PROJECT BASED EXPERIENTIAL LEARNING PROGRAM (NALAIYA THIRAN)

EFFICIENT WATER QUALITY ANALYSIS & PREDICTION USING MACHINE LEARNING

A PROJECT REPORT

Submitted by

1. **Puttu Bharath Kumar -410719106014**
2. **Kishore Kumar -410719106049**
3. **Ashok Kumar -410719106041**
4. **Naga Babu -410719106066**

TEAM ID :PNT2022TMID28764

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

Dhanalakshmi College Of Engineering

Tambarm, Chennai-601 301

Project Report Format

1 INTRODUCTION

1.1PROJECT Overview

Water plays a vital role in everyone's life and is observed everywhere and in every form. In Today's world, due to climatic changes and pollution the water quality has been affected in urban areas and various experiments are done to test the quality of water. Due to poor water quality, risk occurs in the industrial areas which damage the whole environment and causes an economical loss.The root cause for many diseases such as typhoid, diarrhea, cholera is due to usage of contaminated water caused by increased industrialization and urbanization in India.

According to reports form WHO, it is estimated that about 77 million people affected by contaminated water in India and 21% of diseases are caused due to it.Due to insufficient rainfall and drying up of main reservoirs that supplies water, India faces water crisis frequently, hence making water one of the most precious and limited land resources.

Many Organizations including WHO and BIS have framed standards for water parameters that can be used to efficiently analyze the quality of water. For checking the quality of water, conventionally it is required to collect water samples and send it to the lab for testing which is a tedious process. With Machine Learning algorithms it is easy to monitor and predict the quality of water at the comfort of our home.

To evaluate the quality of water, two approaches are considered, including measuring the water quality components and defining the mechanism of pollution transmission. Among water quality components, measuring the dissolved oxygen (DO), chemical oxygen demand (COD), biochemical oxygen demand (BOD), electrical conductivity (EC), pH, temperature,nitrate,fecal coliform etc. have been proposed.

However predicting the urban water quality is a challenging task since the water quality varies in urban spaces non-linearly and depends on multiple factors, such as meteorology, water usage patterns, and land uses, so this project aims at building a Machine Learning (ML) model to Predict Water Quality by considering all water quality standard indicators.

1.2 Purpose

The purpose is to predict the quality of water instead of using physical measurements or sensors to obtain the quality of water. This improves the accuracy of measurement over existing chemical and physical techniques as it is infeasible to obtain all the required features to predict the water quality.

Analyze the available data to clean, normalize and perform feature selection on the water quality measures, and therefore, to obtain the minimum relevant subset that allows high precision with low cost.

The complete methodology is proposed in the context of water quality numerical analysis. After much experimentation, the results reflect that gradient boosting and polynomial regression predict the WQI (water quality index) best with a mean absolute error (MAE).

They make it easily accessible by anyone who has internet services and has no specific software and hardware specification .Thus Helping in getting all required aspects of water.

2 LITERATURE SURVEY

2.1 Existing problems

There needs to be a more user-centric approach towards tackling the water quality issues, by using user- friendly tools and an interactive environment so that the solution actually benefits in tackling water quality issues.

To determine whether the water can be recycled or reused. Using ML techniques (Regression models) to predict the quality of water instead of using physical measurements or sensors to obtain the quality of water.

Not all models have been able to numerically predict the magnesium absorption ratio (MAR) and the permeability index (PI), so classification models may be able to improve the accuracy of predictions.

2.2 References

https://www.mdpi.com/2073-4441/11/11/2210

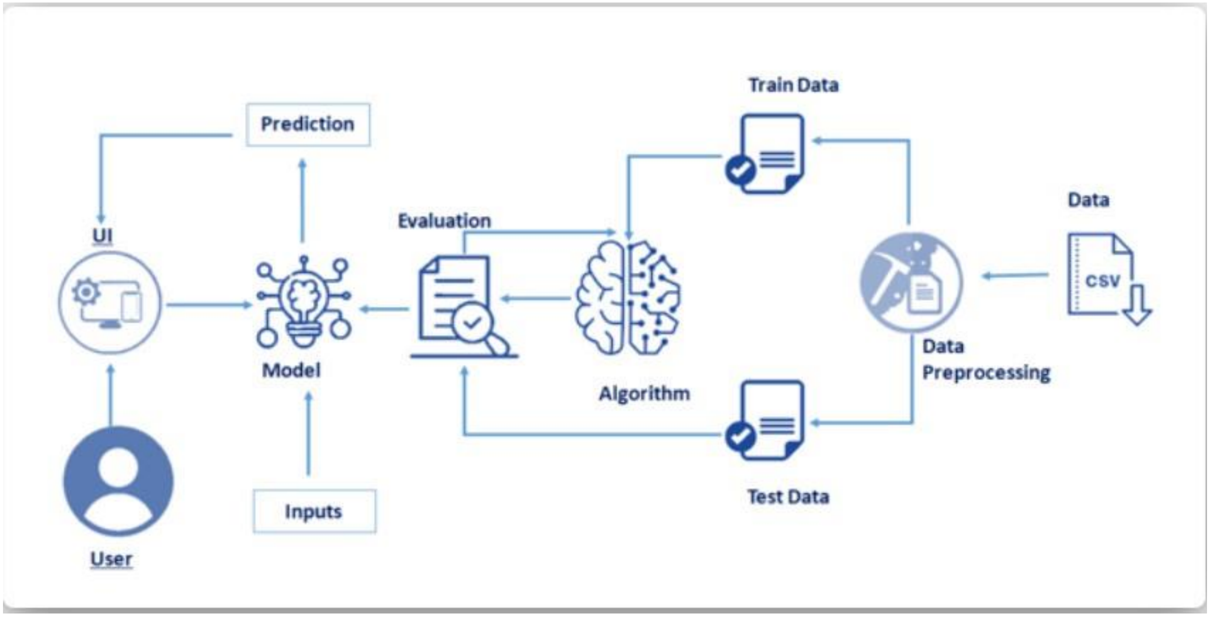
https://ieeexplore.ieee.org/document/9016825 https://www.atlantis-press.com/journals/hcis/125965714/view

https://ieeexplore.ieee.org/document/7494106

2.3 Problem Statement Definition

Water is considered as a vital resource that affects various aspects of human health and lives. The quality of water is a major concern for people living in urban areas. The quality of water serves as a powerful environmental determinant and a foundation for the prevention and control of waterborne diseases.

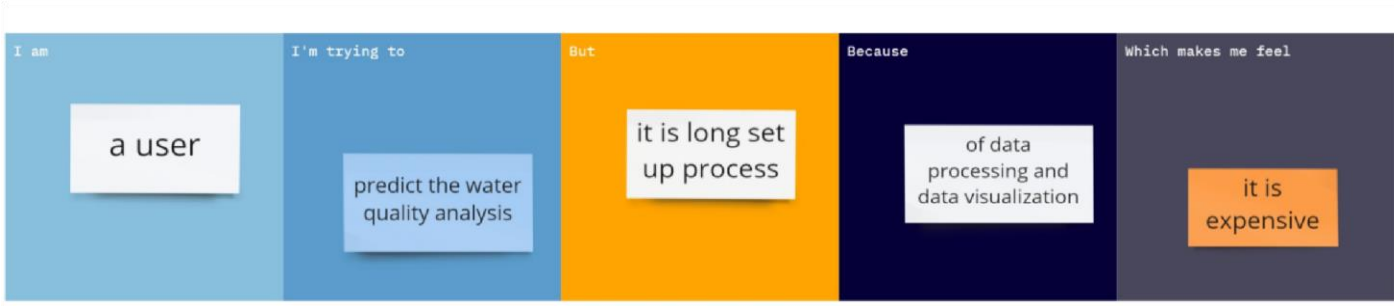
However predicting the urban water quality is a challenging task since the water quality varies in urban spaces non-linearly and depends on multiple factors, such as meteorology, water usage patterns, and land uses, so this project aims at building a Machine Learning (ML) model to Predict Water Quality by considering all water quality standard indicators.

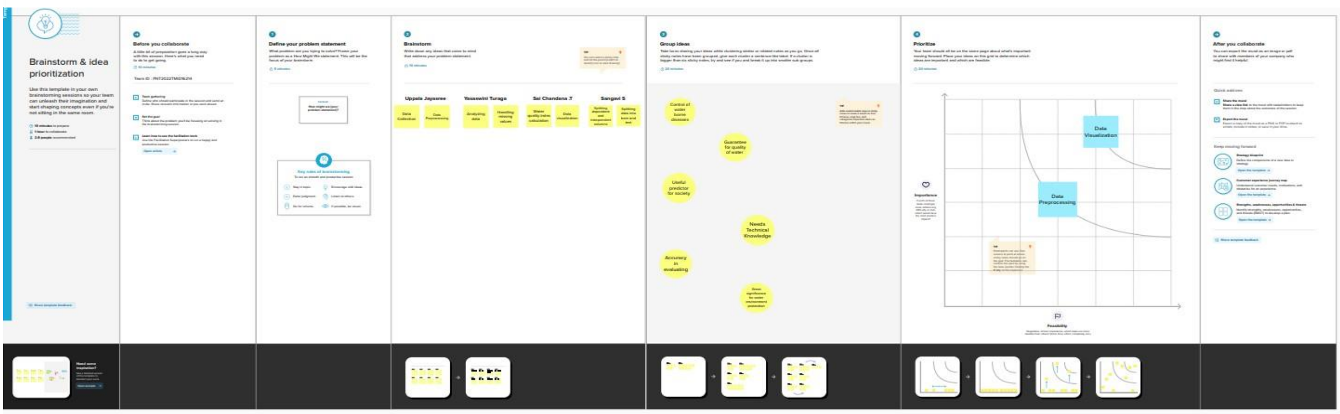


3 IDEATION & PROPOSED SOLUTION

3.1 Empathy map canvas

3.2 Brainstorming And Ideation





Problem Statement (Problem to be solved)	Water, a vital compound, plays an important role in life on earth. It has a direct impact on public health and the environment. It is a foundation for the prevention and control of waterborne diseases. However, predicting the urban water quality is a challenging task since the water quality varies in urban spaces non-linearly and depends on multiple factors, such as meteorology, water usage patterns, and land uses. The main aim of this project is to predict the water quality using machine learning technology.
Idea / Solution description	The solution is obtained from the data sets by comparing the accuracy rate with the previous data set and the current data set. Thus the Machine Learning techniques are used to predict the water quality index which denotes the healthiness of the water.
Novelty / Uniqueness	Used to determine whether the water can be recycled or reused. Its user Friendly. Using ML techniques (Regression models) to predict the quality of water instead of using physical measurements or sensors to obtain the quality of water. Thistechniques improves the accuracy of measurement over existing chemical and physical techniques as it is infeasible to obtain all the required features to predict the water quality.
Social Impact / Customer Satisfaction	Water’s quality is more important which should be considered as many water-borne diseases are more widely known. The proposed solution will help in identifying water pollution and helps the customer to drink healthy water.Beneficial for people's health. By analyzing the quality of water, good and healthy water is provided.

