time=   2.7s
     [CV] END max_depth=10, max_features=auto, min_samples_leaf=10, min_samples_split=100, n_estimators=600; total time=   2.0s
     [CV] END max_depth=10, max_features=auto, min_samples_leaf=10, min_samples_split=100, n_estimators=600; total time=   2.0s
     [CV] END max_depth=10, max_features=auto, min_samples_leaf=10, min_samples_split=100, n_estimators=600; total time=   2.0s
     [CV] END max_depth=15, max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=700; total time=   3.2s
     [CV] END max_depth=15, max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=700; total time=   3.2s
     [CV] END max_depth=15, max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=700; total time=   3.3s
     [CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=1000; total time=   3.9s
     [CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=1000; total time=   4.0s
     [CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=1000; total time=   4.0s
     [CV] END max_depth=5, max_features=sqrt, min_samples_leaf=5, min_samples_split=100, n_estimators=400; total time=   1.1s
     [CV] END max_depth=5, max_features=sqrt, min_samples_leaf=5, min_samples_split=100, n_estimators=400; total time=   1.1s
     [CV] END max_depth=5, max_features=sqrt, min_samples_leaf=5, min_samples_split=100, n_estimators=400; total time=   1.1s
     [CV] END max_depth=5, max_features=auto, min_samples_leaf=5, min_samples_split=10, n_estimators=1000; total time=   2.8s
     [CV] END max_depth=5, max_features=auto, min_samples_leaf=5, min_samples_split=10, n_estimators=1000; total time=   2.7s
     [CV] END max_depth=5, max_features=auto, min_samples_leaf=5, min_samples_split=10, n_estimators=1000; total time=   2.7s
     [CV] END max_depth=30, max_features=auto, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   1.4s
     [CV] END max_depth=30, max_features=auto, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   1.4s
     [CV] END max_depth=30, max_features=auto, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   1.4s
     RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(),
                        param_distributions={'max_depth': [5, 10, 15, 20, 25, 30],
                                             'max_features': ['auto', 'sqrt'],
                                             'min_samples_leaf': [2, 5, 10],
                                             'min_samples_split': [2, 5, 10, 15,
                                                                   100],
                                             'n_estimators': [100, 200, 300, 400,
                                                              500, 600, 700, 800,
                                                              900, 1000, 1100,
                                                              1200]},
                        scoring='neg_mean_squared_error', verbose=2)
```

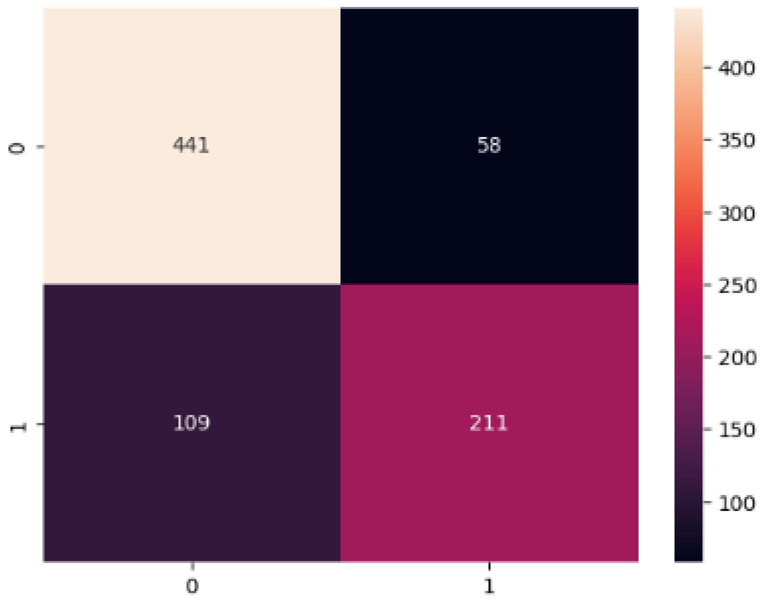


```
[59] rf_random.best_params_

     {'n_estimators': 700,
      'min_samples_split': 10,
      'min_samples_leaf': 2,
      'max_features': 'auto',
      'max_depth': 10}

[60] new_reg_rf = RandomForestClassifier(n_estimators = 700,min_samples_split= 10,min_samples_leaf= 2,max_features= 'auto',max_depth= 10)

[61] new_reg_rf.fit(X_train, y_train)

     RandomForestClassifier(max_depth=10, min_samples_leaf=2, min_samples_split=10,
                            n_estimators=700)

[62] new_y_pred = new_reg_rf.predict(X_test)

[63] new_reg_rf.score(X_test, y_test)

