Water quality prediction

NAME: Yashaswini L
ROLL NO: 20ETCS002159
SEM/YEAR: 6th/2023
DEPARTMENT: CSE
B Section

NAME: AKSHATA
ROLL NO: *20ETCS002010*
SEM/YEAR: *6Th/2023*
DEPARTMENT: CSE
B Section

NAME: HANNAH CHRISTINA G
ROLL NO: *20ETCS002056*
SEM/YEAR: 6th/2023
DEPARTMENT: CSE
B Section

NAME: J RANI
ROLL NO: *20ETCS002058*
SEM/YEAR: *6TH/2023*
DEPARTMENT: CSE
B Section

***Abstract*—** Aquaculture, ecosystem sustainability, and environmental monitoring all benefit from accurate water quality predictions. Conventional forecasting techniques can't account for the nonlinear and Well water quality non-stationarity. Artificial neural networks (ANNs) have experienced significant improvement in recent years, making them a hotspot for predicting water quality**.** We have looked at and analyzed ANN-based water quality prediction in depth from three angles: specifically, hybrid, recurrent, and feedforward architectures. A total of 23 different types of water quality characteristics were identified based on 151 studies published between 2008 and 2019. The sensor was used to collect the variables most effectively, followed by specialized experimental tools such as a UV-visible photometer. The following is a summary of five distinct output strategies: Univariate-Input-Itself-Output, Univariate-Input-Other-Output, Multivariate-Input-Other(multi)-Output, and Multivariate-Input-Itself-Other (multi)-Output. The review's findings indicate that ANN models may successfully handle a variety of modeling issues in rivers, lakes, reservoirs, wastewater treatment facilities (WWTPs), groundwater, ponds, and streams. Researchers in prediction and related subjects can benefit from the findings of many of the review papers. Future development will benefit from the improved modeling quality of a number of novel designs, including recurrent and hybrid structures, which were described in the study.

Keywords— ANNs; feedforward; recurrent; hybrid; water quality prediction

1. Introduction

Water is the most important of sources, vital for sustaining all kinds of life; however, it is in constant threat of pollution by life itself. Water is one of the most communicable mediums with a far reach. Rapid industrialization has consequently led to deterioration of water quality at an alarming rate. Poor water quality results have been known to be one of the major factors of escalation of harrowing diseases. As reported, in developing countries, 80% of the diseases are water borne diseases, which have led to 5 million deaths and 2.5 billion illnesses [1]. The most common of these diseases in Pakistan are diarrhea, typhoid, gastroenteritis, cryptosporidium infections, some forms of hepatitis and giardiasis intestinal worms [2]. In Pakistan, water borne diseases, cause a GDP loss of 0.6–1.44% every year [3]. This makes it a pressing problem, particularly in a developing country like Pakistan. Water quality is currently estimated through expensive and time-consuming lab and statistical analyses, which require sample collection, transport to labs, and a considerable amount of time andcalculation, which is quite ineffective given water is quite a communicable medium and time is of the essence if water is polluted with disease-inducing waste [4].
This study's primary aim is to suggest and assess a different approach based on supervised machine learning for the accurate real-time prediction of water quality.
The dataset for Rawal Watershed, located in Pakistan, is used in this study.

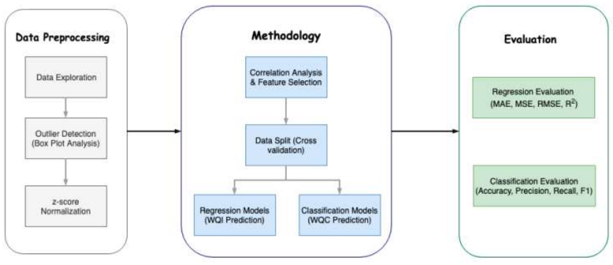
by The Pakistan Council of Research in Water Resources (PCRWR), which is accessible online at the URL http://www.pcrwr.gov.pk. Using the aforementioned dataset, a selection of supervised machine learning methods were used to forecast the water quality index (WQI) and water quality class (WQC)

The main contributions of this study are summarized as follows:
• The available data underwent a preliminary analysis to clean, normalize, and perform feature selection on the water quality measurements in order to find the smallest relevant subset that allows for high precision at a reasonable price. In subsequent investigations of a similar nature, costly and time-consuming lab analysis using particular sensors can be avoided.
• The dataset was used to test a number of sample-supervised prediction (classification and regression) methods. The entire approach is put forward in the context of a numerical examination of water quality.
• Gradient boosting and polynomial regression perform the best WQI prediction, with mean absolute errors (MAEs) of 1.9642 and 2.7273, respectively, according to the findings of extensive testing, while multi-layer perceptron (MLP)

performs the best WQC classification, with an accuracy of 0.8507.
The remainder of this paper is organized as follows: Section 2 provides a literature review in this domain. In Section 3, we explore the dataset and perform preprocessing. In Section 4, we employ various machine learning methodologies to predict water quality using minimal parameters and discuss the results of regression and classification algorithms, in terms of error rates and classification precision. In Section 5, we discuss the implications and novelty of our study and finally in Section 6, we conclude the paper and provide future lines of work.

2. RESEARCH REVIEW

This study examines the approaches that have been used to address issues with water quality. In research, traditional lab analysis and statistical analysis are frequently employed to help determine the quality of the water, while other analyses use machine learning approaches to help identify the best possible solution to the water quality problem.
Local research employing lab analysis helped us gain a greater insight into the water quality problem in Pakistan. In one such research study, Daud et al. [5] gathered water samples from different areas of Pakistan and tested them against different parameters using a manual lab analysis and found a high presence of E. coli and fecal coliform due to industrial and sewerage waste. Alamgir et al. [6]
Using manual lab analysis, 46 distinct samples from Orangi town, Karachi, were examined. Sulphates and the total fecal coliform count were determined to be high.
After getting familiar with the water quality research concerning Pakistan, we explored research employing machine learning methodologies in the realm of water quality. When it comes to estimating water quality using machine learning, Shafi et al. [7] estimated water quality using classical machine learning algorithms namely, Support Vector Machines (SVM), Neural Networks (NN), Deep Neural Networks (Deep NN) and k Nearest Neighbors (kNN), with the highest accuracy of 93% with Deep NN. The estimated water quality in their work is based on only three parameters: turbidity, temperature and pH, which are tested according to World Health Organization (WHO) standards Using only three parameters and comparing them to standardized values is quite a limitation when predicting water quality. Ahmad et al. [8] employed single feed forward neural networks and a combination of multiple neural networks to estimate the WQI. They used 25 water quality parameters as the input. Using a combination of backward elimination and forward selection selective combination methods, they achieved an R2 and MSE of 0.9270, 0.9390 and 0.1200, 0.1158, respectively. The use of 25 parameters makes their solution a little immoderate in terms of an inexpensive real time system, given the price of the parameter sensors. Sakizadeh [9] predicted the WQI using 16 water quality parameters and ANN with Bayesian regularization. His study yielded correlation coefficients between the observed and predicted values of 0.94 and 0.77, respectively. Abyaneh [10] predicted the chemical oxygen demand (COD) and the biochemical oxygen demand (BOD) using two conventional machine learning methodologies namely, ANN and multivariate linear regression. They used four parameters, namely pH, temperature, total suspended solids (TSS) and total suspended (TS) to predict the COD and BOD. Ali and Qamar [11] used the unsupervised technique of the average linkage (within groups) method of hierarchical clustering to classify samples into water quality classes. However, they ignored the major parameters associated with WQI during the learning process and they did not use any standardized water quality index to evaluate their predictions. Gazzaz et al. [4] used ANN to predict the WQI with a model explaining almost 99.5% of variation in the data. They used 23 parameters to predict the WQI, which turns out to be quite expensive if one is to use it for an IoT system, given the prices of the sensors. Rankovic et al. [12] predicted the dissolved oxygen (DO) using a feedforward neural network (FNN). They used 10 parameters to predict the DO, which again defeats the purpose if it has to be used for a real-time WQI estimation with an IoT system
The majority of studies either utilised manual lab analysis, did not estimate the water quality index standard, or used too many factors to be sufficiently effective. Figure 1 shows the approach currently used and how the suggested methodology improves on these ideas.


Data Preprocessing
Data Exploration
Outlier Detection (Box Plot Analysis)
z-score Normalization
Methodology
Correlation Analysis & Feature Selection
Data Split (Cross validation)
Regression Models (WQI Prediction)
Classification Models (WQC Prediction)
Evaluation
Regression Evaluation (MAE, MSE, RMSE, R^2)
Classification Evaluation (Accuracy, Precision, Recall, F1)
Figure 1. Methodology flow.

3. DATA PREPROCESSING

The PCRWR provided the data for this study, which was then cleaned by doing the box plot analysis that is covered in this section. Following data cleaning, they were standardised using q-value normalisation to change their range to 0-100 so that the WQI could be calculated using the six accessible parameters. The initial values were normalised using a z-score once the WQI was determined so they were all on the same scale. The entire process is then described.

3.1 Data collection.

663 samples from 13 distinct Rawal Water Lake sources were included in the dataset obtained from PCRWR, which was gathered from 2009 to 2012. It included the 12 parameters given in Table 1 and 51 samples from each source.

Table 1. Parameters along with their "WHO" standard limits [11].

Parameter	WHO Limits
Alkalinity	500 mg/L
Appearance	Clear
Calcium	200 mg/L
Chlorides	200 mg/L
Conductance	2000 µS/cm
Fecal Coliforms	Nil Colonies/100 mL
Hardness as $CaCO_3$	500 mg/L
Nitrite as NO_2^-	<1 mg/L
pH	6.5–8.5
Temperature	°C
Total Dissolved Solids	1000 mg/L
Turbidity	5 NTU

3.2. Boxplot Analysis and Outlier Detection

We chose boxplot analysis for outlier detection because most of the parameters varied enough and were on the higher end of the values, and a boxplot provides insightful visualization to decide outlier detection threshold values depending upon the problem domain. Boxplot analysis showed that most parameters lied outside the box, deeming outliers normal, so we adopted an upper-cap strategy to filter out outliers. We recognized the parameter values that were very different from other values and replaced them with the max threshold value. We set the max threshold value as the parameter value that was just below the outlier values. For example, for turbidity, as reflected in Figure 2, we set the threshold value as the sample value, which was 753, and applied it to all values above 753, so that all the values that lied above 753 were assigned the value of 753. We repeated the same process with all the parameters and manually removed the outliers such as to not risk any data loss at all, given our limited dataset [4]. In addition, we were extremely lenient while choosing the upper threshold of parameters so as not to bias the dataset and just to loosely penalize the values that seemed way out of limits and unlikely to occur.