Business Model (Revenue Model)	First the application is tested with few people. Later on it comes into the picture where everyone can see by networking. By conducting various activities regarding the importance of quality of water.Industries that provide sanitation facilities and products (like water purifiers, quality testers etc.) can deploy this solution

3.3 Proposed solution

	to provide more wastewater treatment plants, better insights in health concerns and there may also be an increase in awareness and demand for better water quality testing and availability. People will start looking for treatments related to water-borne diseases as the awareness increases.

3.4 problem solution fit

4 REQUIREMENT ANALYSIS

4.1 Functional Requirements:

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	Check water quality analysis	• Water's quality is more important which should be considered as many water-borne diseases are more widely known. • So, it is necessary to analyze and predict the quality of water samples so as to determine and detect the contaminants present in those samples Patient dataset such as Temperature, PH, Conductivity, B.O.D, Nitratenan, Fecal, Coliform, Total Coliform, Year etc.
FR-2	Predict Water Quality by considering all water quality standard indicators	Using Machine learning model
FR-3	Accessing datasets	Datasets are collected by data preprocessing method then followed by data visualization.
FR-4	Classification of dataset	• Dataset includes data exploration. • In which prediction of water quality index calculation is performed using KNN ,SVM, ANN, Navis bayes and linear regression algorithms.
FR-5	Splitting and train the data	In this phase, we split the dataset into training and test dataset , and then trained the models using the training data set.

FR- 6	Test the model	In this phase, we tested the accuracy, precision and sensitivity of the models
		with the test dataset that was formed in previous phase and the most an accurate model is figured out.

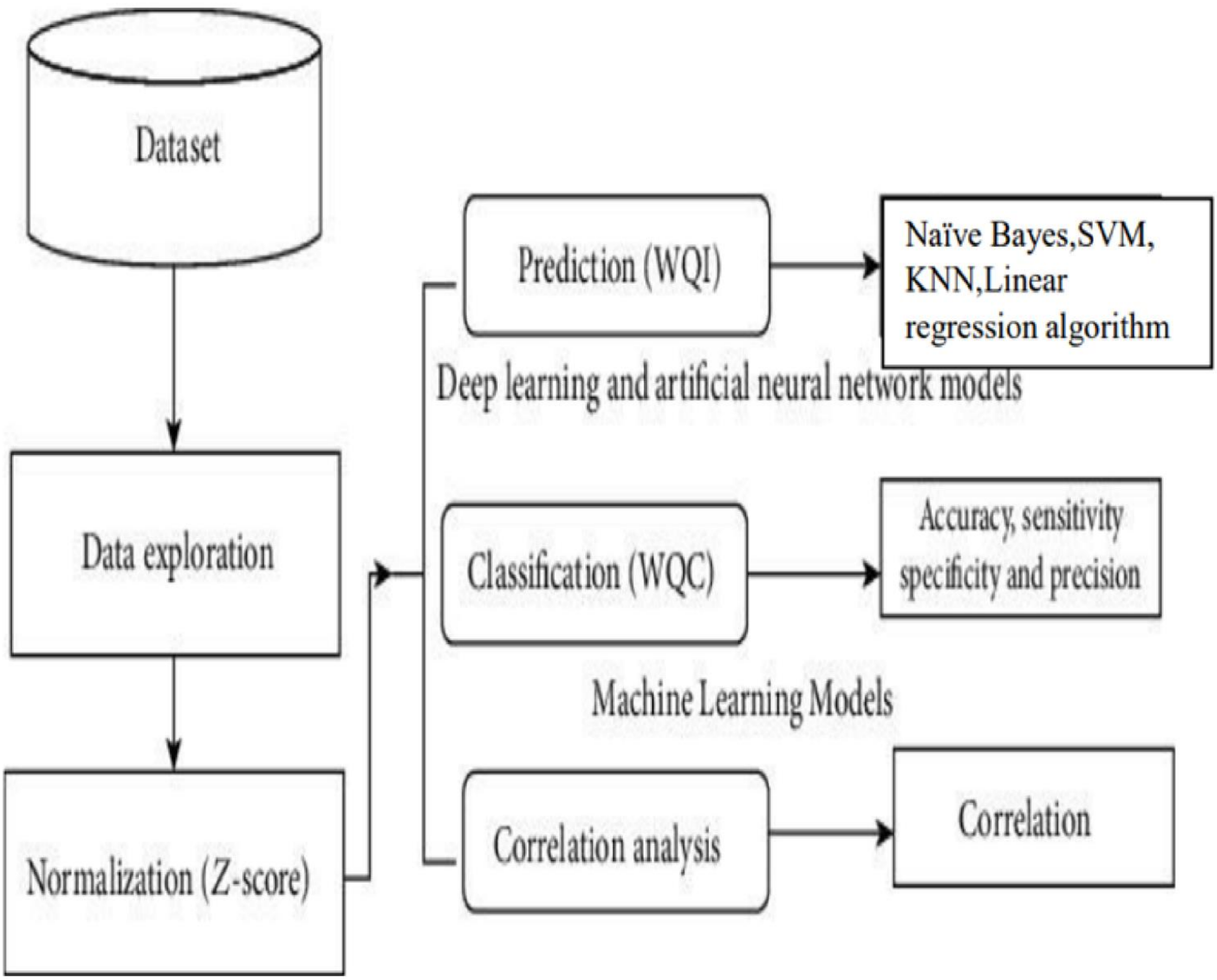
4.2 Non-Functional Requirements:

FR No	Non-Functional Requirement	Description
NFR-1	Usability	Predicting the urban water quality is a challenging task since the water quality varies in urban spaces non- linearly and depends on multiple factors, such as meteorology, water usage patterns, and land uses.
NFR-2	Availability	Industries that provide sanitation facilities and products (like water purifiers, quality testers etc.) can deploy this solution to provide more wastewater treatment plants, better insights in health concerns and there may also be an increase in awareness and demand for better water quality testing and availability.

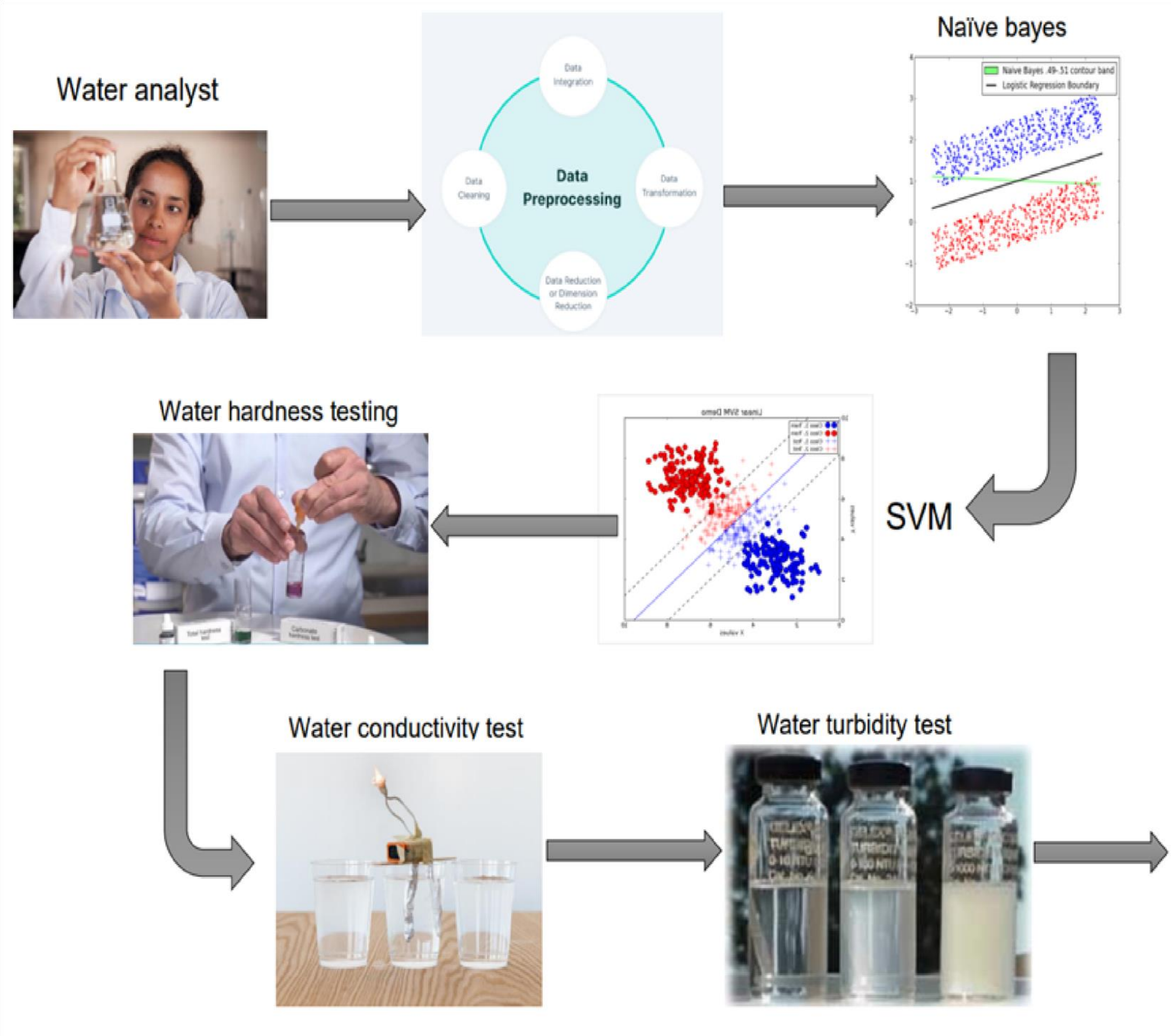
NFR-3	Reliability	• This project will help everyone in protecting their health. • Accurate water quality prediction is the basis of water environment management and is of great significance for water environment protection.
NFR-4	Performance	• This system uses different sensors for monitoring the water quality by determine pH, Turbidity, conductivity and temperature. • Data is gathered from different sources it is collected in a raw format and this data isn’t feasible for the analysis.
NFR-5	Security	• The quality of water is a major concern for people living in urban areas. • The quality of water serves as a powerful environmental determinant and a foundation for the prevention and control of waterborne diseases.
NFR-6	Scalability	This project used to measure and determine the quality of water. This provides pollution free and purified water.