  0.8769203284775322
```

The process of hyperparameter tuning entails determining a set of best parameters values for the model to improve the performance measures. This method uses random sampling from a grid of hyperparameters rather than a thorough search. The best parameters are generated after the iterations and fed into the model. With these parameters, the model was trained providing an accuracy of 0.876.

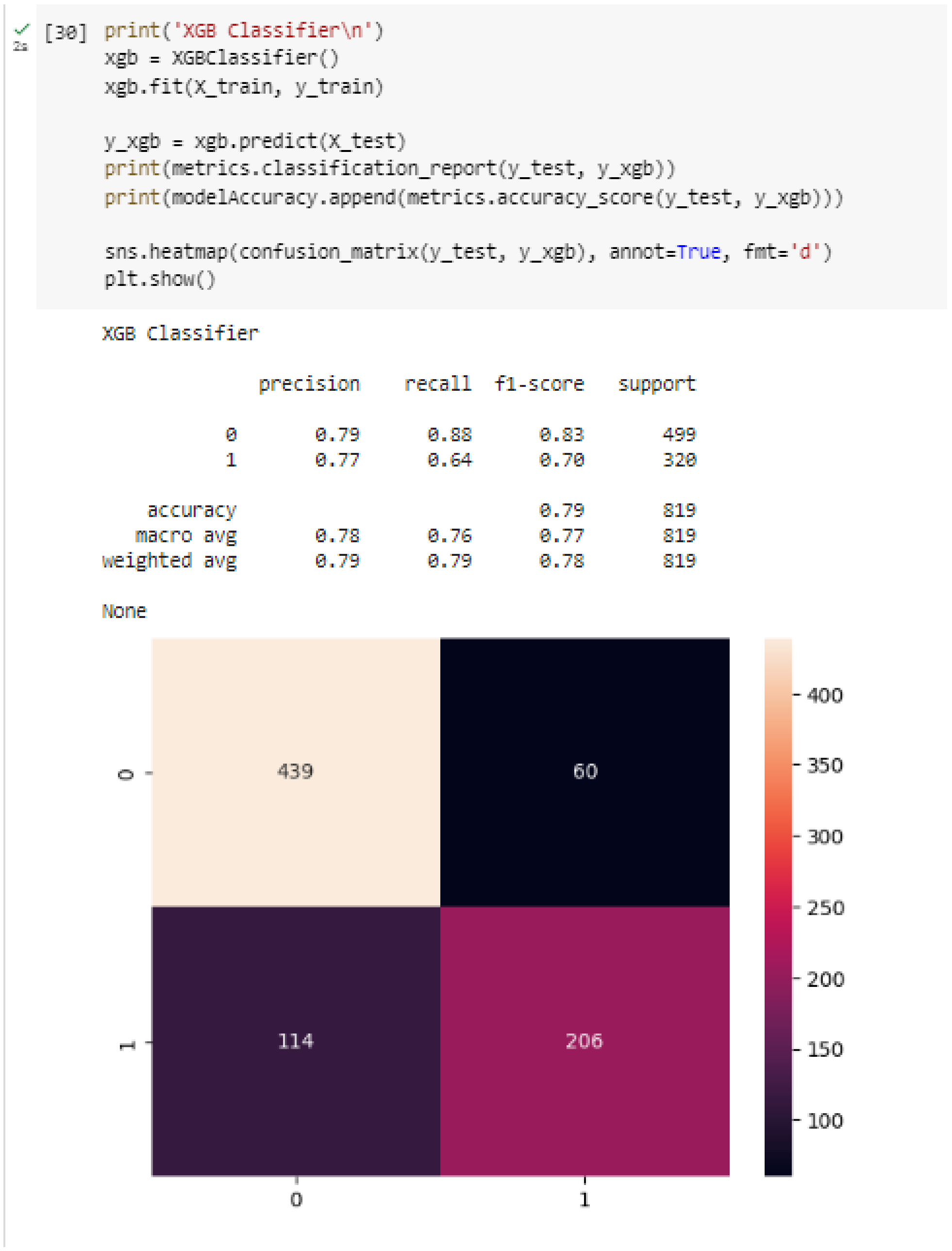
```
[29] print('Random Forest Classifier\n')
     Rfc = RandomForestClassifier()
     Rfc.fit(X_train, y_train)

     y_Rfc = Rfc.predict(X_test)
     print(metrics.classification_report(y_test, y_Rfc))
     print(modelAccuracy.append(metrics.accuracy_score(y_test, y_Rfc)))

     sns.heatmap(confusion_matrix(y_test, y_Rfc), annot=True, fmt='d')
     plt.show()
```

```
Random Forest Classifier

              precision    recall  f1-score   support

           0       0.80      0.88      0.84       499
           1       0.78      0.66      0.72       320

    accuracy                           0.80       819
   macro avg       0.79      0.77      0.78       819
weighted avg       0.80      0.80      0.79       819

None
```



Inference:

From the above confusion matrix it can be inferred that 441 values are correctly predicted as not potable water bodies and 211 values are correctly predicted as potable water bodies. 109 values are incorrectly predicted as not potable water bodies whereas 58 values are incorrectly predicted as potable.