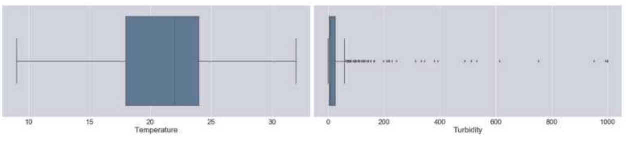


Figure 2. Outlier detection using box plot analysis.

3.3. water quality index (WQI)

Water quality index (WQI) is a single metric that shows water quality and is derived utilising numerous criteria that are actually indicative of water quality. Traditionally, nine water quality parameters are used to construct the WQI; however, if we did not know all of them, we could still estimate the water quality index using at least six specified parameters. In our dataset, we had five parameters: faecal coliform, pH, temperature, turbidity, and total dissolved solids. We also considered nitrites as the sixth component since numerous WQI research [13-15] claimed that the weight and relative value of nitrites in the WQI calculation is equivalent to that of nitrates. We estimated the WQI using parameters and their assigned weightages, we calculated the WQI of each sample as reflected in Equation (1), where qvalue reflects the value of a parameter in the range of 0-100 and w_factor represents the weight of a particular parameter as listed in Table 2. WQI is fundamentally calculated by initially multiplying the q value of each parameter by its corresponding weight, adding them all up, and then dividing the result by the sum of the weights of the employed parameters [14,15].

$$\text{WQI} = \frac{\sum q_{value} \times w\_factor}{\sum w\_factor}$$

Table 2. Parameters weights for the WQI calculation [14,15].

Weighing Factor	Weight
pH	0.11
Temperature	0.10
Turbidity	0.08
Total Dissolved Values	0.07
Nitrates	0.10
Fecal Coliform	0.16

3.4. water quality class

Once we had estimated the WQI, we defined the water quality class (WQC) of each sample using the WQI in classification algorithms [14,15] as shown in Table 3.

Table 3. Ranges [14,15].

Water Quality Index Range	Class
0–25	Very bad
25–50	Bad
50–70	Medium
70–90	Good
90–100	Excellent

3.5. Q-Value Normalization

-value normalization was used to normalize the parameters, particularly the water quality parameters to fit them in the range of 0 to 100 for easier index calculation. Figure 3 shows the q-value charts for six of the water quality parameters. We used them to convert five of these parameters within the range of 0 to 100 [14,15]. For the sixth parameter, namely nitrites, due to unavailability of its q-value ranges, we used the WHO standards to distinctly convert them to the 0–100 range by means of a set of thresholds as follows: assigning 100 if its below 1, 80 if its below 2, 50 if its below 3 and 0 if its greater than 3, reflecting strict penalization. Once the values were q-normalized and were in the range of 0–100, they were used for calculating the WQI of the dataset using (1). Water 2019, 11, x FOR PEER REVIEW 5 of 14 calculated by initially multiplying the q value of each parameter by its corresponding weight, adding them all up and then dividing the result by the sum of weights of the employed parameters [14,15]. WQI = $\sum q$[illegible] $* \ w\_factor \ \sum w\_factor$ (1) Table 2. Parameters weights for the WQI calculation [14,15]. Weighing Factor Weight pH 0.11 Temperature 0.10 Turbidity 0.08 Total Dissolved Values 0.07 Nitrates 0.10 Fecal Coliform 0.16 3.4. Water Qulaity Class (WQC) Once we had estimated the WQI, we defined the water quality class (WQC) of each sample using the WQI in classification algorithms [14,15] as shown in Table 3. Table 3. Ranges [14,15]. Water Quality Index Range Class 0–25 Very bad 25–50 Bad 50–70 Medium 70–90 Good 90–100 Excellent 3

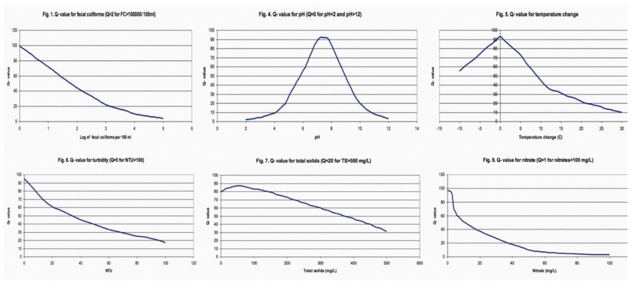


Figure 3. Q-value normalization range charts.

3.6. Z-Score Normalization

A raw data point that is above or below the population mean has a higher or lower z-score, which is a common standardization and normalizing procedure. The ideal position is between -3 and +3. To transform all the data with different scales to the default scale, the dataset is normalized to the aforementioned scale.

To use the z-score to normalize the data, subtract the population means from each raw data point and divide the result by the standard deviation. This yields a score that should range between 3 and +3, reflecting how many standard deviations a point is above or below the mean as calculated by Equation (2), where x represents the value of a specific sample, represents the mean, and represents the standard deviation [16].

$$\text{z\_score} = \frac{(x - \mu)}{\sigma}$$

3.7. Data Analysis

Following complete data processing, a variety of machine learning methods were used for data analysis in order to predict the WQI and WQC using the fewest possible parameters. There are certain preparatory stages, such as correlation analysis and data splitting, to prepare the data to be supplied as input to the real machine learning algorithms before using a machine learning algorithm.

3.7.1. Correlation Analysis

Following complete data processing, a variety of machine learning methods were used for data analysis in order to predict the WQI and WQC using the fewest possible parameters. There are certain preparatory stages, such as correlation analysis and data splitting, to prepare the data to be supplied as input to the real machine learning algorithms before using a machine learning algorithm.

Table 4's association chart shows that calcium and alkalinity (Alk) are closely connected to hardness (CaCO3) (Ca).

- Hardness has a weak correlation with pH and a strong correlation with calcium and alkalinity.
- Total dissolved solids, chlorides, and faecal coliform count are strongly connected with conductance, but calcium and temperature are very weakly correlated.
- Alkalinity and hardness have strong correlations with calcium, but TDS, chlorides, conductance, and pH have less correlations.
- TDS is inversely connected with calcium and temperature and strongly correlated with conductance, chlorides, and faecal coliform.
- Temperature, calcium, and faecal coliform are very weakly connected with chlorides, but conductance and TDS are substantially correlated.
- Fecal coliform is inversely linked with conductance, TDS, and chlorides.

Now that we have listed the correlation analysis observations, we find that our predicting parameter WQI is correlated with seven parameters, namely temperature, turbidity, pH, hardness as CaCO3, conductance, total dissolved solids and fecal coliform count. We have to choose the minimal number of parameters to predict the WQI, in order to lower the cost of the system. The three parameters whose sensors are easily available, cost the lowest and contribute distinctly to the WQI are temperature, turbidity and pH, which deems them naturally selected. The other convenient parameter is total dissolved solids, whose sensor is also easily available and is correlated with conductance and fecal coliform count, which means selecting TDS would allow us to discard the other two parameters. We leave the remaining inconvenient parameter, hardness as CaCO3, out because it is not highly correlated comparatively and is not easy to acquire

To conclude the correlation analysis, we selected four parameters for the prediction of WQI, namely, temperature, turbidity, pH and total dissolved solids. We initially just considered the first three parameters, given their low cost, and if needed, TDS will be included later to analyze its contribution to the accuracy.

Table 4. Correlation Analysis Chart.

	Temp	Turb	pH	Alk	$CaCO_3$	Cond	Ca	TDS	Cl	NO_2	FC	WQI
Temp	1.000	0.103	0.005	−0.193	−0.288	0.266	−0.150	0.274	0.293	−0.154	0.194	**−0.467**
Turb	0.103	1.000	−0.0886	−0.093	−0.146	0.048	−0.122	0.042	0.037	0.0002	0.037	**−0.354**
pH	0.005	−0.088	1.000	−0.177	−0.278	−0.065	−0.236	−0.060	−0.149	0.167	0.054	**−0.431**
Alk	−0.193	−0.092	−0.177	1.000	0.462	0.011	0.444	0.012	0.061	0.046	0.013	0.223
$CaCO_3$	−0.288	−0.146	−0.278	0.462	1.000	0.068	0.637	0.060	0.135	0.078	0.016	0.360
Cond	0.266	0.048	−0.064	0.011	0.068	1.000	0.225	0.973	0.780	0.100	0.456	−0.370
Ca	−0.150	−0.122	−0.236	0.444	0.637	0.225	1.000	0.219	0.262	0.124	0.113	0.188
TDS	0.273	0.041	−0.060	0.012	0.060	0.974	0.219	1.000	0.765	0.095	0.454	**−0.381**
Cl	0.292	0.037	−0.149	0.061	0.135	0.780	0.262	0.765	1.000	0.036	0.353	−0.274
NO_2	−0.154	0.0002	0.167	0.046	0.078	0.100	0.124	0.095	0.036	1.000	0.193	−0.209
FC	0.194	0.037	0.053	0.012	0.016	0.456	0.113	0.454	0.353	0.193	1.000	−0.421
WQI	−0.467	−0.354	−0.431	0.223	0.360	−0.370	0.188	−0.381	−0.274	−0.209	−0.421	1.000

3.7.2. Data Splitting–Cross Validation

The last step prior to applying the machine learning model is splitting the provided data in order to train the model, test it with a certain part of the data and compute the accuracy measures to establish the model's performance. This research explores the cross-validation data-splitting technique. Cross-validation splits the data into k subsets and iterates over all the subsets, considering k-1 subsets as the training dataset and 1 subset as the testing dataset. This ensures an efficient split and use of proper and definitive data for training and testing. This is generally computationally expensive, given the iterations, but our research uses a small dataset, which is mostly the case with water quality datasets, making cross-validation more suited for this problem. We split the data into k = 6 subsets and ran cross-validation. Therefore, as the complete training set consists of 663 samples, we ensured at least 100 samples for each fold subset, including the test set.