5 PROJECT DESIGN

Data FlowDiagrams



5.2 Solution & Technical Architecture



5.3 USER STORIES

6. PROJECT PLANNING AND SCHEDULING:-

6.1 SPRINT PLANNING AND ESTIMATION

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Data Collection	USN-1	Collecting dataset for pre-processing	10	High	Uppala Jayasree T Sai Chandana Turaga Yasaswini Sangavi S
Sprint-1		USN-2	Data pre-processing- Used to transform the data into useful format.	10	Medium	Uppala Jayasree T Sai Chandana Turaga Yasaswini Sangavi S
Sprint-2	Model Building	USN-3	Calculate the Water Quality Index (WQI) using Regression algorithm of machine learning.	10	High	Uppala Jayasree T Sai Chandana Turaga Yasaswini Sangavi S
Sprint-2		USN-4	Splitting the data into training and testing from the entire dataset.	10	Medium	Uppala Jayasree T Sai Chandana Turaga Yasaswini Sangavi S
Sprint-3	Training and Testing	USN-5	Training the model using regression algorithm and testing the performance of the	20	Medium	Uppala Jayasree T Sai Chandana Turaga Yasaswini Sangavi S

			model.			
Sprint-4	Implementation of Web page	USN-6	Implementing the web page for collecting the data from user	10	High	Uppala Jayasree T Sai Chandana Turaga Yasaswini Sangavi S
Sprint-4		USN-6	Deploying the model using IBM Cloud and IBM Watson Studio	10	Medium	Uppala Jayasree T Sai Chandana Turaga Yasaswini Sangavi S

6.2 Sprint delivery schedule:

Velocity:

Sprint	**Total Story Points**	**Duration**	**Sprint Start Date**	**Sprint End Date (Planned)**	**Story Points Completed (as on Planned End Date)**	**Sprint Release Date (Actual)**
Sprint- 1	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint- 2	20	6 Days	31 Oct 2022	05 Nov 2022	20	05 Nov 2022
Sprint- 3	20	6 Days	07 Nov 2022	12 Nov 2022	20	12 Nov 2022
Sprint- 4	20	6 Days	14 Nov 2022	19 Nov 2022	20	19 Nov 2022

Sprint 1 Average Velocity:

Average Velocity = 20/2 = 10

Sprint 2 Average Velocity:

Average Velocity = 20/2 = 10

Sprint 3 Average Velocity:

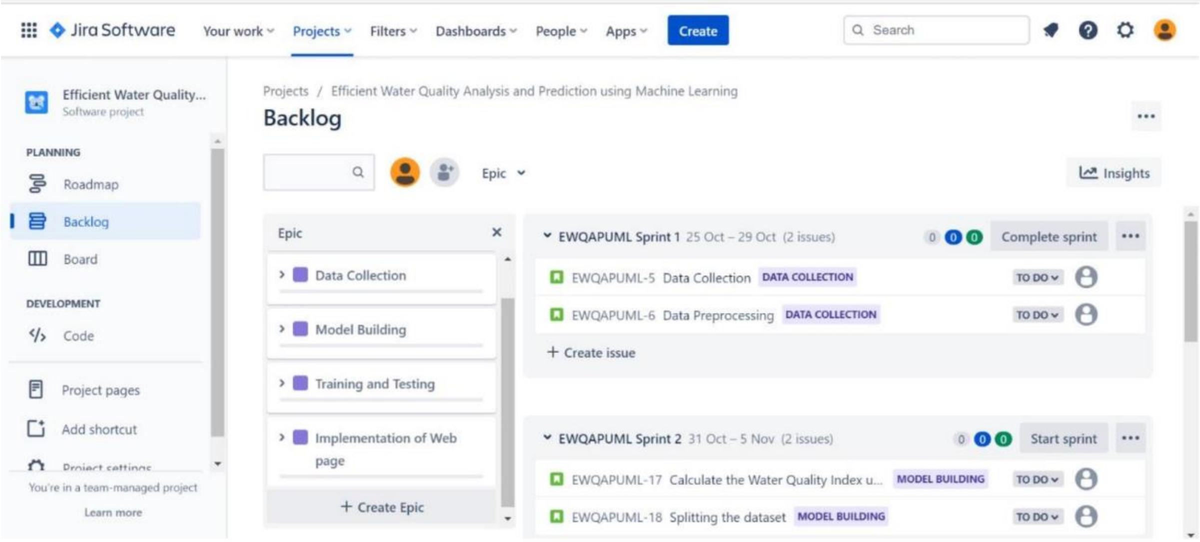
Average Velocity = 20/1 = 20

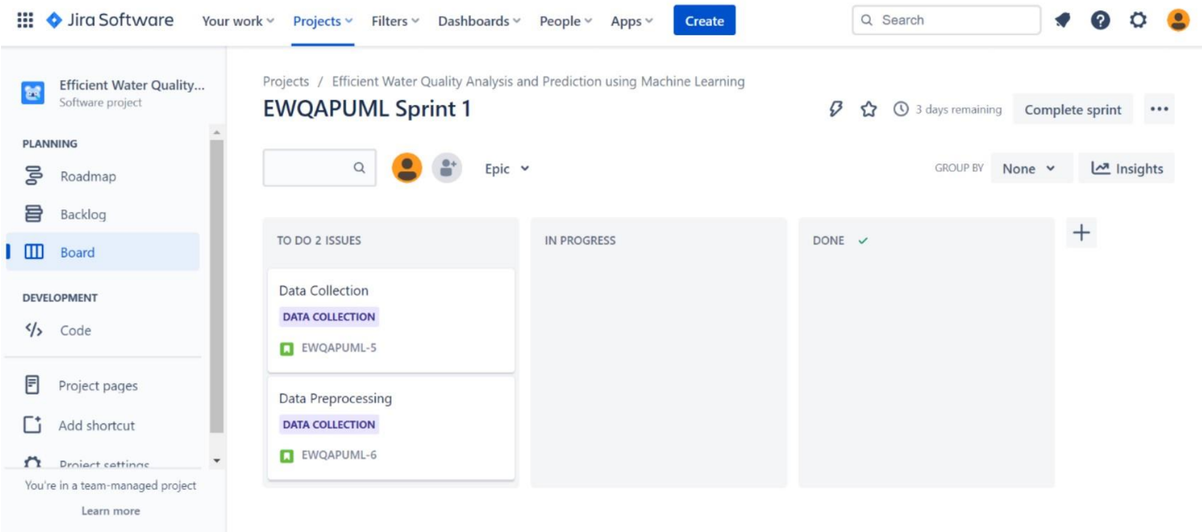
Sprint 4 Average Velocity:

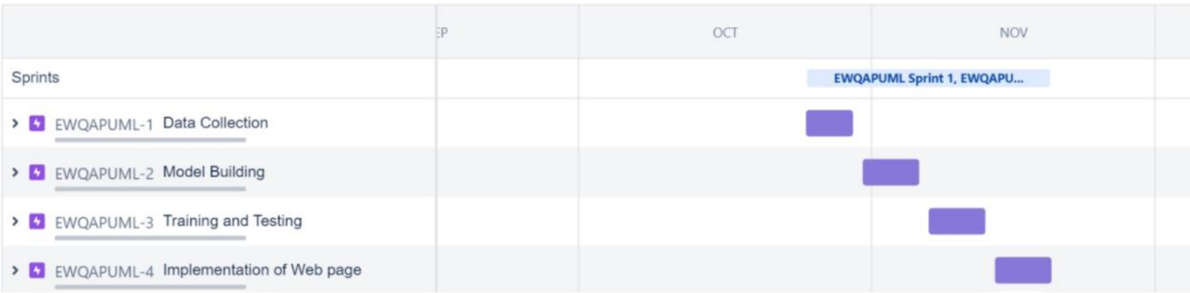
Average Velocity = 20/2 = 10

BURNDOWN CHART

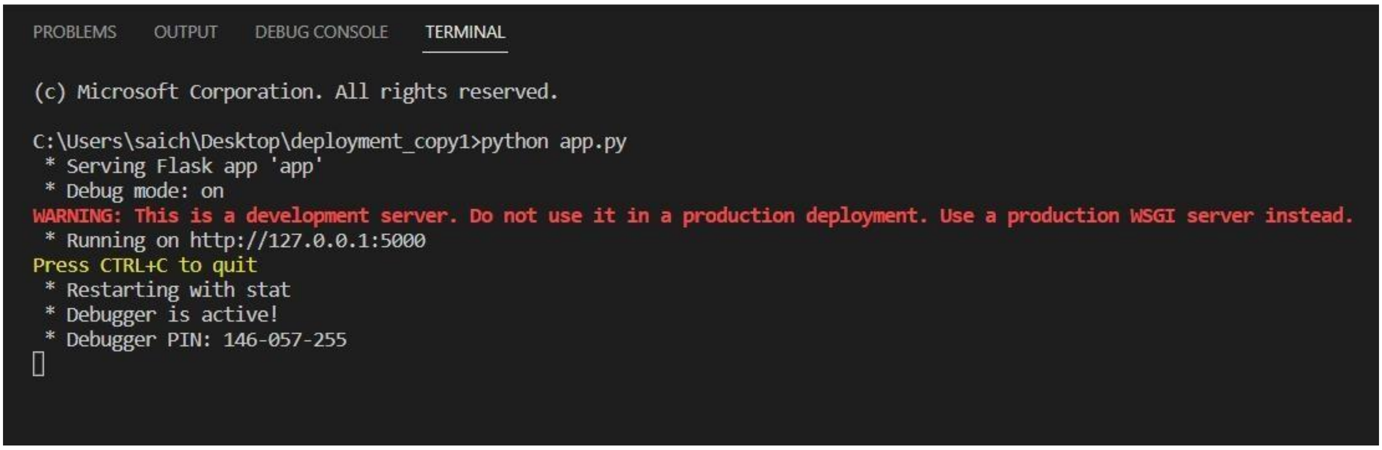
6.3 REPORTs from Jira:




Jira Software
Your work
Projects
Filters
Dashboards
People
Apps
Create
Search
Efficient Water Quality...
Software project
PLANNING
Roadmap
Backlog
Board
DEVELOPMENT
Code
Project pages
Add shortcut
You're in a team-managed project
Learn more
Projects / Efficient Water Quality Analysis and Prediction using Machine Learning
EWQAPUML Sprint 1
3 days remaining
Complete sprint
Epic
GROUP BY
None
Insights
TO DO 2 ISSUES
IN PROGRESS
DONE
Data Collection
DATA COLLECTION
EWQAPUML-5
Data Preprocessing
DATA COLLECTION
EWQAPUML-6


OCT
NOV
Sprints
EWQAPUML Sprint 1, EWQAPU...
EWQAPUML-1 Data Collection
EWQAPUML-2 Model Building
EWQAPUML-3 Training and Testing
EWQAPUML-4 Implementation of Web page