```
[30] print('XGB Classifier\n')
     xgb = XGBClassifier()
     xgb.fit(X_train, y_train)

     y_xgb = xgb.predict(X_test)
     print(metrics.classification_report(y_test, y_xgb))
     print(modelAccuracy.append(metrics.accuracy_score(y_test, y_xgb)))

     sns.heatmap(confusion_matrix(y_test, y_xgb), annot=True, fmt='d')
     plt.show()
```

```
XGB Classifier

              precision    recall  f1-score   support

           0       0.79      0.88      0.83       499
           1       0.77      0.64      0.70       320

    accuracy                           0.79       819
   macro avg       0.78      0.76      0.77       819
weighted avg       0.79      0.79      0.78       819

None
```

Inference:

From the above confusion matrix it can be inferred that 439 values are correctly predicted as not potable water bodies and 206 values are correctly predicted as potable water bodies. 114 values are incorrectly predicted as not potable water bodies whereas 60 values are incorrectly predicted as potable.

RESULT AND DISCUSSION

The machine learning model is evaluated in terms of how well it has performed. This is achieved by computing the performance metrics of the algorithms. The performance metrics that are used here, to evaluate the classification model performance are as follows:

RANDOM FOREST
Accuracy obtained on test data was 0.797314.

From the classification report above for Random Forest it can be inferred that:

- Precision was 0.80 and 0.78 for Not potable and Potable, respectively. The model classified as No, 80% of them really were not potable; and the model classified as Yes, only 78% of them really were potable.
- Recall was 0.88 and 0.66 for No and Yes, respectively. In other words, it correctly identifies 88% as not potable and 66% as potable.
- F1-score was 0.84 and 0.72 for No and Yes, respectively which shows that the model has predicted the observations of class No correctly better than class Yes.

XG BOOST
Accuracy obtained on test data was 0.787546.

From the classification report above for XG Boost it can be inferred that:

- Precision was 0.79 and 0.77 for No and Yes, respectively. The model classified as No, 79% of them really were not potable; and the model classified as Yes, only 77% of them really were potable.
- Recall was 0.88 and 0.64 for No and Yes, respectively. In other words, it correctly identifies 88% as not potable and 64% as potable.
- F1-score was 0.83 and 0.70 for No and Yes, respectively which shows that the model has predicted the observations of class No correctly better than class Yes.

CONCLUSION AND FUTURE WORK

It is very necessary to understand the need to analyze the quality of the water bodies to conserve the environment. In this approach, the performance of the machine learning models such as Random Forest, XGBoost, Decision Tree, Logistic Regression, KNN Regression, Gaussian and Bernoulli Naive Bayes were evaluated to predict the water quality. The missing values were filled through the median imputation and the outliers were capped to avoid the loss of dimensions.

The results have shown that the Random Forest has given an accuracy of 0.797314 and XG Boost has given an accuracy of 0.787546 which is the highest accuracy provided among the ML algorithms applied. Hyperparametric tuning has given an accuracy of 0.684 and the best performance was seen in the Random Forest algorithm which presented an accuracy of 0.69. The comparison of the applied models has shown that the Random Forest and the Hyperparametric tuning models were more reliable throughout the entire project.

The future suggestion to enhance the performance of the prediction is to implement Multi-layer perceptron (MLP). This model uses the proximity of the data points to make predictions about the individual data points, thus allowing for a better scope of accuracy. It can provide better graphical representation and predict an efficient Water Quality Index (WQI) of the water bodies with the parameters in the dataset.

REFERENCES

[1] Aldhyani TH, Al-Yaari M, Alkahtani H, Maashi M. Water quality prediction using artificial intelligence algorithms. Applied Bionics and Biomechanics. 2020 Dec 29;2020.

[2] Khan MS, Islam N, Uddin J, Islam S, Nasir MK. Water quality prediction and classification based on principal component regression and gradient boosting classifier approach. Journal of King Saud University-Computer and Information Sciences. 2022 Sep 1;34(8):4773-81.

[3] Ahmed U, Mumtaz R, Anwar H, Shah AA, Irfan R, García-Nieto J. Efficient water quality prediction using supervised machine learning. Water. 2019 Oct 24;11(11):2210.

[4] Liu Y, Liang Y, Ouyang K, Liu S, Rosenblum DS, Zheng Y. Predicting urban water quality with ubiquitous data-a data-driven approach. IEEE Transactions on Big Data. 2020 Feb 10;8(2):564-78.

[5]Haghiabi AH, Nasrolahi AH, Parsaie A. Water quality prediction using machine learning methods. Water Quality Research Journal. 2018 Feb;53(1):3-13.

[6]Zhou J, Wang Y, Xiao F, Wang Y, Sun L. Water Quality Prediction Method Based on IGRA and LSTM. *Water*. 2018; 10(9):1148. https://doi.org/10.3390/w10091148