3.7.3. Machine Learning Algorithms

We used both regression and classification algorithms. We used the regression algorithms to estimate the WQI and the classification algorithms to classify samples into the previously defined WQC. We used eight regression algorithms and 10 classification algorithms. The following algorithms were employed in our study:

(1) Multiple Linear Regression

Multiple linear regression is a form of linear regression used when there is more than one predicting variable at play. When there are multiple input variables, we use multiple linear regression to assess the input of each variable that affects the output, as reflected in Equation (3), where y is the output for which machine learning has been applied to predict the value, x is the observed value, β is
the slope on the observed value, and ∈ is the error term [17].

$$y = \beta 0 + \beta 1x1 + \beta 2x2 + . . . + \beta kxk + \in \quad (3)$$

(2) Polynomial Regression

Polynomial regression is used when the relation between input and output variables is not linear and a little complex. We used a higher order of variables to capture the relation of input and output variables, which is not as linear. We used the order of two. Using a higher order of variables does carry the risk of overfitting, as reflected in Equation (4), where y is the output for which machine learning has been applied to predict the value, x is the observed value, β is the fitting value, i is the number of parameters considered, k is the order of the polynomial equation, and ∈I is the error term or residuals of the ith predictor [18]. We used it with 2-degree polynomials with an order of C.

$$y = \beta o + \beta 1xi + \beta 2x^2 + \beta 3x3 + ... + \beta kx + €_i, \text{ for } i = 1,2,, n$$

(3) Random Forest

Random forest is a model that uses multiple base models on subsets of the given data and makes decisions based on all the models. In random forest, the base model is a decision tree, carrying all the pros of a decision tree with the additional efficiency of using multiple models [19].

(4) Gradient Boosting Algorithm

This is the most contemporary algorithm used in most competitions. It uses an additive model that allows for optimization of differentiable loss function. We used it with a loss function of 'ls', a min_samples_split of 2 and a learning rate of 0.1 [20].

(5) Support Vector Machines

Support vector machines (SVMs) are mostly used for classification but they can be used for regression as well. Visualizing data points plotted on a plane, SVMs define a hyperplane between the classes and extend the margin in order to maximize the distinction between two classes, which results in fewer close miscalculations [21].

(6) Ridge Regression

Ridge regression works on the same principles as linear regression, it just adds a certain bias to negate the effect of large variances and to void the requirement of unbiased estimators. It penalizes the coefficients that are far from zero and minimizes the sum of squared residuals [22,23].

(7) Lasso Regression

Lasso regression works on the same principles as ridge regression, the only difference is how they penalize their coefficients that are off. Lasso penalizes the sum of absolute errors instead of the sum of squared coefficients [24].

(8) Elastic Net Regression

Elastic net regression combines the best of both ridge and lasso regression. It combines the method of penalties of both methods and minimizes the loss function [25].

(9) Neural Net/Multi-Layer Perceptrons (MLP)

Neural nets are loosely based on the structure of neurons. They contain multiple layers with interconnected nodes. They contain an input layer and output layer, and hidden layers in between these two mandatory layers. The input layer takes in the predicting parameters and the output layer shows the prediction based on the input. They iterate through each training data point and generalize the model by giving and updating the weight on each node of each layer. The trained model then uses those weights to decide what units to activate based on the input. Multi-layer perceptron (MLP) is a conventional model of a neural net, which is mostly used for classification, but it can be used for regression as well [26]. We used it for classification with the configuration of (3, 7) running for a maximum of 200 epochs using 'lbfgs' solver.

(10) Gaussian Naïve Bayes

Naïve Bayes is a simple and a fast algorithm that works on the principle of Bayes theorem with the assumption that the probability of the presence of one feature is unrelated to the probability of the presence of the other feature [27].

(11) Logistic Regression

Logistic regression is a classification algorithm. It is based on the logistic function or the sigmoid function, hence the name. It is the most common algorithm used in the case of binary classification, but in our case we used multinomial logistic regression because there was more than two classes [28]. We used it with 'warn' solver and l2 penalty.

(12) Stochastic gradient descent

This iterative optimization algorithm minimizes the loss function iteratively to find the global optimum. In stochastic gradient descent, the sample selection is random [29].

(13) K Nearest Neighbor

The K nearest neighbor algorithm classifies by finding the given points nearest N neighbors and assigns the class of majority of n neighbors to it. In the case of a draw, one could employ different techniques to resolve it, e.g., increase n or add bias towards one class. K nearest neighbor is not recommended for large datasets because all the processing takes place while testing, and it iterates through the whole training data and computes nearest neighbors each time [30]. We used a n = 5 configuration for our model.

(14) Decision Tree

A decision tree is a simple self-explanatory algorithm, which can be used for both classification and regression. The decision tree, after training, makes decisions based on values of all the relevant input parameters. It uses entropy to select the root variable, and, based on this, it looks towards the other parameters' values. It has all the parameter decisions arranged in a top-to-down tree and projects the decision based on different values of different parameters [31].

(15) Bagging Classifier

A bagging classifier fits multiple base classifiers on random subsets of data and then averages out their predictions to form the final prediction. It greatly helps out with the variance [32].

We used default values for the algorithms, except MLP, which uses a (3, 7) configuration.

4. Our Work

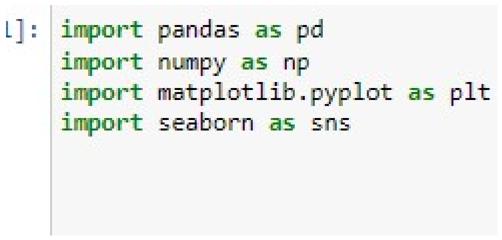
Access to safe drinking water is crucial to human health, especially in today's world where the pollutants flow on the air, in the water, to destruct human health, water quality is inevitebly important to consider.

not only the about the drinkable water, when water is considered as a resource, when polluted, can harm all the creatures in the world. Being said that, our project focuses on the 1st problem whether or not the water is potable?

The machine learning algorithms we use predicts the potability of the water, This prediction is not only meant for 1 on 1 application for human, it is also used in different plates to exemplify, water treatement plants are one of the application where this water quality prediction system can be used, where it determines the amount of disinfectent to be added, the ph value at which the water in a plant should be maintained. It all starts with series of sensors collecting the data on water quality attributes,

Our projects uses one such dataset availably publicly at kaggle website. It has 9 input features namely, sulphate, ph, hardness and so on and so forth, which determines the potability of the water. Which results the output variable in one of the two class 0: which is not potable and 1 which is potable

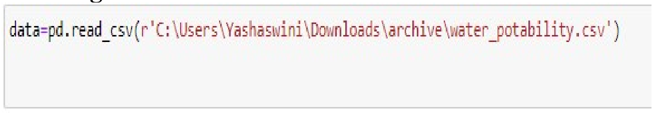
Data Collection:



```
]: import pandas as pd
   import numpy as np
   import matplotlib.pyplot as plt
   import seaborn as sns
```

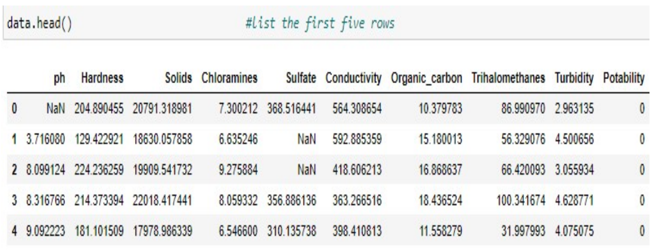
The data collected is from publicly available data set, from Kaggle website. It has 9 input features which corresponds to the values collected by the sensors that define the quality of the water. The dataframe is created.

Creating the dataframe:



```
data=pd.read_csv(r'C:\Users\Yashaswini\Downloads\archive\water_potability.csv')
```

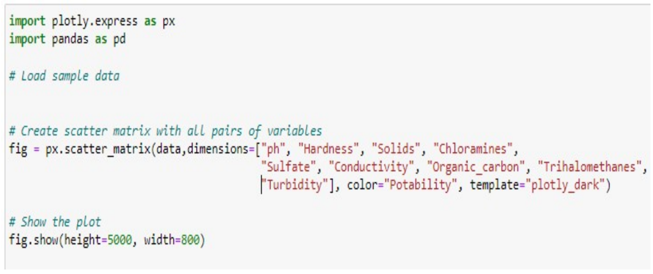
Data Samples:



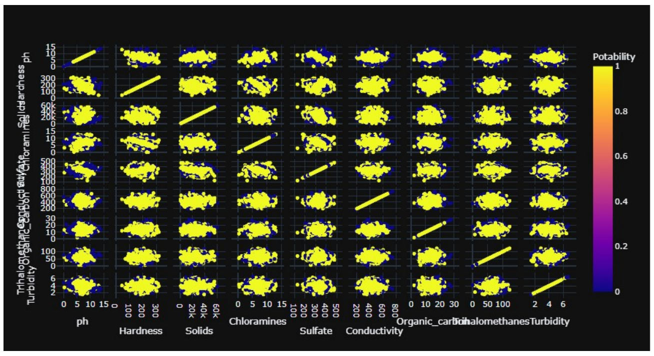
```
data.head()                         #list the first five rows
```

	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

Data Visualization:

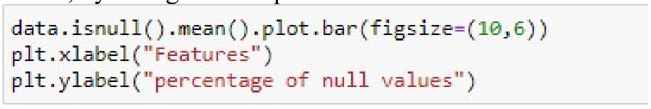


```
import plotly.express as px
import pandas as pd

# Load sample data


# Create scatter matrix with all pairs of variables
fig = px.scatter_matrix(data,dimensions=["ph", "Hardness", "Solids", "Chloramines",
                                         "Sulfate", "Conductivity", "Organic_carbon", "Trihalomethanes",
                                         "Turbidity"], color="Potability", template="plotly_dark")


# Show the plot
fig.show(height=5000, width=800)
```