7 CODING & SOLUTIONING

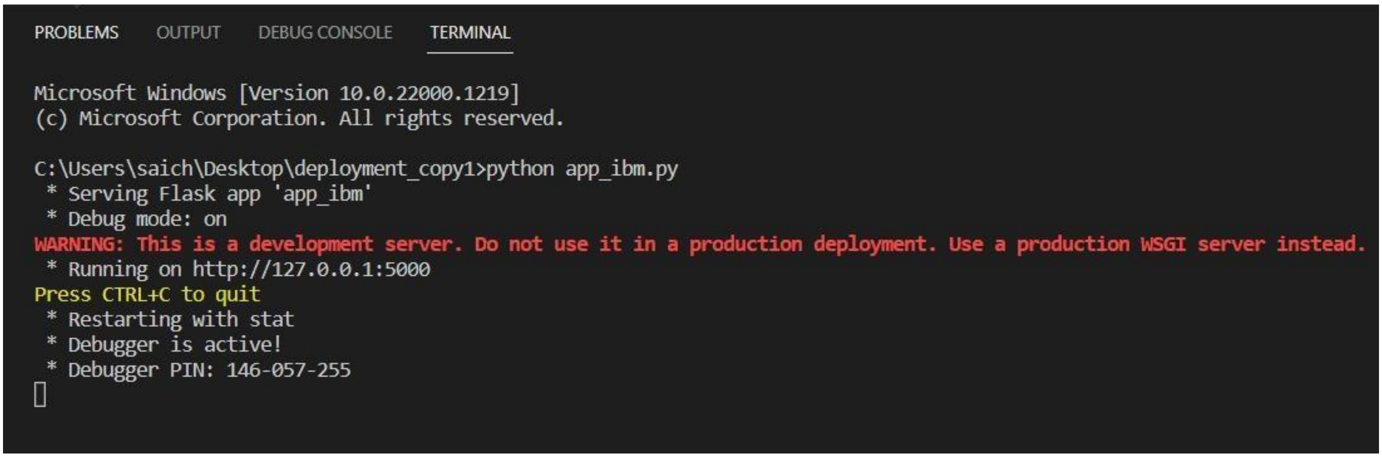


```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

(c) Microsoft Corporation. All rights reserved.

C:\Users\saich\Desktop\deployment_copy1>python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 146-057-255
```

7.2 FEATURE 2



```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Microsoft Windows [Version 10.0.22000.1219]
(c) Microsoft Corporation. All rights reserved.

C:\Users\saich\Desktop\deployment_copy1>python app_ibm.py
 * Serving Flask app 'app_ibm'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 146-057-255
```

8. TESTING

8.1 TEST CASES

Test case ID	Feature Type	Component	Test Scenario	Steps To Execute	Test Data	Expected Result	Actual Result	Status
Home Page_TC_OO1	Functional	Home Page	Verify user can see the prediction button and theinput columnsfor prediction	Verify the prediction button to analyze thequality	-	Input columns and theprediction button should be displayed	Working as expected	Pass
Home Page_TC_OO2	Functional	Home Page	Verify whetherthe page redirection is correct	Verify whether the redirection of page to about and info page are on clicking	-	Redirection to aboutpage and info page should be correct	Working as expected	Pass
Home Page_TC_OO3	UI	Home Page	Verify whetherthe logo image, background image, font alignment and size are correct	Verify whether the logo image, background image, font alignment andsize are correct	-	The logo image, backgroun d image, font alignment and size are correct	Workin g as expecte d	Pass
Info Page_TC_OO4	Functional	Info Page	Verify whetherthe info page displays all the data correctly	Verify whether theinfo page displays all the data correctly	-	The info page displays all the data correctly	Worki ng as expec ted	Pass
About Page_TC_OO5	Functional	About Page	Verify whetherthe about page displays all the data correctly	Verify whether theabout page displays all the data correctly	-	The about page displays all the data correctly	Worki ng as expec ted	Pass
Home Page_TC_006	Functional	Home Page	Verify whetherthe predicted value is display or not	On entering the input values the predicted value is display on home page		The predicted value should be displayed	Worki ng as expec ted	Pass

Test Scenarios:

1. **Verify user can see home page**
2. **Verify users can predict the WQI or not?**
3. **Verify users can navigate to the info page?**
4. **Verify users can enter values?**
5. **Verify prediction data is displayed or not?**
6. **Verify users can enter any text?**
7. **Verify users can see the prediction value and result?**

8.2 USER ACCEPTANCE TESTING

1. Purpose of Document

The purpose of this document is to briefly explain the test coverage and open issues of the Efficient Water Quality Analysis and Prediction using Machine Learning project at the time of the release to User Acceptance Testing (UAT).

2. Defect Analysis

This report shows the number of resolved or closed bugs at each severity level, and how they were resolved

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	10	4	2	3	20
Duplicate	1	0	3	0	4
External	2	3	0	1	6
Fixed	11	2	4	20	37
Not Reproduced	0	0	1	0	1
Skipped	0	0	1	1	2
Won't Fix	0	5	2	1	8
Totals	24	14	13	26	77

3. Test Case Analysis

This report shows the number of test cases that have passed, failed, and untested

Section	Total Cases	Not Tested	Fail	Pass
Home Page	7	0	0	7
Client Application	51	0	0	51
Prediction	2	0	0	2
Pop ups	3	0	0	3
URL port	9	0	0	9
Final Report Output	4	0	0	4
Redirecting	2	0	0	2

9 RESULT

9.1 MODEL PERFORMANCE TESTING

<table>
<tr><th>S.No.</th><th>Parameter</th><th>Values</th><th>Screenshot</th></tr>
<tr><td>1.</td><td>Metrics</td><td>Regression Model:
MAE - , MSE - , RMSE - , R2 score -</td><td>Model Evaluation<pre>
In [37]: from sklearn import metrics
         print('MAE:',metrics.mean_absolute_error(y_test,y_pred))
         print('MSE:',metrics.mean_squared_error(y_test,y_pred))
         print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test,y_pred)))

         MAE: 0.4550025062656734
         MSE: 2.5859671077694255
         RMSE: 1.6080942471663238

In [38]: metrics.r2_score(y_test, y_pred)

Out[38]: 0.9759652869193766
</pre></td></tr>
<tr><td>2.</td><td>Tune the Model</td><td>Hyperparameter Tuning -
Validation Method -</td><td>Hyperparameter Tuning<pre>
In [ ]: from sklearn.model_selection import cross_val_score, GridSearchCV

In [ ]: param_grid = { 'bootstrap': [True], 'max_depth': [5, 10, None], 'max_features': ['auto', 'log2'], 'n_estimators': [5, 6, 7, 8,

In [ ]: rfr = RandomForestRegressor(random_state = 1)
        g_search = GridSearchCV(estimator = rfr, param_grid = param_grid,
                                cv = 3, n_jobs = 1, verbose = 0, return_train_score=True)

In [ ]: g_search.fit(x_train, y_train)
        print(g_search.best_params_)

        {'bootstrap': True, 'max_depth': 10, 'max_features': 'auto', 'n_estimators': 15}
</pre>Validation Method Cross validation<pre>
In [ ]: scores = cross_val_score(regressor, y_test, y_pred, cv=10, scoring='neg_mean_absolute_error')
        print(scores)

        [-0.88937508 -0.2277642 -0.62957576 -0.28678912 -0.52877112 -0.33818409
         -0.59450265 -0.16186615 -0.17046191 -1.16749981]
</pre></td></tr>
</table>

10. ADVANTAGES & DISADVANTAGES

ADVANTAGES

- Reliable one with the prediction accuracy.

- The future of water quality modeling seems to be very bright and remarkable

• Elective technique for AI to foresee water quality utilizing negligible and effectively accessible water quality boundaries.

• This project can be used in urban areas to predict the quality of the drinking water thereby preventing the spread of diseases such as dysentery, typhoid and cholera due to consumption of contaminated water.

• System is low cost and efficient.

DISADVANTAGES

• There needs to be a more user-centric approach towards tackling the water quality issues, by using user friendly tools and an interactive environment so that the solution actually benefits in tackling water quality issues.

• Not all models have been able to numerically predict the magnesium absorption ratio (MAR) and the permeability index (PI), so classification models may be able to improve the accuracy of predictions.

• Internet Connectivity and times may be a problem, since data won’t be updated.

11 CONCLUSION

Water is one of the most essential resources for survival and its quality is determined through WQI(Water quality Index). To this end most dataset related well-known components, such as Temperature, PH, DO(dissolved oxygen), conductivity, BOD(biochemical oxygen demand), nitratenan, fecal coliform etc., were collected.By developing and deploying resilient hardware and software we can analyze the drinking water.

By using Random Forest Regression algorithm we need to train the dataset and see the incremental improvement in prediction rate. The unnecessary data will be rejected. Analyze the data and interpret the output.Finally we test the data collected and predict the good condition of water.

We test the accuracy of the models with the test dataset that was formed in the previous phase and the most accurate model was figured out. The measurements of water quality parameters were scaled between 0 and 1 for better data handling.

12 FUTURE SCOPE

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.

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.

13 APPENDIX

SOURCE CODE

HTML CODE

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">
```

```
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>WATER QUALITY ANALYSIS</title>


    <link rel="stylesheet" type='text/css'
href="{{url_for('static',filename='style.css')}}">



</head>
<div class="container">
    <header class="header">
      <h1 id="title" class="text-center">WATER QUALITY
PREDICTOR</h1>
      <p id="description" class="description text-center">
        using random forest algorithm

      </p>
    </header>
    <form method='POST' action='/predict' id="survey-form" >


      <div class="form-group">
        <label id="name-label" for="name">ENTER YEAR</label>
        <input type="text"
          name="year" id="name"
          class="form-control"
```

```
        placeholder="Enter the
        year" required
      />
    </div>
    <div class="form-group">
      <label id="name-label" for="name">ENTER
TEMPERATURE</label>
      <input type="text"
        name="temperature" id="name"
        class="form-control"
        placeholder="Enter
        temperature" required

      />
    </div>
    <div class="form-group">
      <label id="name-label" for="name">ENTER DO</label>
      <input type="text" name="do"
        id="email" class="form-control"
        placeholder="Enter Dissolved
        Oxygen" required
      />


    </div>
```

```
    <div class="form-group">
      <label id="name-label" for="name">ENTER pH</label>
      <input type="text"
        name="ph" id="name"
        class="form-control"
        placeholder="Enter
        pH" required

      />
    </div>
    <div class="form-group">
      <label id="name-label" for="name">ENTER
CONDUCTIVITY</label>
      <input type="text"
        name="conductivity" id="name"
        class="form-control"
        placeholder="Enter the
        conductivity" required
      />
    </div>
    <div class="form-group">
      <label id="name-label" for="name">ENTER THE BOD</label>
```

```
  <input type="text" name="bod" id="name"
    class="form-control" placeholder="Enter
    Biochemical Oxygen Demand" required
  />
</div>
<div class="form-group">
  <label id="name-label" for="name">ENTER NI</label>
  <input type="text" name="ni"
    id="name" class="form-
    control" placeholder="Enter
    nitratenen" required
  />
</div>
<div class="form-group">
  <label id="name-label" for="name">ENTER Fec_col</label>
  <input type="text" name="fec"
    id="name" class="form-control"
    placeholder="Enter fecal
    coliform" required
  />
</div>
<div class="form-group">
  <label id="name-label" for="name">ENTER tot_col</label>
```

```
      <input type="text" name="tot"
        id="name" class="form-control"
        placeholder="Enter total
        coliform" required
      />
    </div>

    <div class="form-group">
      <button type="submit" id="submit" class="submit-button">
        SUBMIT
      </button>
    </div>
  </form>
 </div>
</html>
```

CSS CODE

```
body {
   font-family: 'Poppins', sans-serif;
 font-size: 1rem; font-weight: 400;
 line-height: 1.4; color: var(--
 color-pink); margin: 0; }

 body::before {
```

```
content: '';
position:
fixed; top: 0;
left: 0;
height: 100%; width: 100%; z-
index: -1; background: var(--color-
blue); background-image: linear-
gradient(
  115deg, rgba(26, 176,
  234, 0.2), rgba(26,
  176, 255, 0.2)
 ),
 url(https://i.pinimg.com/736x/32/b5/a3/32b5a3a70f89c9ee66d192a06e
 012
25d.jpg); background-size:
cover; filter: blur(1px);
background-repeat: no-
repeat; background-position:
center;
}

h1 { font-weight:
1400; line-height:
1.2;
```

```
}

p { font-size:
  1.125rem;
}

h1, p { margin-top: 0;
margin-bottom: 0.5rem;
}
label { display: flex;
align-items: center; font-
size: 1.125rem; margin-
bottom: 0.5rem; }

input, button, select,
textarea { margin: 0;
font-family: inherit;
font-size: inherit;
line-height: inherit;
font:black; }

button {
  border: none;
```

```
}
.container { width: 100%;
  margin: 3.125rem auto 0 auto;
}

@media (min-width: 576px) {
  .container { max-
    width: 540px;
  }
}

@media (min-width: 768px) {
  .container { max-
    width: 720px;
  }
}

.header { padding: 0
  0.625rem; margin-bottom:
  1.875rem;
}

.description {
```

```
  font-style: italic; font-weight: 200; text-
  shadow: 1px 1px 1px rgba(0, 0, 0, 0.4);
}

.text-center { text-
  align: center;


}



.form-group { margin: 0 auto
  1.25rem auto; padding:
  0.25rem; font:black;
  margin:auto;
}

.form-control { display:
  block; width: 90%; height:
  2.375rem; padding:
  0.375rem 0.75rem;
  background:white;
  background-clip: padding-
  box; border: 1px solid
```

```
  #ced4da; border-radius:
  0.25rem; margin:auto;
  transition: border-color 0.15s ease-in-out, box-shadow 0.15s
ease-in-out;
 }

 .form-control:focus { border-color:#3f6139;
  outline: 0; box-shadow: 0 0 0 0.2rem
  rgba(63,69,105,0.25); margin:auto;
 }

 .input-textarea { min-
  height: 120px; width:
  100%; padding:
  0.625rem; resize:
  vertical;
 }

 .submit-button { display: block; width: 30%;
  padding: 1rem; background: white; color:
  white; font-size:1.150rem; background-image:
  linear-gradient(blue, grey); border-radius:
  50px; cursor: pointer; margin:auto;
```

}

Efficient Water Quality Analysis using Jupyter Notebook

In [1]:

```
import os, types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.
# You might want to remove those credentials before you share the notebook.
cos_client = ibm_boto3.client(service_name='s3',
    ibm_api_key_id='UmuWJ1wfB9i_GshiIUnLdsIVQJevwFvzN4n8kyoEkXHL',
    ibm_auth_endpoint="https://iam.cloud.ibm.com/oidc/token",
    config=Config(signature_version='oauth'),
    endpoint_url='https://s3.private.us.cloud-object-storage.appdomain.cloud')

bucket = 'waterqualitypredictor-donotdelete-pr-tanesbvu4upv3x'
object_key = 'water_data.csv'

body = cos_client.get_object(Bucket=bucket,Key=object_key)['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

df = pd.read_csv(body)
df.head()
```

Out[1]:

	STATION CODE	LOCATIONS	STATE	Temp	D.O. (mg/l)	PH	CONDUCTIVITY (µmhos/cm)	B.O.D. (mg/l)	NITRATENAN N+ NITRITENANN (mg/l)	FECAL COLIFORM (MPN/100ml)	TOTAL COLIFORM (MPN/100ml)Mean	year
0	1393	DAMANGANGA AT D/S OF MADHUBAN, DAMAN	DAMAN & DIU	30.6	6.7	7.5	203	NAN	0.1	11	27	2014
1	1399	ZUARI AT D/S OF PT. WHERE KUMBARJRIA CANAL JOI...	GOA	29.8	5.7	7.2	189	2	0.2	4953	8391	2014
2	1475	ZUARI AT PANCHAWADI	GOA	29.5	6.3	6.9	179	1.7	0.1	3243	5330	2014
3	3181	RIVER ZUARI AT BORIM BRIDGE	GOA	29.7	5.8	6.9	64	3.8	0.5	5382	8443	2014
4	3182	RIVER ZUARI AT MARCAIM JETTY	GOA	29.5	5.8	7.3	83	1.9	0.4	3428	5500	2014

In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
from sklearn.neighbors import LocalOutlierFactor
from scipy.stats import probplot
from scipy.stats import zscore
```

In [3]:
```
df = df.iloc[0:1900, :]
df.shape
```

Out[3]: (1900, 12)

In [4]:
```
# Checking for datatypes of the dataset
df.dtypes
```

Out[4]:
```
STATION CODE                          object
LOCATIONS                             object
STATE                                 object
Temp                                  object
D.O. (mg/l)                           object
PH                                    object
CONDUCTIVITY (µmhos/cm)               object
B.O.D. (mg/l)                         object
NITRATENAN N+ NITRITENANN (mg/l)      object
FECAL COLIFORM (MPN/100ml)            object
TOTAL COLIFORM (MPN/100ml)Mean        object
year                                   int64
dtype: object
```

In [5]:
```
df = df.rename(columns={"D.O. (mg/l)": "DO", "CONDUCTIVITY (µmhos/cm)": "Conductivity", "B.O.D. (mg/l)": "BOD", "NITRATENAN N+ NI
```

In [6]:
```
# Converting object data type to numeric
def convert_to_numeric(df):
    num_col = df.shape[1]
    # Start from index 3
    for index in range(3, num_col):
        col_name = df.iloc[:, index].name
        df[col_name] = pd.to_numeric(df[col_name], errors="coerce")
    return df

df = convert_to_numeric(df)
df.dtypes
```

Out[6]:
```
STATION CODE     object
LOCATIONS        object
STATE            object
Temp            float64
DO              float64
PH              float64
Conductivity    float64
BOD             float64
NI              float64
Fec_col         float64
Tot_col         float64
year              int64
dtype: object
```

In [7]:
```
def convert_to_nan(df):
    n_col = df.shape[1]
    for index in range(n_col):
        df.iloc[:, index]  = df.iloc[:, index].replace("NAN", np.nan)
    return df

df = convert_to_nan(df)
```

In [8]:
```
# Checking for missing values
df.isnull().sum().sort_values()
```

Out[8]:
```
year              0
PH                7
Conductivity     24
DO               30
BOD              42
Temp             89
STATION CODE    120
Tot_col         130
LOCATIONS       183
NI              189
Fec_col         280
STATE           670
dtype: int64
```

In [9]:
```
# Replacing NULL values with median of column
# Selecting numeric data
df_num = df.select_dtypes(exclude="object")
df_num_col = df_num.columns
imputer = SimpleImputer(strategy="median")

df_num = imputer.fit_transform(df_num)
df_num = pd.DataFrame(df_num, columns=df_num_col)
```

In [10]:
```
# Filling Categorical missing values
df_cat = df.select_dtypes(include="object")
df_cat.isnull().sum()
```

Out[10]:
```
STATION CODE    120
LOCATIONS       183
STATE           670
dtype: int64
```

In [11]:
```
# Here we can fill these values by obeserving other attributes
# Example -
pd.set_option('mode.chained_assignment', None)
df_cat_copy = df_cat.copy()

df_cat_copy[df_cat_copy["STATION CODE"] == "1330"]
# Station Code with value 1330 will have Location - TAMBIRAPARANI which belongs in STATE - TAMIL NADU
# I can replace all the NAN occurences in STATE with TAMILNADU
df_cat_copy["STATE"][df_cat_copy["STATION CODE"] == "1330"] = df_cat_copy["STATE"][df_cat_copy["STATION CODE"] == "1330"].fillna(

df_cat_copy[df_cat_copy["STATION CODE"] == "1330"]
```

Out[11]:

	STATION CODE	LOCATIONS	STATE
166	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
424	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
677	1330	TAMBIRAPARANI AT ARUMUGANERI	TAMILNADU
1168	1330	TAMBIRAPARANI AT ARUMUGANERI	TAMILNADU
1351	1330	NaN	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU
1513	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
1626	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU

In [12]:
```python
def fill_locations(df_cat):
    location_null = df_cat[df_cat["LOCATIONS"].isnull()]
    location_null_indices = location_null.index
    for index in location_null_indices:
        state_value = location_null["STATE"][index]
        location_null["LOCATIONS"][index] = state_value
        location_null["STATE"][index] = np.nan
    df_cat[df_cat["LOCATIONS"].isnull()] = location_null
    return

fill_locations(df_cat_copy)
df_cat_copy[df_cat_copy["STATION CODE"] == "1330"]
```

Out[12]:

	STATION CODE	LOCATIONS	STATE
166	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
424	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
677	1330	TAMBIRAPARANI AT ARUMUGANERI	TAMILNADU
1168	1330	TAMBIRAPARANI AT ARUMUGANERI	TAMILNADU
1351	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	NaN
1513	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
1626	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
1745	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU

In [13]:
```python
df_cat_copy[df_cat_copy["LOCATIONS"] == "TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU"]
```

Out[13]:

	STATION CODE	LOCATIONS	STATE
166	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
424	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
1351	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	NaN
1513	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
1626	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
1745	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
1896	NaN	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	NaN

In [14]:
```python
def fill_code(df_cat):
    station_null = df_cat[df_cat["STATION CODE"].isnull()]
    station_null_indices = station_null.index
    for index in station_null_indices:
        stat_code = np.nan
        location_index = station_null["LOCATIONS"][index]
        code_at_location = df_cat["STATION CODE"][df_cat["LOCATIONS"] == location_index]
        for index_code in code_at_location.index:
            if (code_at_location[index_code] != np.nan):
                stat_code = code_at_location[index_code]
                break
        station_null["STATION CODE"][index] = stat_code
    df_cat[df_cat["STATION CODE"].isnull()] = station_null
    return

fill_code(df_cat_copy)
df_cat_copy[df_cat_copy["LOCATIONS"] == "TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU"]
```