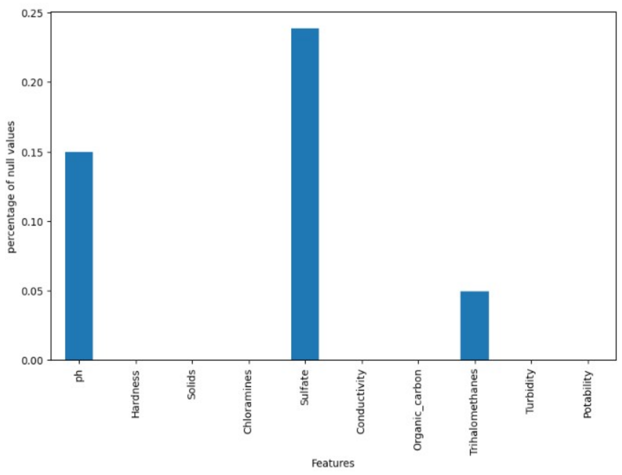


Data Cleaning:

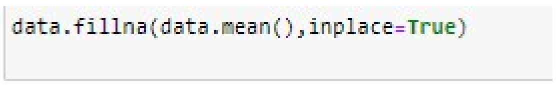
As a part of cleaning the data, we have handled the null values, by filling the null places with the mean of the data.



```
data.isnull().mean().plot.bar(figsize=(10,6))
plt.xlabel("Features")
plt.ylabel("percentage of null values")
```



Visualizing the null values

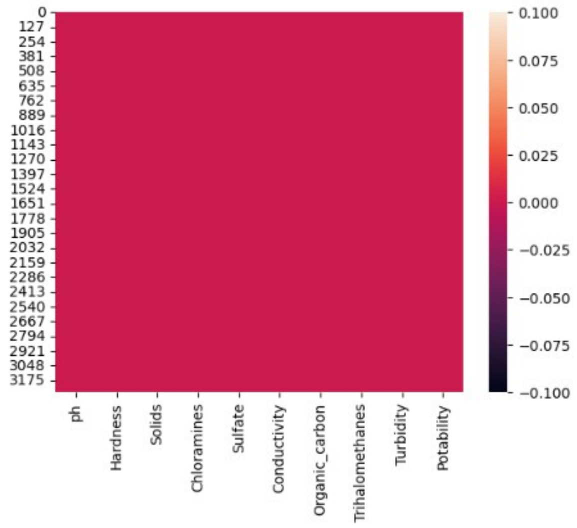


```
data.fillna(data.mean(),inplace=True)
```

Cleaning the data

```
sns.heatmap(data.isnull()) #no null values
```

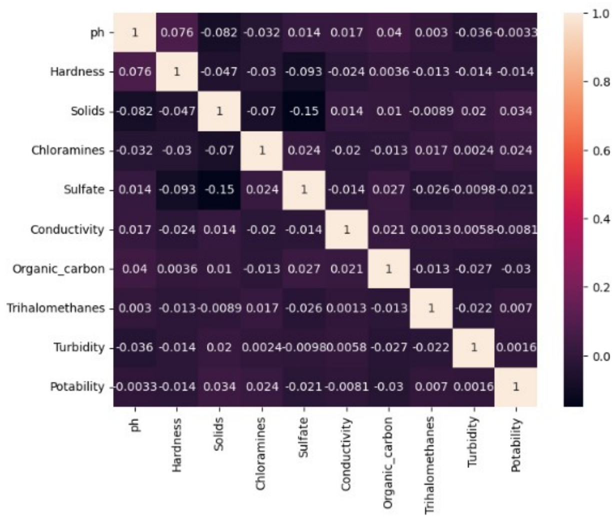
```
<Axes: >
```



Visualization after filling null values

Dimensionality Reduction

```
sns.heatmap(data.corr(),annot=True)
fig=plt.gcf()
fig.set_size_inches(8,6)
plt.show()
```



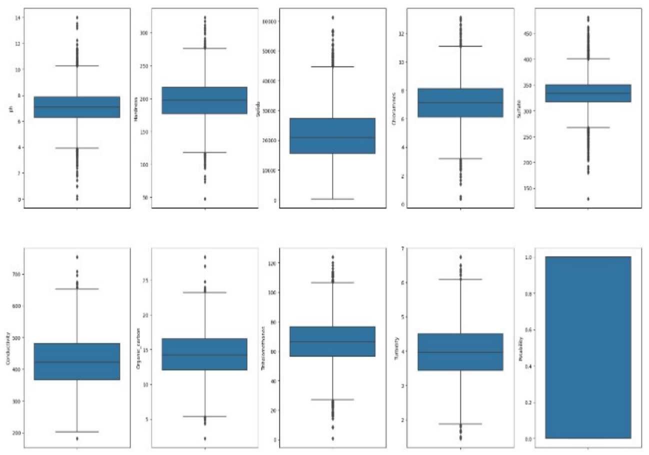
	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
ph	1	0.076	-0.082	-0.032	0.014	0.017	0.04	0.003	-0.036	-0.0033
Hardness	0.076	1	-0.047	-0.03	-0.093	-0.024	0.0036	-0.013	-0.014	-0.014
Solids	-0.082	-0.047	1	-0.07	-0.15	0.014	0.01	-0.0089	0.02	0.034
Chloramines	-0.032	-0.03	-0.07	1	0.024	-0.02	-0.013	0.017	0.0024	0.024
Sulfate	0.014	-0.093	-0.15	0.024	1	-0.014	0.027	-0.026	-0.0098	-0.021
Conductivity	0.017	-0.024	0.014	-0.02	-0.014	1	0.021	0.0013	0.0058	-0.0081
Organic_carbon	0.04	0.0036	0.01	-0.013	0.027	0.021	1	-0.013	-0.027	-0.03
Trihalomethanes	0.003	-0.013	-0.0089	0.017	-0.026	0.0013	-0.013	1	-0.022	0.007
Turbidity	-0.036	-0.014	0.02	0.0024	-0.0098	0.0058	-0.027	-0.022	1	0.0016
Potability	-0.0033	-0.014	0.034	0.024	-0.021	-0.0081	-0.03	0.007	0.0016	1

Visualizing the correlation

The correlation between the input features, is visualized using heat map. No features are removed out of the data because, there is no such strong correlation between the data, so every input feature has unique impact on the output, therefore no input features are removed.

Removal of Outliers

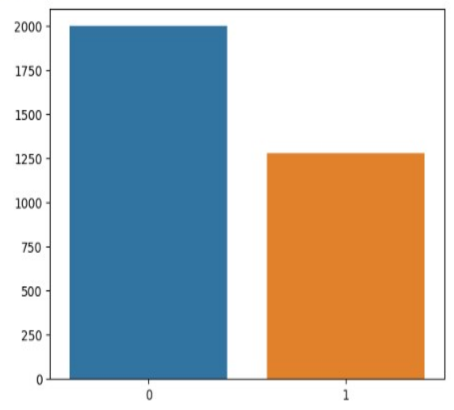
```
fig,ax =plt.subplots(ncols=5, nrows=2,figsize=(25,15))
ax=ax.flatten()
index=0
for col,value in data.items():
    sns.boxplot(y=col,data=data,ax=ax[index])
    index+=1
```



The outliers are are not removed in the data set because, in the real time, we can get the solution, probably the chemical solution that can be the outlier here.

Balancing the data:

```
sns.barplot(x=data['Potability'].value_counts().index, y = data['Potability'].value_counts().values)
plt.show()
```

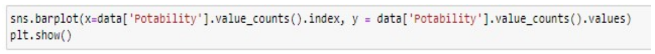


Before balancing the data

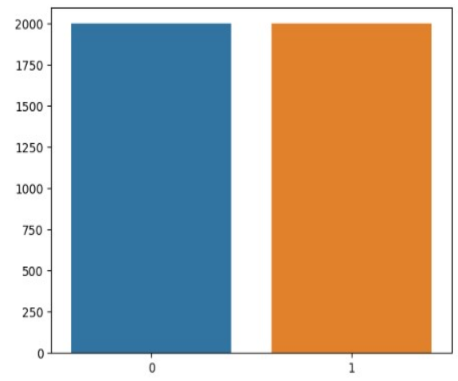
The balance of data was found to be 74% and 36% for the two classes of the output variable, out of which the class 0 has the highest value, that is solved using the

```
from imblearn.over_sampling import SMOTE

X1 = data.loc[:, data.columns != 'Potability']
Y1 = data.Potability


smote = SMOTE(k_neighbors=3, random_state=1234)
resam_X1,resam_Y1 = smote.fit_resample(X1, Y1)


data = pd.concat([pd.DataFrame(resam_X1), pd.DataFrame(resam_Y1)], axis=1)


print(data['Potability'].value_counts())
```

We use SMOTE to create new datapoints with neighbors of 3 and random seed is given as 1234 and the original data is appended with the new datapoints

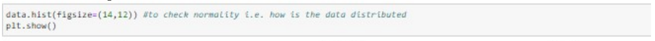


```
sns.barplot(x=data['Potability'].value_counts().index, y = data['Potability'].value_counts().values)
plt.show()
```

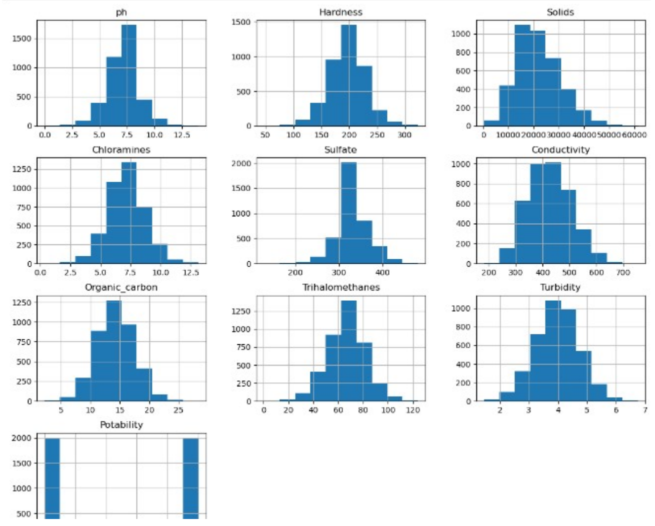


After balancing the data

Normality

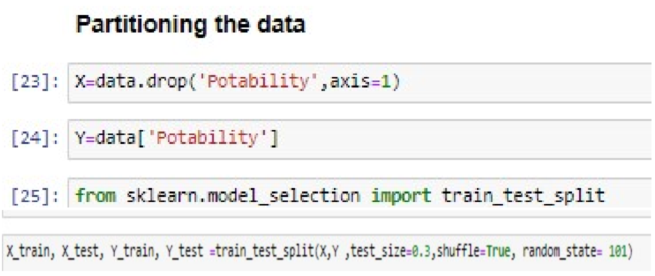


```
data.hist(figsize=(14,12)) #to check normality i.e. how is the data distributed
plt.show()
```



All the values in every feature, is normally distributed over the mean and hence need not to normalize further.

Partitioning the data

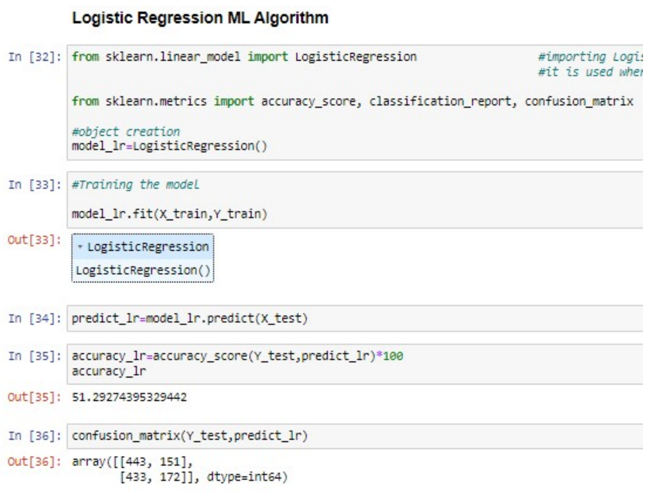


Partitioning the data

```
[23]: X=data.drop('Potability',axis=1)

[24]: Y=data['Potability']

[25]: 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,shuffle=True, random_state= 101)
```

The data is partitioned to 70:30 ratio

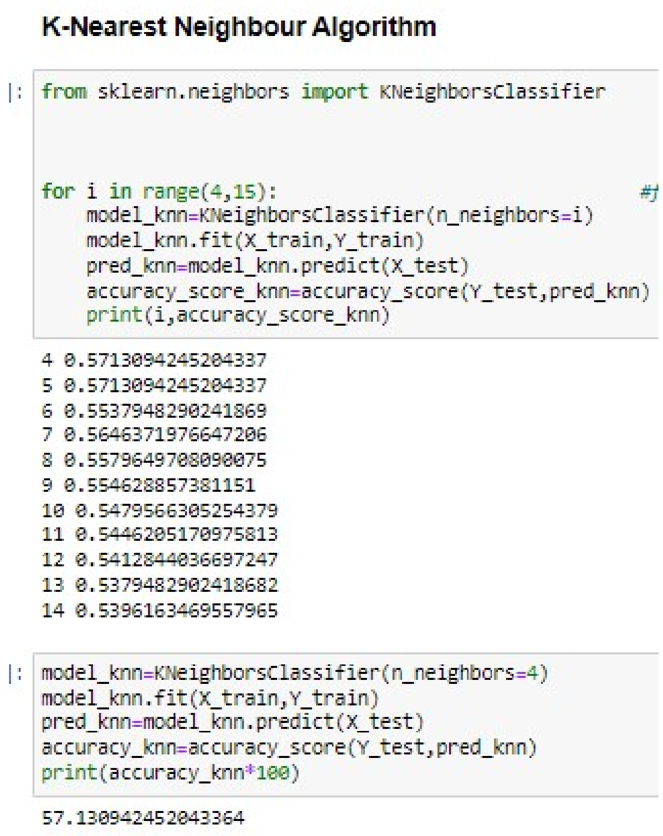
Model Training



Logistic Regression ML Algorithm

```
In [32]: from sklearn.linear_model import LogisticRegression             #importing Logis
                                                                          #it is used wher

         from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

         #object creation
         model_lr=LogisticRegression()

In [33]: #Training the model

         model_lr.fit(X_train,Y_train)

Out[33]: ▸ LogisticRegression
         LogisticRegression()

In [34]: predict_lr=model_lr.predict(X_test)

In [35]: accuracy_lr=accuracy_score(Y_test,predict_lr)*100
         accuracy_lr

Out[35]: 51.29274395329442

In [36]: confusion_matrix(Y_test,predict_lr)

Out[36]: array([[443, 151],
                [433, 172]], dtype=int64)
```

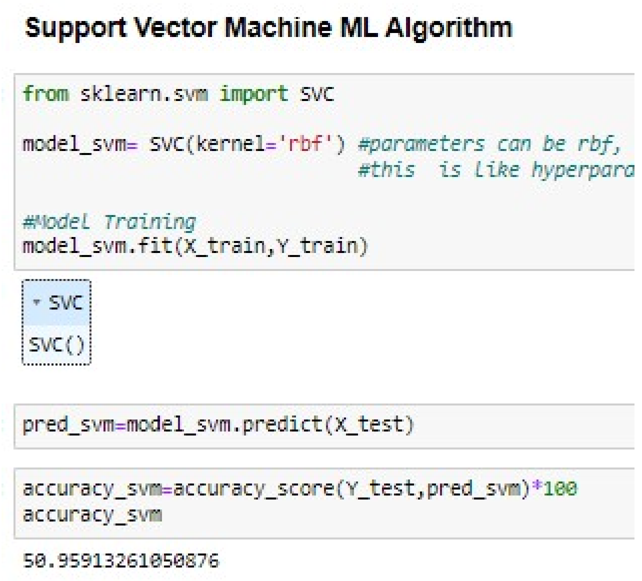
Logistic Regression ML Algorithm



K-Nearest Neighbour Algorithm

```
[ ]: from sklearn.neighbors import KNeighborsClassifier


     for i in range(4,15):                                              #f
         model_knn=KNeighborsClassifier(n_neighbors=i)
         model_knn.fit(X_train,Y_train)
         pred_knn=model_knn.predict(X_test)
         accuracy_score_knn=accuracy_score(Y_test,pred_knn)
         print(i,accuracy_score_knn)

     4 0.5713094245204337
     5 0.5713094245204337
     6 0.5537948290241869
     7 0.5646371976647206
     8 0.5579649708090075
     9 0.554628857381151
     10 0.5479566305254379
     11 0.5446205170975813
     12 0.5412844036697247
     13 0.5379482902418682
     14 0.5396163469557965

[ ]: model_knn=KNeighborsClassifier(n_neighbors=4)
     model_knn.fit(X_train,Y_train)
     pred_knn=model_knn.predict(X_test)
     accuracy_knn=accuracy_score(Y_test,pred_knn)
     print(accuracy_knn*100)

     57.130942452043364
```

KNN Algorithm



```
Support Vector Machine ML Algorithm

from sklearn.svm import SVC

model_svm= SVC(kernel='rbf') #parameters can be rbf,
                              #this  is like hyperpara

#Model Training
model_svm.fit(X_train,Y_train)
```

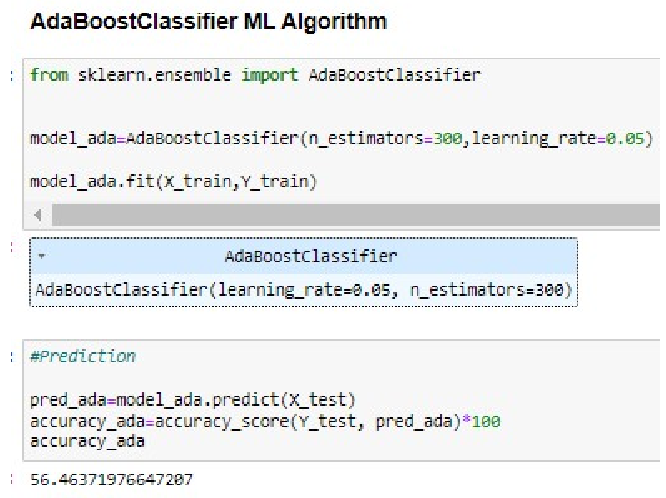
```
▾ SVC
SVC()
```

```
pred_svm=model_svm.predict(X_test)
```

```
accuracy_svm=accuracy_score(Y_test,pred_svm)*100
accuracy_svm
```

```
50.95913261050876
```

SVC ML Algorithm



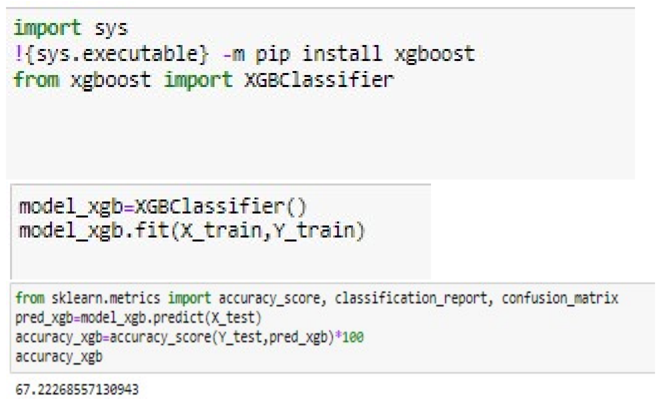
```
AdaBoostClassifier ML Algorithm

from sklearn.ensemble import AdaBoostClassifier


model_ada=AdaBoostClassifier(n_estimators=300,learning_rate=0.05)

model_ada.fit(X_train,Y_train)
```

```
▾                    AdaBoostClassifier
AdaBoostClassifier(learning_rate=0.05, n_estimators=300)
```

```
#Prediction

pred_ada=model_ada.predict(X_test)
accuracy_ada=accuracy_score(Y_test, pred_ada)*100
accuracy_ada
```

```
56.46371976647207
```