Out[14]:

	STATION CODE	LOCATIONS	STATE
166	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
424	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
1351	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	NaN
1513	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
1626	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
1745	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
1896	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	NaN

In [15]:
```python
# Filling all state NAN values which have corresponding station code value
def fill_state(df_cat):
    station_code = df_cat["STATION CODE"].unique()
    for index in range(station_code.shape[0]):
        if (station_code[index] != np.nan):
            df_state = df_cat["STATE"][df_cat["STATION CODE"] == station_code[index]]
            state_values = df_cat["STATE"][df_cat["STATION CODE"] == station_code[index]]
            state = np.nan
            for index_state in range(state_values.shape[0]):
                if (state_values.iloc[index_state] != np.nan):
                    state = state_values.iloc[index_state]
                    break
```

```
            df_cat["STATE"][df_cat["STATION CODE"] == station_code[index]] = df_state_fill
    return
fill_state(df_cat_copy)
df_cat_copy[df_cat_copy["STATION CODE"] == "1330"]
```

Out[15]:

	STATION CODE	LOCATIONS	STATE
166	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
424	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
677	1330	TAMBIRAPARANI AT ARUMUGANERI	TAMILNADU
1168	1330	TAMBIRAPARANI AT ARUMUGANERI	TAMILNADU
1351	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
1513	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
1626	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
1745	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU
1896	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU

In [16]:
```
df_cat_copy.isnull().sum()
```

Out[16]:
```
STATION CODE     4
LOCATIONS        2
STATE           12
dtype: int64
```

In [17]:
```
df_cat_copy[df_cat_copy["STATE"].isnull()]
```

Out[17]:

	STATION CODE	LOCATIONS	STATE
260	NaN	NaN	NaN
431	NaN	NaN	NaN
1106	1207	KABBANI AT MUTHANKARA	NaN
1107	1208	BHAVANI AT ELACHIVAZHY	NaN
1650	2047	NNANCHOE (ATTAWA CHOE), CHANDIGARH	NaN
1651	2048	PATIALA KI RAO, CHANDIGARH	NaN
1652	2049	SUKHNA CHOE, CHANDIGARH	NaN
1770	2047	NNANCHOE (ATTAWA CHOE)	NaN
1771	2048	PATIALA KI RAO	NaN
1772	2049	SUKHNA CHOE	NaN
1784	NaN	DAMANGANGA AFTER CONFL. OF PIPARIA DRAIN, DAMAN	NaN
1785	NaN	DAMANGANGA AT CIRCUIT HOUSE, SILVASA, DADRA AN...	NaN

In [18]:
```
# The first Location KABBANI AT MUTHANKARA is in STATE Kerela
df_cat_copy["STATE"][1106] = "KERALA"
df_cat_copy["STATE"][1107] = "KERALA"
df_cat_copy["STATE"][1650] = "CHANDIGARH"
df_cat_copy["STATE"][1651] = "CHANDIGARH"
df_cat_copy["STATE"][1652] = "CHANDIGARH"
df_cat_copy["STATE"][1770] = "CHANDIGARH"
df_cat_copy["STATE"][1771] = "CHANDIGARH"
df_cat_copy["STATE"][1772] = "CHANDIGARH"
```

```
df_cat_copy["STATION CODE"][1784] = "0000" # I am setting this according to myself
df_cat_copy["STATION CODE"][1785] = "0000"
```

In [19]:
```
df_cat = df_cat_copy
df_cat.isnull().sum()
```

Out[19]:
```
STATION CODE    2
LOCATIONS       2
STATE           2
dtype: int64
```

In [20]:
```
df_num.isnull().sum()
```

Out[20]:
```
Temp            0
DO              0
PH              0
Conductivity    0
BOD             0
NI              0
Fec_col         0
Tot_col         0
year            0
dtype: int64
```

In [21]:
```
df_final = pd.concat([df_cat, df_num], axis=1)
df_final.isnull().sum()
```

Out[21]:
```
STATION CODE    2
LOCATIONS       2
STATE           2
Temp            0
DO              0
PH              0
Conductivity    0
BOD             0
NI              0
Fec_col         0
Tot_col         0
year            0
dtype: int64
```

In [22]:
```
# These are the samples which don't contain any attribute
# The filled attributes are median of corresponding columns
# So it is best to remove them
df_null = df_final[(df_final["STATION CODE"].isnull()) & (df_final["LOCATIONS"].isnull()) & (df_final["STATE"].isnull())]
df_null_indices = df_null.index
df_final.drop(df_null_indices, axis=0, inplace=True)
df_null
```

Out[22]:

	STATION CODE	LOCATIONS	STATE	Temp	DO	PH	Conductivity	BOD	NI	Fec_col	Tot_col	year
260	NaN	NaN	NaN	27.0	6.7	7.3	198.0	1.8965	0.52	233.0	465.0	2013.0
431	NaN	NaN	NaN	27.0	6.7	7.3	198.0	1.8965	0.52	233.0	465.0	2013.0

In [23]:
```
df_final.isnull().sum()
```

Out[23]:
```
STATION CODE    0
LOCATIONS       0
STATE           0
Temp            0
DO              0
PH              0
Conductivity    0
BOD             0
NI              0
Fec_col         0
Tot_col         0
year            0
dtype: int64
```

In [24]:
```
df_final.shape
```

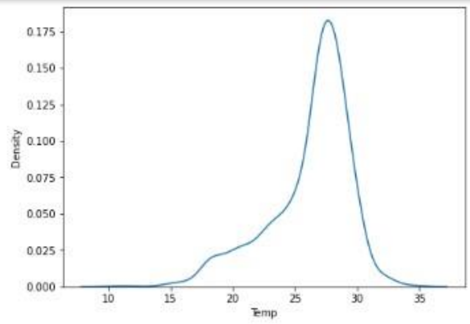
Out[24]:
```
(1898, 12)
```

In [26]:
```
# Plotting PDFs of all the numeric attributes in the dataset

df_num_final = df_final.select_dtypes(exclude="object")

def plot_kde(df):
    n_col = df.shape[1]
    for index in range(n_col):
        col_index = df.iloc[:, index]
        fig, ax = plt.subplots(1,1, figsize=(7, 5))
        sns.kdeplot(data=df, x=col_index.name)

plot_kde(df_num_final)
```



In [27]:
```
# Here, almost all kde plots are Gaussian Like
# Using Z-Score Normalization to detect outliers

df_num_final_norm = zscore(df_num_final, axis=0)


def indices_of_greater_than_3(df_norm):
    indices_arr = []
```

```
    n_col = df_norm.shape[1]
    for index in range(n_col):
        col_index = df_norm.iloc[: ,index]
        greater_than_3 = df_norm[col_index > 3]
        greater_than_3_index = greater_than_3.index
        indices_arr.extend(greater_than_3_index)
    return indices_arr

indices_arr = indices_of_greater_than_3(df_num_final_norm)
print("Number of outliers using Z-Score method-",len(indices_arr))
df_final.iloc[indices_arr, :]
```

```
Number of outliers using Z-Score method- 125
```

Out[27]:

	STATION CODE	LOCATIONS	STATE	Temp	DO	PH	Conductivity	BOD	NI	Fec_col	Tot_col	year
741	2880	NAMBUL RIVER AT BISHNUPUR	MANIPUR	28.0	8.2	7.6	112.0	2.1	0.52	233.0	31.0	2012.0
745	2856	THOUBAL RIVER AT YAIRIPOK, THOUBAL	MANIPUR	30.0	9.3	7.6	193.0	2.3	0.52	233.0	41.0	2012.0
37	2671	KUNDALIKA RIVER NEAR SALAV BRIDGE (SALINA ZONE...	MAHARASHTRA	25.3	5.3	7.7	24062.0	9.9	1.20	156.0	304.0	2014.0
88	2294	R KALLAI AT KALLAI BRIDGE	KERALA	26.3	3.7	7.7	32005.0	1.2	0.90	40000.0	60392.0	2014.0
108	2304	R MOGRAL AT MOGRAL BR.	KERALA	30.0	5.6	7.2	24360.0	2.1	0.30	92.0	447.0	2014.0
...	...	...	...	...	...	...	...	...	...	...	...	...
432	1023	GHAGGAR AT MUBARAKPUR REST HOUSE (PATIALA), PU...	PUNJAB	23.3	5.5	7.2	636.0	9.7	4.00	1328.0	4975.0	2013.0
685	1023	GHAGGAR AT MUBARAKPUR REST HOUSE (PATIALA)	PUNJAB	21.0	5.5	7.4	635.0	8.8	5.08	1400.0	5500.0	2012.0
172	3023	VASISTA AT SALEM, D/S OF SAGO INDUSRIES EFFLUE...	TAMILNADU	24.3	0.9	7.6	2039.0	104.5	0.90	272521616.0	511090873.0	2014.0
432	1023	GHAGGAR AT MUBARAKPUR REST HOUSE (PATIALA), PU...	PUNJAB	23.3	5.5	7.2	636.0	9.7	4.00	1328.0	4975.0	2013.0
685	1023	GHAGGAR AT MUBARAKPUR REST HOUSE (PATIALA)	PUNJAB	21.0	5.5	7.4	635.0	8.8	5.08	1400.0	5500.0	2012.0

125 rows × 12 columns

In [28]:

```
df_final.drop(indices_arr, axis=0, inplace=True)
df_final.shape
```

Out[28]:

```
(1785, 12)
```

In [29]:

```
# Calculating Water Quality Index of each sample
df_num_final = df_final.select_dtypes(exclude="object")
# Dropping year and Temp attribute because they are not used for computing WQI
df_num_final.drop(["year", "Temp"], axis=1, inplace=True)

# Weight Vector(wi)
wi = np.array([0.2213, 0.2604, 0.0022, 0.4426, 0.0492, 0.0221, 0.0022])

# Standard values of parameters(si)
si = np.array([10, 8.5, 1000, 5, 45, 100, 1000])

# Ideal values of paramters(vIdeal)
vIdeal = np.array([14.6, 7, 0, 0, 0, 0, 0])
```

```python
def calc_wqi(sample):
    wqi_sample = 0
    num_col = 7
    for index in range(num_col):
        v_index = sample[index] # Obeserved value of sample at index
        v_index_ideal = vIdeal[index] # Ideal value of obeserved value
        w_index = wi[index] # weight of corresponding parameter of obeserved value
        std_index = si[index] # Standard value recommended for obeserved value
        q_index = (v_index - v_index_ideal) / (std_index - v_index_ideal)
        q_index = q_index * 100 # Final qi value of obeserved value
        wqi_sample += q_index*w_index
    return wqi_sample
```

In [30]:
```python
# Computing WQI for the whole dataset
def calc_wqi_for_df(df):
    wqi_arr = []
    for index in range(df.shape[0]):
        index_row = df.iloc[index, :]
        wqi_row = calc_wqi(index_row)
        wqi_arr.append(wqi_row)
    return wqi_arr
```