AdaBoostClassifier ML Algorithm



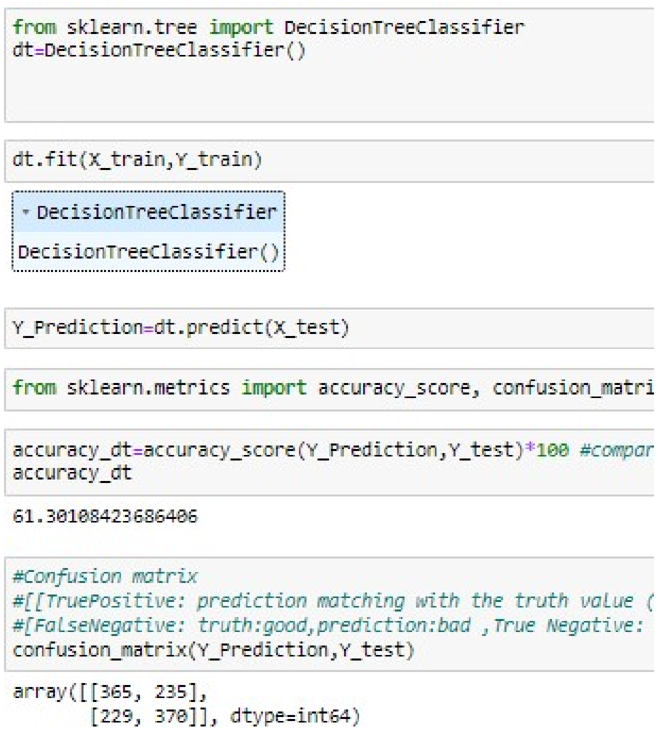
```
import sys
!{sys.executable} -m pip install xgboost
from xgboost import XGBClassifier
```

```
model_xgb=XGBClassifier()
model_xgb.fit(X_train,Y_train)
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
pred_xgb=model_xgb.predict(X_test)
accuracy_xgb=accuracy_score(Y_test,pred_xgb)*100
accuracy_xgb
```

```
67.2268557130943
```

XGBoost ML Algorithm



```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
```

```
dt.fit(X_train,Y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
Y_Prediction=dt.predict(X_test)
```

```
from sklearn.metrics import accuracy_score, confusion_matri
```

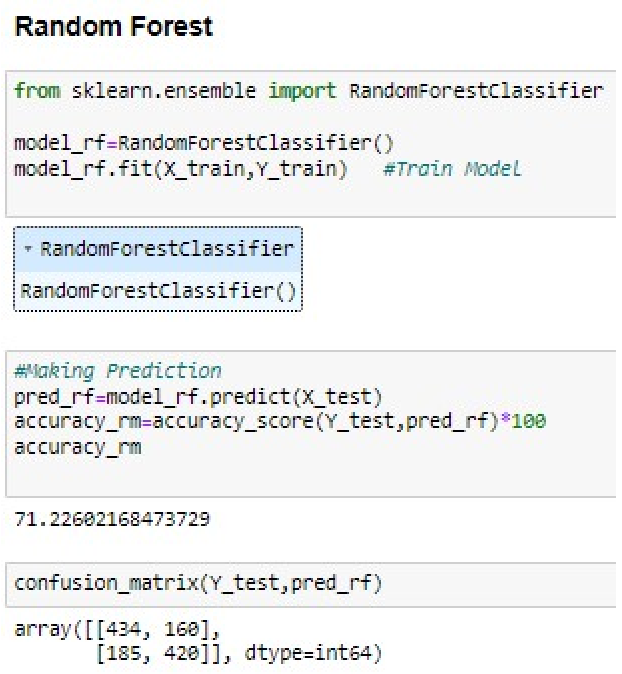
```
accuracy_dt=accuracy_score(Y_Prediction,Y_test)*100 #compar
accuracy_dt
```

```
61.30108423686406
```

```
#Confusion matrix
#[[TruePositive: prediction matching with the truth value (
#[FalseNegative: truth:good,prediction:bad ,True Negative:
confusion_matrix(Y_Prediction,Y_test)
```

```
array([[365, 235],
       [229, 370]], dtype=int64)
```

Decision Tree ML Algorithm



```
Random Forest

from sklearn.ensemble import RandomForestClassifier

model_rf=RandomForestClassifier()
model_rf.fit(X_train,Y_train)    #Train Model
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
#Making Prediction
pred_rf=model_rf.predict(X_test)
accuracy_rm=accuracy_score(Y_test,pred_rf)*100
accuracy_rm
```

```
71.22602168473729
```

```
confusion_matrix(Y_test,pred_rf)
```

```
array([[434, 160],
       [185, 420]], dtype=int64)
```

Random Forest Algorithm

Accuracy

Accuracy Result

```
models=pd.DataFrame({
    "Model":["Logistic Regression", "Random Forest", "KNN", "SVM", "AdaBoost","XGBoost","Decision Tree"],
    "Accuracy_Score":[accuracy_lr, accuracy_rm, accuracy_knn, accuracy_svm, accuracy_ada, accuracy_xgb, accuracy_dt]
})
```

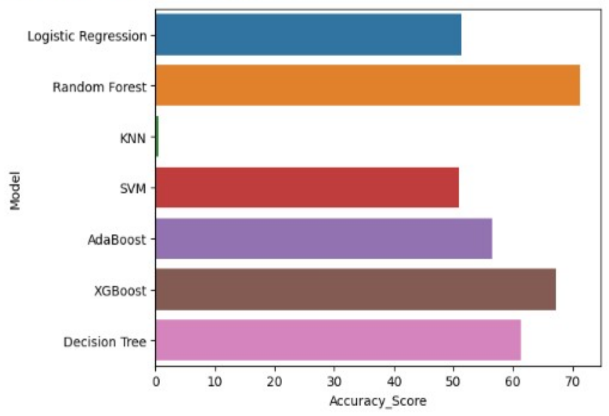
```
models
```

	Model	Accuracy_Score
0	Logistic Regression	51.292744
1	Random Forest	71.226022
2	KNN	0.557965
3	SVM	50.959133
4	AdaBoost	56.463720
5	XGBoost	67.222686
6	Decision Tree	61.301084

Analyzing the accuracy of different Models

```
import pandas as pd
import seaborn as sns


# Create the bar plot
sns.barplot(x="Accuracy_Score", y="Model", data=models)
```

```
<Axes: xlabel='Accuracy_Score', ylabel='Model'>
```



Visualizing the accuracy

Optimization

```
n_estimators=[20,60,100,120]
max_features=[0,2,0,6,1,0]
max_depth=[2,8,None]
max_samples=[0.5,0.75,1.0]
```

```
param_grid={'n_estimators':n_estimators,
           'max_features': max_features,
           'max_depth': max_depth,
           'max_samples': max_samples}
print(param_grid)
```

```
{'n_estimators': [20, 60, 100, 120], 'max_features': [0, 2, 0, 6, 1, 0], 'max_depth': [2, 8, None], 'max_samples': [0.5, 0.75,
1.0]}
```

```
rf=RandomForestClassifier()
```

```
from sklearn.model_selection import GridSearchCV

rf_grid=GridSearchCV(estimator=rf,
                     param_grid=param_grid,
                     cv=5,
                     verbose=2,
                     n_jobs=-1)
```

```
rf_grid.fit(X_train,Y_train)
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
```

```
▸          GridSearchCV
▸ estimator: RandomForestClassifier
      ▸ RandomForestClassifier
```

```
rf_grid.best_params_
```

```
{'max_depth': None, 'max_features': 6, 'max_samples': 1.0, 'n_estimators': 60}
```

```
rf_grid.best_score_
```

```
0.6728775875287504
```

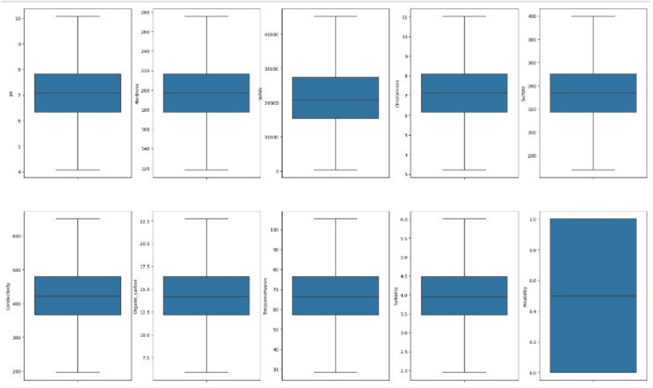
The optimization is done for random forest algorithm, we tune the hyper parameter using GridSearchCV, and trying to find the best parameters by training with the data. The best parameter score is printed which is less than the accuracy we got.

Outlier Remover

```
def cap_data(data):
    for col in data.columns:
        print("\n\n capping the \n",col)
        if (((data[col].dtype)=='float64') | ((data[col].dtype)=='int64')):

            q1=data[col].quantile(0.25)
            q3=data[col].quantile(0.75)
            iqr=q3-q1
            lower,upper=(q1-(iqr*1.5)),(q3+(iqr*1.5))
            print("q1=",q1,"q3=",q3,"iqr=",iqr,"lower=",lower,"upper=",upper)
            data[col][data[col] <= lower] = lower
            data[col][data[col] >= upper] = upper
            print("\n",data[col][data[col] <= lower] )
            print("\n",data[col][data[col] >= upper] )

        else:
            data[col]=data[col]
    return data

final_df=cap_data(data)
```

The outier remover uses the method of capping where the extreme values are set to the threshold value

```
fig,ax =plt.subplots(ncols=5, nrows=2,figsize=(25,15))
ax=ax.flatten()
index=0
for col,value in data.items():
    sns.boxplot(y=col,data=data,ax=ax[index])
    index+=1
```



After removal of outliers

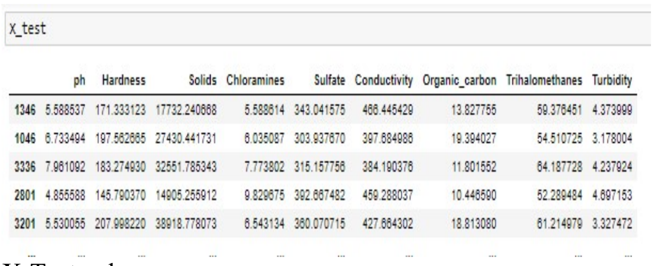
```
from sklearn.ensemble import RandomForestClassifier

model_rf=RandomForestClassifier()
model_rf.fit(X_train,Y_train)    #Train Model
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
#Making Prediction
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
pred_rf=model_rf.predict(X_test)
accuracy_rm=accuracy_score(Y_test,pred_rf)*100
accuracy_rm
```

```
69.47456213511259
```

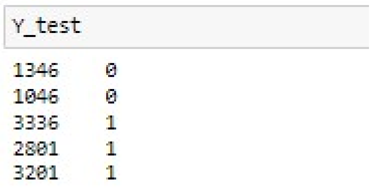
Removing outliers doesn't make any difference in accuracy



```
X_test
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity
1346	5.588537	171.333123	17732.240668	5.588614	343.041575	466.445429	13.827755	59.376451	4.373999
1046	6.733494	197.562665	27430.441731	6.035087	303.937670	397.684986	19.394027	54.510725	3.178004
3336	7.961092	183.274930	32551.785343	7.773802	315.157756	384.190376	11.801552	64.187728	4.237924
2801	4.855588	145.790370	14905.255912	9.829675	392.667482	459.288037	10.446590	52.289484	4.697153
3201	5.530055	207.998220	38918.778073	6.543134	360.070715	427.664302	18.813080	61.214979	3.327472
...	...	...	...	...	...	...	...	...	...

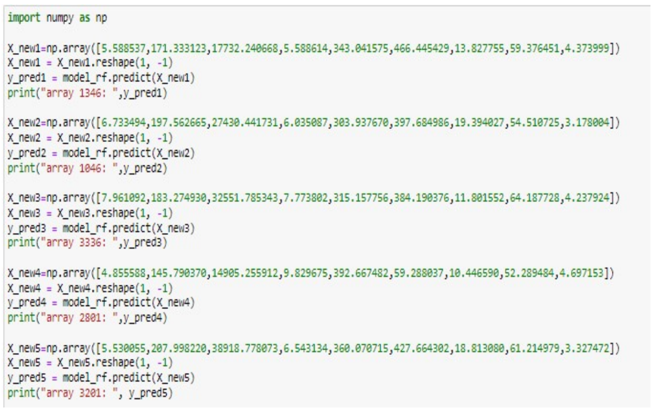
X-Test values



```
Y_test
```

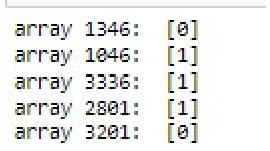
```
1346    0
1046    0
3336    1
2801    1
3201    1
```

Y-test values



```python
import numpy as np

X_new1=np.array([5.588537,171.333123,17732.240668,5.588614,343.041575,466.445429,13.827755,59.376451,4.373999])
X_new1 = X_new1.reshape(1, -1)
y_pred1 = model_rf.predict(X_new1)
print("array 1346: ",y_pred1)

X_new2=np.array([6.733494,197.562665,27430.441731,6.035087,303.937670,397.684986,19.394027,54.510725,3.178004])
X_new2 = X_new2.reshape(1, -1)
y_pred2 = model_rf.predict(X_new2)
print("array 1046: ",y_pred2)

X_new3=np.array([7.961092,183.274930,32551.785343,7.773802,315.157756,384.190376,11.801552,64.187728,4.237924])
X_new3 = X_new3.reshape(1, -1)
y_pred3 = model_rf.predict(X_new3)
print("array 3336: ",y_pred3)

X_new4=np.array([4.855588,145.790370,14905.255912,9.829675,392.667482,59.288037,10.446590,52.289484,4.697153])
X_new4 = X_new4.reshape(1, -1)
y_pred4 = model_rf.predict(X_new4)
print("array 2801: ",y_pred4)

X_new5=np.array([5.530055,207.998220,38918.778073,6.543134,360.070715,427.664302,18.813080,61.214979,3.327472])
X_new5 = X_new5.reshape(1, -1)
y_pred5 = model_rf.predict(X_new5)
print("array 3201: ", y_pred5)
```

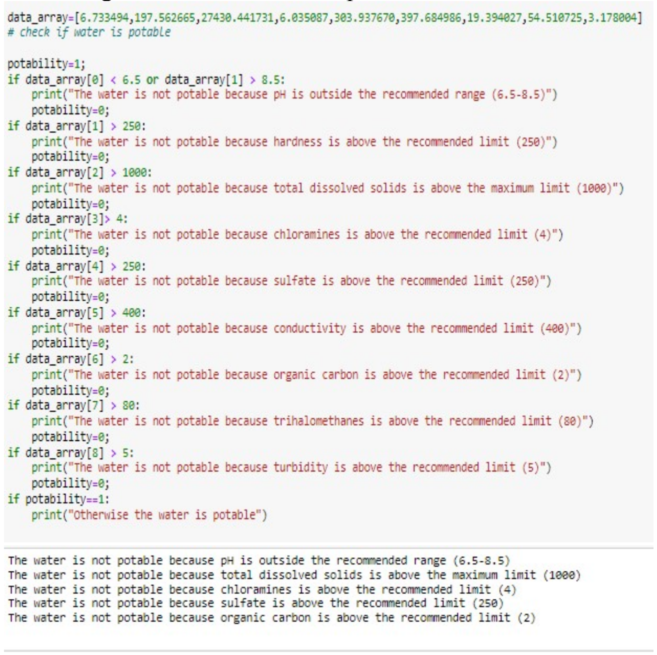
Given X-values



```
array 1346:  [0]
array 1046:  [1]
array 3336:  [1]
array 2801:  [1]
array 3201:  [0]
```

Predicted Y-values

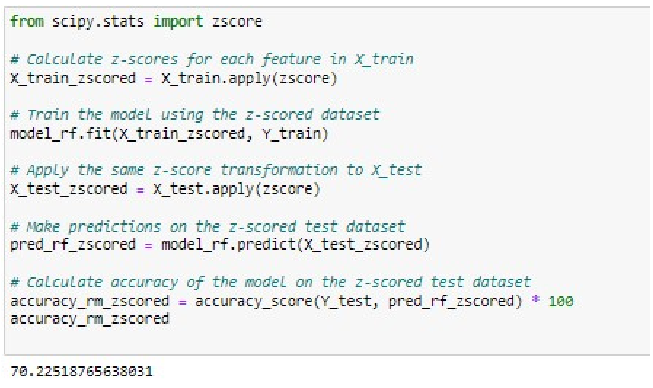
Predicting the reason for the output



```python
data_array=[6.733494,197.562665,27430.441731,6.035087,303.937670,397.684986,19.394027,54.510725,3.178004]
# check if water is potable

potability=1;
if data_array[0] < 6.5 or data_array[1] > 8.5:
    print("The water is not potable because pH is outside the recommended range (6.5-8.5)")
    potability=0;
if data_array[1] > 250:
    print("The water is not potable because hardness is above the recommended limit (250)")
    potability=0;
if data_array[2] > 1000:
    print("The water is not potable because total dissolved solids is above the maximum limit (1000)")
    potability=0;
if data_array[3]> 4:
    print("The water is not potable because chloramines is above the recommended limit (4)")
    potability=0;
if data_array[4] > 250:
    print("The water is not potable because sulfate is above the recommended limit (250)")
    potability=0;
if data_array[5] > 400:
    print("The water is not potable because conductivity is above the recommended limit (400)")
    potability=0;
if data_array[6] > 2:
    print("The water is not potable because organic carbon is above the recommended limit (2)")
    potability=0;
if data_array[7] > 80:
    print("The water is not potable because trihalomethanes is above the recommended limit (80)")
    potability=0;
if data_array[8] > 5:
    print("The water is not potable because turbidity is above the recommended limit (5)")
    potability=0;
if potability==1:
    print("Otherwise the water is potable")
```

```
The water is not potable because pH is outside the recommended range (6.5-8.5)
The water is not potable because total dissolved solids is above the maximum limit (1000)
The water is not potable because chloramines is above the recommended limit (4)
The water is not potable because sulfate is above the recommended limit (250)
The water is not potable because organic carbon is above the recommended limit (2)
```

Z-Score



```python
from scipy.stats import zscore

# Calculate z-scores for each feature in X_train
X_train_zscored = X_train.apply(zscore)

# Train the model using the z-scored dataset
model_rf.fit(X_train_zscored, Y_train)

# Apply the same z-score transformation to X_test
X_test_zscored = X_test.apply(zscore)

# Make predictions on the z-scored test dataset
pred_rf_zscored = model_rf.predict(X_test_zscored)

# Calculate accuracy of the model on the z-scored test dataset
accuracy_rm_zscored = accuracy_score(Y_test, pred_rf_zscored) * 100
accuracy_rm_zscored
```

```
70.22518765638031
```

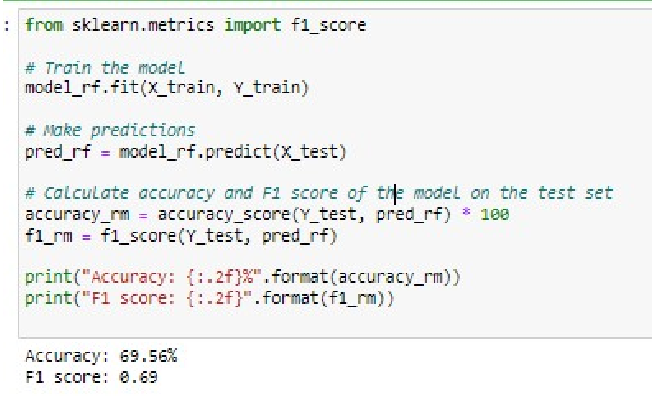
In this example, we first apply the zscore function to each feature in X_train using the apply method of a pandas DataFrame. This returns a new DataFrame X_train_zscored with the z-scores for each feature.

We then fit the RandomForestClassifier model to the z-scored training dataset X_train_zscored.

Next, we apply the same z-score transformation to the test dataset X_test using the apply method and store the result in a new DataFrame X_test_zscored.

Finally, we make predictions on the z-scored test dataset X_test_zscored using the predict method of the trained RandomForestClassifier model, and calculate the accuracy of the model on this z-scored test dataset using the accuracy_score function from scikit-learn.

F1-score