In [31]:
```python
wqi_arr = calc_wqi_for_df(df_num_final)
# Converting oridnary array to numpy array
wqi_arr = np.array(wqi_arr)
wqi_arr = np.reshape(wqi_arr, (-1, 1))

# Resetting index values of the dataframes
wqi_arr_df = pd.DataFrame(wqi_arr, columns=["WQI"]).reset_index()
df_final = df_final.reset_index()
```

In [32]:
```python
# Combining dataframe of WQI and dataframe of attributes
df_wqi = pd.concat([df_final, pd.DataFrame(wqi_arr, columns=["WQI"])], axis=1)
df_wqi.drop("index", axis=1, inplace=True)
df_wqi.shape
```

Out[32]: (1785, 13)

In [33]:
```python
# These are samples with negative WQI
df_wqi[(df_wqi["WQI"] < 0)]
```

Out[33]:

	STATION CODE	LOCATIONS	STATE	Temp	DO	PH	Conductivity	BOD	NI	Fec_col	Tot_col	year	WQI
196	3375	LUKHA RIVER AT MYNDIHATI (TRIBUTARY OF LUNAR)	MEGHALAYA	20.5	6.7	2.7	1350.0	3.3000	1.10	7.0	16.0	2014.0	-6.855044
231	2	DAMANGANGA AT D/S OF MADHUBAN, DAMAN	DAMAN & DIU	27.0	6.7	0.0	208.0	1.8965	0.52	233.0	465.0	2013.0	-61.372099
234	1865	RIVER DHADAR AT KOTHADA	GUJARAT	27.0	6.7	0.0	506.0	1.8965	6.00	26.0	227.0	2013.0	-65.334452
446	3375	LUKHA RIVER	MEGHALAYA	21.3	6.8	2.7	1074.0	3.2000	2.33	4.0	11.0	2013.0	-8.214971
719	3375	LUKHA RIVER AT MYNDIHATI (TRIBUTARY OF LUNAR)	MEGHALAYA	25.0	6.9	2.6	1072.0	3.2000	1.17	3.0	21.0	2012.0	-10.579224

In [34]:
```python
# Removing the samples with negative WQI
df_neg_indices = df_wqi[(df_wqi["WQI"] < 0)].index
df_wqi.drop(df_neg_indices, axis=0, inplace=True)
```

In [35]:
```
df_wqi["WQI clf"] = df_wqi["WQI"].apply(lambda x: (4 if (x <= 25)
                                            else(3 if (26<=x<=50)
                                            else(2 if (51<=x<=75)
                                            else(2 if (76<=x<=100)
                                            else 0)))))
```

In [36]:
```
df_wqi.tail()
```

Out[36]:

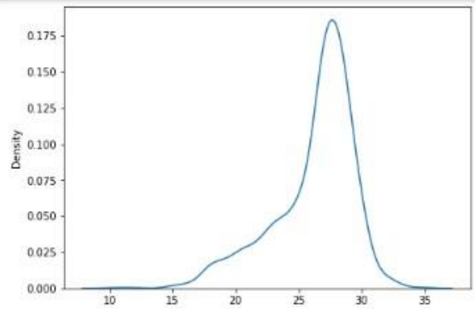
	STATION CODE	LOCATIONS	STATE	Temp	DO	PH	Conductivity	BOD	NI	Fec_col	Tot_col	year	WQI	WQI clf
1780	1329	TAMBIRAPARANI AT RAIL BDG. NR. AMBASAMUDAM, TA...	TAMILNADU	27.0	7.4	7.00	88.5	0.977	0.186	27.0	105.0	2005.0	43.946271	3
1781	1330	TAMBIRAPARANI AT ARUMUGANERI, TAMILNADU	TAMILNADU	27.0	6.6	7.81	603.2	2.675	0.263	40.0	191.0	2005.0	77.315135	2
1782	1450	PALAR AT VANIYAMBADI WATER SUPPLY HEAD WORK, T...	TAMILNADU	28.0	6.6	7.49	571.5	2.091	0.256	151.0	273.0	2005.0	69.053768	2
1783	1403	GUMTI AT U/S SOUTH TRIPURA,TRIPURA	TRIPURA	28.0	5.4	7.16	75.8	2.092	0.520	404.0	513.0	2005.0	74.670773	2
1784	1404	GUMTI AT D/S SOUTH TRIPURA, TRIPURA	TRIPURA	30.0	5.4	7.37	104.8	1.802	0.215	456.0	557.0	2005.0	76.881207	2

In [37]:
```
df_wqi.describe()
```

Out[37]:

	Temp	DO	PH	Conductivity	BOD	NI	Fec_col	Tot_col	year	WQI	WQI clf
count	1780.000000	1780.000000	1780.000000	1780.000000	1780.000000	1780.000000	1780.000000	1780.000000	1780.000000	1780.000000	1780.000000
mean	26.241931	6.432263	7.228045	1006.325691	3.961788	1.119716	2264.420646	7242.598876	2010.380337	130.303409	1.668539
std	3.231044	1.258875	0.582761	2764.600752	7.130494	1.629483	8259.474360	40230.193629	2.704223	222.261326	1.024280
min	10.000000	0.000000	2.900000	11.000000	0.100000	0.000000	0.000000	0.000000	2005.000000	16.795814	0.000000
25%	25.000000	6.000000	6.900000	83.000000	1.100000	0.263000	46.000000	108.750000	2008.000000	54.641036	2.000000
50%	27.000000	6.700000	7.224500	183.000000	1.800000	0.520000	233.000000	465.000000	2011.000000	69.840286	2.000000
75%	28.200000	7.200000	7.600000	489.250000	3.400000	1.100000	672.500000	1650.000000	2013.000000	94.348696	2.000000
max	35.000000	10.000000	9.010000	18569.000000	88.000000	13.200000	150250.000000	967500.000000	2014.000000	3524.421534	4.000000

In [38]:
```
plot_kde(df_wqi.select_dtypes(exclude="object"))
```



In [39]:
```
df_wqi.describe()
```

Out[39]:

	Temp	DO	PH	Conductivity	BOD	NI	Fec_col	Tot_col	year	WQI	WQI clf
count	1780.000000	1780.000000	1780.000000	1780.000000	1780.000000	1780.000000	1780.000000	1780.000000	1780.000000	1780.000000	1780.000000
mean	26.241931	6.432263	7.228045	1006.325691	3.961788	1.119716	2264.420646	7242.598876	2010.380337	130.303409	1.668539
std	3.231044	1.258875	0.582761	2764.600752	7.130494	1.629483	8259.474360	40230.193629	2.704223	222.261326	1.024280
min	10.000000	0.000000	2.900000	11.000000	0.100000	0.000000	0.000000	0.000000	2005.000000	16.795814	0.000000
25%	25.000000	6.000000	6.900000	83.000000	1.100000	0.263000	46.000000	108.750000	2008.000000	54.641036	2.000000
50%	27.000000	6.700000	7.224500	183.000000	1.800000	0.520000	233.000000	465.000000	2011.000000	69.840286	2.000000
75%	28.200000	7.200000	7.600000	489.250000	3.400000	1.100000	672.500000	1650.000000	2013.000000	94.348696	2.000000
max	35.000000	10.000000	9.010000	18569.000000	88.000000	13.200000	150250.000000	967500.000000	2014.000000	3524.421534	4.000000

In [40]:
```
features = list(df_wqi.columns)[3:11]
data_f = df_wqi[features]
data_f.describe()
```

Out[40]:

	Temp	DO	PH	Conductivity	BOD	NI	Fec_col	Tot_col
count	1780.000000	1780.000000	1780.000000	1780.000000	1780.000000	1780.000000	1780.000000	1780.000000
mean	26.241931	6.432263	7.228045	1006.325691	3.961788	1.119716	2264.420646	7242.598876
std	3.231044	1.258875	0.582761	2764.600752	7.130494	1.629483	8259.474360	40230.193629
min	10.000000	0.000000	2.900000	11.000000	0.100000	0.000000	0.000000	0.000000
25%	25.000000	6.000000	6.900000	83.000000	1.100000	0.263000	46.000000	108.750000
50%	27.000000	6.700000	7.224500	183.000000	1.800000	0.520000	233.000000	465.000000
75%	28.200000	7.200000	7.600000	489.250000	3.400000	1.100000	672.500000	1650.000000
max	35.000000	10.000000	9.010000	18569.000000	88.000000	13.200000	150250.000000	967500.000000

In [41]:
```
features = list(df_wqi.columns)[:]
data_cluster = df_wqi['WQI clf']
data_cluster.describe()
```

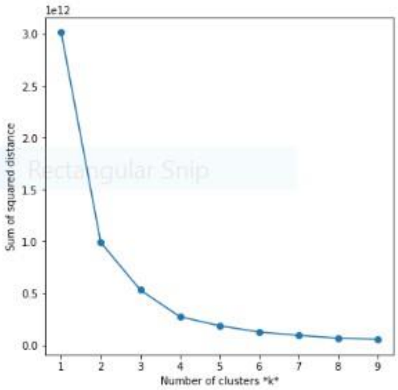
Out[41]:
```
count    1780.000000
mean        1.668539
std         1.024280
min         0.000000
25%         2.000000
50%         2.000000
75%         2.000000
max         4.000000
Name: WQI clf, dtype: float64
```

In [42]:
```
# normalize data
import pandas as pd
from sklearn import preprocessing
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
import seaborn as sns
```

```python
sse = []
list_k = list(range(1, 10))

for k in list_k:
    km = KMeans(n_clusters=k)
    km.fit(data_f)
    sse.append(km.inertia_)

# Plot sse against k
plt.figure(figsize=(6, 6))
plt.plot(list_k, sse, '-o')
plt.xlabel(r'Number of clusters *k*')
plt.ylabel('Sum of squared distance');
```



In [43]:
```python
data_f
```

Out[43]:

	Temp	DO	PH	Conductivity	BOD	NI	Fec_col	Tot_col
0	30.6	6.7	7.50	203.0	1.8965	0.100	11.0	27.0
1	29.8	5.7	7.20	189.0	2.0000	0.200	4953.0	8391.0
2	29.5	6.3	6.90	179.0	1.7000	0.100	3243.0	5330.0
3	29.7	5.8	6.90	64.0	3.8000	0.500	5382.0	8443.0
4	29.5	5.8	7.30	83.0	1.9000	0.400	3428.0	5500.0
...	...	...	...	...	...	...	...	...
1780	27.0	7.4	7.00	88.5	0.9770	0.186	27.0	105.0
1781	27.0	6.6	7.81	603.2	2.6750	0.263	40.0	191.0
1782	28.0	6.6	7.49	571.5	2.0910	0.256	151.0	273.0
1783	28.0	5.4	7.16	75.8	2.0920	0.520	404.0	513.0
1784	30.0	5.4	7.37	104.8	1.8020	0.215	456.0	557.0