```python
from sklearn.metrics import f1_score

# Train the model
model_rf.fit(X_train, Y_train)

# Make predictions
pred_rf = model_rf.predict(X_test)

# Calculate accuracy and F1 score of the model on the test set
accuracy_rm = accuracy_score(Y_test, pred_rf) * 100
f1_rm = f1_score(Y_test, pred_rf)

print("Accuracy: {:.2f}%".format(accuracy_rm))
print("F1 score: {:.2f}".format(f1_rm))
```

```
Accuracy: 69.56%
F1 score: 0.69
```

we first train the RandomForestClassifier model on the training data X_train and labels Y_train.
We then use the trained model to make predictions on the test set X_test, and store the predicted labels in the variable pred_rf.
Next, we calculate the accuracy of the model on the test set using the accuracy_score function, as you have already done in your code.
Finally, we calculate the F1 score of the model on the test set using the f1_score function from the sklearn.metrics module, and print both the accuracy and F1 score for the model. The F1 score is a measure of the model's accuracy that takes into account both precision and recall, and can be a useful metric for evaluating classification models when the classes are imbalanced or when both precision and recall are important.

5. Discussion

Water Quality is conventionally calculated using water quality parameters, which are acquired through time-consuming lab analysis. We explored alternative methods of machine learning to estimate it and found several studies employing them. These studies used more than 10 water quality parameters to predict WQI. Ahmad et al. [8] used 25 input parameters, Sakizadeh [9] used 16 parameters, Gazzaz et al. [4] used 23 input parameters in their methodology, and Rankovic et al. [12] used 10 input parameters, which is unsuitable for inexpensive real-time systems. Whereas, our methodology employs only four water quality parameters to predict WQI, with an MAE of 1.96, and to predict water quality class with an accuracy of 85%. Our results make a base for an inexpensive real-time water quality detection system, while other studies, although they use machine learning, use too many parameters to be incorporated in real-time systems.

6. Conclusions and Future Work

Water is one of the most essential resources for survival and its quality is determined through WQI. Conventionally, to test water quality, one has to go through expensive and cumbersome lab analysis. This research explored an alternative method of machine learning to predict water quality using minimal and easily available water quality parameters. The data used to conduct the study were acquired from PCRWR and contained 663 samples from 12 different sources of Rawal Lake, Pakistan.

A set of representative supervised machine learning algorithms were employed to estimate WQI.

This showed that polynomial regression with a degree of 2, and gradient boosting, with a learning rate of 0.1, outperformed other regression algorithms by predicting WQI most efficiently, while MLP with a configuration of (3, 7) outperformed other classification algorithms by classifying WQC most efficiently.

In future works, we propose integrating the findings of this research in a large-scale IoT-based online monitoring system using only the sensors of the required parameters. The tested algorithms would predict the water quality immediately based on the real-time data fed from the IoT system.

The proposed IoT system would employ the parameter sensors of pH, turbidity, temperature and TDS for parameter readings and communicate those readings using an Arduino microcontroller and ZigBee transceiver. It would identify poor quality water before it is released for consumption and alert concerned authorities. It will hopefully result in curtailment of people consuming poor quality water and consequently de-escalate harrowing diseases like typhoid and diarrhea. In this regard, the application of a prescriptive analysis from the expected values would lead to future facilities to support decision and policy makers.

Author Contributions: Conceptualization, R.M.; Data curation, H.A.; Formal analysis, U.A. and R.I.; Investigation, A.A.S.; Methodology, U.A.; Resources, R.M.; Supervision, R.M.; Validation, U.A. and R.M.; Writing—original draft, U.A.; Writing—review & editing, R.M., H.A., R.I. and J.G.-N.

).

References

1. PCRWR. National Water Quality Monitoring Programme, Fifth Monitoring Report (2005–2006); Pakistan Council of Research in Water Resources Islamabad: Islamabad, Pakistan, 2007. Available online: http://www.pcrwr.gov.pk/Publications/Water%20Quality%20Reports/Water%20Quality% 20Monitoring%20Report%202005-06.pdf (accessed on 23 August 2019).
2. Mehmood, S.; Ahmad, A.; Ahmed, A.; Khalid, N.; Javed, T. Drinking Water Quality in Capital City of Pakistan. Open Access Sci. Rep. 2013, 2. [CrossRef]
3. PCRWR. Water Quality of Filtration Plants, Monitoring Report; PCRWR: Islamabad, Pakistan, 2010. Available online:http://www.pcrwr.gov.pk/Publications/Water%20Quality%20Reports/FILTRTAION%20PLANTS% 20REPOT-CDA.pdf (accessed on 23 August 2019).
4. Gazzaz, N.M.; Yusoff, M.K.; Aris, A.Z.; JuahirH.; Ramli, M.F. Artificial neural network modeling of the water quality index for Kinta River (Malaysia) using water quality variables as predictors. Mar.Pollut. Bull. 2012, 64, 2409–2420. [CrossRef]
5. Daud, M.K.; Nafees, M.; Ali, S.; Rizwan, M.; Bajwa, R.A.; Shakoor, M.B.; Arshad, M.U.; Chatha, S.A.S.; Deeba, F.; Murad, W.; et al. Drinking water quality status and contamination in Pakistan. BioMed Res. Int. 2017, 2017, 7908183. [CrossRef]
6. Alamgir, A.; Khan, M.N.A.; Hany; Shaukat, S.S.; Mehmood, K.; Ahmed, A.; Ali, S.J.; Ahmed, S. Public health quality of drinking water supply in Orangi town, Karachi, Pakistan. Bull. Environ. Pharmacol. Life Sci. 2015, 4, 88–94.
7. Shafi, U.; Mumtaz, R.; Anwar, H.; Qamar, A.M.; Khurshid, H. Surface Water Pollution Detection using Internet of Things. In Proceedings of the 2018 15th International Conference on Smart Cities: Improving Quality of Life Using ICT & IoT (HONET-ICT), Islamabad, Pakistan, 8–10 October 2018; pp. 92–96.
8. Ahmad, Z.; Rahim, N.; Bahadori, A.; Zhang, J. Improving water quality index prediction in Perak River basin Malaysia through a combination of multiple neural networks. Int. J. River Basin Manag. 2017, 15, 79–87. [CrossRef]
9. Sakizadeh, M. Artificial intelligence for the prediction of water quality index in groundwater systems. Model. Earth Syst. Environ. 2016, 2, 8. [CrossRef]
10. Abyaneh, H.Z. Evaluation of multivariate linear regression and artificial neural networks in prediction of water quality parameters. J. Environ. Health Sci. Eng. 2014, 12, 40. [CrossRef]
11. Ali, M.; Qamar, A.M. Data analysis, quality indexing and prediction of water quality for the management of rawal watershed in Pakistan. In Proceedings of the Eighth International Conference on Digital Information

Management (ICDIM 2013), Islamabad, Pakistan, 10–12 September 2013; pp. 108–113.
12. Rankovi´c, V.; Radulovi´c, J.; Radojevi´c, I.; Ostoji´c, A.; Comi´c, L. Neural network modeling of dissolved oxygen ˇ
in the Gruža reservoir, Serbia. Ecol. Model. 2010, 221, 1239–1244. [CrossRef]
13. Kangabam, R.D.; Bhoominathan, S.D.; Kanagaraj, S.; Govindaraju, M. Development of a water quality index (WQI) for the Loktak Lake in India. Appl. Water Sci. 2017, 7, 2907–2918. [CrossRef]
14. Thukral, A.; Bhardwaj, R.; Kaur, R. Water quality indices. Sat 2005, 1, 99.
15. Srivastava, G.; Kumar, P. Water quality index with missing parameters. Int. J. Res. Eng. Technol. 2013, 2, 609–614
16. Jayalakshmi, T.; Santhakumaran, A. Statistical normalization and back propagation for classification. Int. J. Comput. Theory Eng. 2011, 3, 1793–8201.
17. Amral, N.; Ozveren, C.; King, D. Short term load forecasting using multiple linear regression. In Proceedings of
the 2007 42 nd International Universities Power Engineering Conference, Brighton, UK, 4–6 September 2007;
pp. 1192–1198.
18. Ostertagová, E. Modelling using polynomial regression. Procedia Eng. 2012, 48, 500–506. [CrossRef]
19. Liaw, A.; Wiener, M. Classification and regression by randomForest. R News 2002, 2, 18–22.
20. Friedman, J.H. Stochastic gradient boosting. Comput. Stat. Data Anal. 2002, 38, 367–378. [CrossRef]
21. Tong, S.; Koller, D. Support vector machine active learning with applications to text classification. J. Mach. Learn. Res. 2001, 2, 45–66.
22. Hoerl, A.E.; Kennard, R.W. Ridge regression: Biased estimation for nonorthogonal problems. Technometrics 1970, 12, 55–67. [CrossRef]
23. Zhang, Y.; Duchi, J.; Wainwright, M. Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates. J. Mach. Learn. Res. 2015, 16, 3299–3340.
24. Tibshirani, R. Regression shrinkage and selection via the lasso. J. R. Stat. Soc. Ser. B 1996, 58, 267–288. [CrossRef]
25. Zou, H.; Hastie, T. Regression shrinkage and selection via the elastic net, with applications to microarrays. J. R. Stat. Soc. Ser. B 2003, 67, 301–320. [CrossRef]
26. Günther, F.; Fritsch, S. Neuralnet: Training of neural networks. R J. 2010, 2, 30–38. [CrossRef]
27. Zhang, H. The optimality of naive Bayes. AA 2004, 1, 3.
28. Hosmer, D.W., Jr.; Lemeshow, S.; Sturdivant, R.X. Applied Logistic Regression; John Wiley Sons: Hoboken, NJ,
USA, 2013.
29. Bottou, L. Large-scale machine learning with stochastic gradient descent. In Proceedings of the COMPSTAT'2010, Paris, France, 22–27August 2010; pp. 177–186.
30. Beyer, K.; Goldstein, J.; Ramakrishnan, R.; Shaft, U. When is "nearest neighbor" meaningful? In Proceedings of the International Conference on Database Theory, Jerusalem, Israel, 10–12 January 1999; pp. 217–235.
31. Quinlan, J.R. Decision trees and decision-making. IEEE Trans. Syst. Man Cybern. 1990, 20, 339–346. [CrossRef]
32. Breiman, L. Bagging predictors. Mach. Learn. 1996, 24, 123–140. [CrossRef]