1780 rows × 8 columns

In [44]:
```
Y = data_cluster
```

In [45]:
```
features = list(df_wqi.columns)[3:11]
X = df_wqi[features]
X.describe()
X.dtypes
```

Out[45]:
```
Temp            float64
DO              float64
PH              float64
Conductivity    float64
BOD             float64
NI              float64
Fec_col         float64
Tot_col         float64
dtype: object
```

In [46]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled
```

Out[46]:
```
array([[ 1.34919038,  0.21273941,  0.46679752, ..., -0.62596683,
        -0.27290525, -0.1794082 ],
       [ 1.10152283, -0.58184389, -0.04813777, ..., -0.56458042,
         0.32560606,  0.02855377],
       [ 1.0086475 , -0.10509391, -0.56307306, ..., -0.62596683,
         0.11851291, -0.04755474],
       ...,
       [ 0.54427084,  0.13328108,  0.44963301, ..., -0.53020403,
        -0.25595025, -0.17329167],
       [ 0.54427084, -0.82021888, -0.11679581, ..., -0.36814393,
        -0.22531016, -0.16732433],
       [ 1.16343972, -0.82021888,  0.24365889, ..., -0.55537246,
        -0.21901259, -0.16623031]])
```

In [47]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, Y, test_size=0.2, random_state=30)
```

In [48]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix,accuracy_score, classification_report
# model = LinearRegression()
# # model = LogisticRegression(solver='liblinear')
# model.fit(X_train,y_train)
# model.score(X_test,y_test)
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X_train, y_train)
score = neigh.score(X_test,y_test)
print("score: ",score)
y_pred = neigh.predict(X_test)
print(classification_report(y_test,y_pred, zero_division=1))
```

```
score:  0.8146067415730337
              precision    recall  f1-score   support

           0       0.89      0.69      0.78        93
           2       0.80      0.92      0.86       199
           3       0.76      0.67      0.71        63
           4       1.00      0.00      0.00         1

    accuracy                           0.81       356
   macro avg       0.86      0.57      0.59       356
weighted avg       0.82      0.81      0.81       356
```

In [49]:
```
from ibm_watson_machine_learning import APIClient
import json
```

In [50]:
```
wml_credentials = {
    "apikey":"9nW4DKjV9Et1NSwVo3FX0bR61av15QRQwc993a9Tpifp",
    "url":"https://us-south.ml.cloud.ibm.com"
}
#in the api key field enter the api key that u have generated.. generate a new api key fresh ah and dont use the same which u us
#prediction
```

In [52]:
```
wml_client = APIClient(wml_credentials)
wml_client.spaces.list()
#in the output below u might find two spaces.. use the space id which is been allocated for this water quality project
```

```
Note: 'limit' is not provided. Only first 50 records will be displayed if the number of records exceed 50
------------------------------------  -----------------------  ------------------------
ID                                    NAME                     CREATED
c870c0b4-f040-4f8f-8778-410c757a8e21  quality_prediction_water  2022-11-16T06:08:44.245Z
70dc3b4c-88ea-4e25-83d0-6b79e56b90d5  water_quality            2022-11-16T05:27:32.095Z
------------------------------------  -----------------------  ------------------------
```

In [55]:
```
SPACE_ID= "c870c0b4-f040-4f8f-8778-410c757a8e21"
```

In [56]:
```
wml_client.set.default_space(SPACE_ID)
```

Out[56]:
```
'SUCCESS'
```

In [57]:
```
wml_client.software_specifications.list(500)
```

```
-------------------------------  ------------------------------------  ----
NAME                             ASSET_ID                              TYPE
default_py3.6                    0062b8c9-8b7d-44a0-a9b9-46c416adcbd9  base
kernel-spark3.2-scala2.12        020d69ce-7ac1-5e68-ac1a-31189867356a  base
pytorch-onnx_1.3-py3.7-edt       069ea134-3346-5748-b513-49120e15d288  base
scikit-learn_0.20-py3.6          09c5a1d0-9c1e-4473-a344-eb7b665ff687  base
spark-mllib_3.0-scala_2.12       09f4cff0-90a7-5899-b9ed-1ef348aebdee  base
pytorch-onnx_rt22.1-py3.9        0b848dd4-e681-5599-be41-b5f6fccc6471  base
ai-function_0.1-py3.6            0cdb0f1e-5376-4f4d-92dd-da3b69aa9bda  base
shiny-r3.6                       0e6e79df-875e-4f24-8ae9-62dcc2148306  base
tensorflow_2.4-py3.7-horovod     1092590a-307d-563d-9b62-4eb7d64b3f22  base
pytorch_1.1-py3.6                10ac12d6-6b30-4ccd-8392-3e922c096a92  base
tensorflow_1.15-py3.6-ddl        111e41b3-de2d-5422-a4d6-bf776828c4b7  base
```

pytorch_1.1-py3.6	10ac12d6-6b30-4ccd-8392-3e922c096a92	base
tensorflow_1.15-py3.6-ddl	111e41b3-de2d-5422-a4d6-bf776828c4b7	base
autoai-kb_rt22.2-py3.10	125b6d9a-5b1f-5e8d-972a-b251688ccf40	base
runtime-22.1-py3.9	12b83a17-24d8-5082-900f-0ab31fbfd3cb	base
scikit-learn_0.22-py3.6	154010fa-5b3b-4ac1-82af-4d5ee5abbc85	base
default_r3.6	1b70aec3-ab34-4b87-8aa0-a4a3c8296a36	base
pytorch-onnx_1.3-py3.6	1bc6029a-cc97-56da-b8e0-39c3880dbbe7	base
kernel-spark3.3-r3.6	1c9e5454-f216-59dd-a20e-474a5cdf5988	base
pytorch-onnx_rt22.1-py3.9-edt	1d362186-7ad5-5b59-8b6c-9d0880bde37f	base
tensorflow_2.1-py3.6	1eb25b84-d6ed-5dde-b6a5-3fbdf1665666	base
spark-mllib_3.2	20047f72-0a98-58c7-9ff5-a77b012eb8f5	base
tensorflow_2.4-py3.8-horovod	217c16f6-178f-56bf-824a-b19f20564c49	base
runtime-22.1-py3.9-cuda	26215f05-08c3-5a41-a1b0-da66306ce658	base
do_py3.8	295addb5-9ef9-547e-9bf4-92ae3563e720	base
autoai-ts_3.8-py3.8	2aa0c932-798f-5ae9-abd6-15e0c2402fb5	base
tensorflow_1.15-py3.6	2b73a275-7cbf-420b-a912-eae7f436e0bc	base
kernel-spark3.3-py3.9	2b7961e2-e3b1-5a8c-a491-482c8368839a	base
pytorch_1.2-py3.6	2c8ef57d-2687-4b7d-acce-01f94976dac1	base
spark-mllib_2.3	2e51f700-bca0-4b0d-88dc-5c6791338875	base
pytorch-onnx_1.1-py3.6-edt	32983cea-3f32-4400-8965-dde874a8d67e	base
spark-mllib_3.0-py37	36507ebe-8770-55ba-ab2a-eafe787600e9	base
spark-mllib_2.4	390d21f8-e58b-4fac-9c55-d7ceda621326	base
autoai-ts_rt22.2-py3.10	396b2e83-0953-5b86-9a55-7ce1628a406f	base
xgboost_0.82-py3.6	39e31acd-5f30-41dc-ae44-60233c80306e	base
pytorch-onnx_1.2-py3.6-edt	40589d0e-7019-4e28-8daa-fb03b6f4fe12	base
pytorch-onnx_rt22.2-py3.10	40e73f55-783a-5535-b3fa-0c8b94291431	base
default_r36py38	41c247d3-45f8-5a71-b065-8580229facf0	base
autoai-ts_rt22.1-py3.9	4269d26e-07ba-5d40-8f66-2d495b0c71f7	base
autoai-obm_3.0	42b92e18-d9ab-567f-988a-4240ba1ed5f7	base
pmml-3.0_4.3	493bcb95-16f1-5bc5-bee8-81b8af80e9c7	base
spark-mllib_2.4-r_3.6	49403dff-92e9-4c87-a3d7-a42d0021c095	base
xgboost_0.90-py3.6	4ff8d6c2-1343-4c18-85e1-689c965304d3	base
pytorch-onnx_1.1-py3.6	50f95b2a-bc16-43bb-bc94-b0bed208c60b	base
autoai-ts_3.9-py3.8	52c57136-80fa-572e-8728-a5e7cbb42cde	base
spark-mllib_2.4-scala_2.11	55a70f99-7320-4be5-9fb9-9edb5a443af5	base
spark-mllib_3.0	5c1b0ca2-4977-5c2e-9439-ffd44ea8ffe9	base
autoai-obm_2.0	5c2e37fa-80b8-5e77-840f-d912469614ee	base
spss-modeler_18.1	5c3cad7e-507f-4b2a-a9a3-ab53a21dee8b	base
cuda-py3.8	5d3232bf-c86b-5df4-a2cd-7bb870a1cd4e	base
autoai-kb_3.1-py3.7	632d4b22-10aa-5180-88f0-f52dfb6444d7	base
pytorch-onnx_1.7-py3.8	634d3cdc-b562-5bf9-a2d4-ea90a478456b	base
spark-mllib_2.3-r_3.6	6586b9e3-ccd6-4f92-900f-0f8cb2bd6f0c	base
tensorflow_2.4-py3.7	65e171d7-72d1-55d9-8ebb-f813d620c9bb	base
spss-modeler_18.2	687eddc9-028a-4117-b9dd-e57b36f1efa5	base
pytorch-onnx_1.2-py3.6	692a6a4d-2c4d-45ff-a1ed-b167ee55469a	base
spark-mllib_2.3-scala_2.11	7963efe5-bbec-417e-92cf-0574e21b4e8d	base
spark-mllib_2.4-py37	7abc992b-b685-532b-a122-a396a3cdbaab	base
caffe_1.0-py3.6	7bb3dbe2-da6e-4145-918d-b6d84aa93b6b	base
pytorch-onnx_1.7-py3.7	812c6631-42b7-5613-982b-02098e6c909c	base
cuda-py3.6	82c79ece-4d12-40e6-8787-a7b9e0f62770	base
tensorflow_1.15-py3.6-horovod	8964680e-d5e4-5bb8-919b-8342c6c0dfd8	base
hybrid_0.1	8c1a58c6-62b5-4dc4-987a-df751c2756b6	base
pytorch-onnx_1.3-py3.7	8d5d8a87-a912-54cf-81ec-3914adaa988d	base
caffe-ibm_1.0-py3.6	8d863266-7927-4d1e-97d7-56a7f4c0a19b	base
spss-modeler_17.1	902d0051-84bd-4af6-ab6b-8f6aa6fdeabb	base
do_12.10	9100fd72-8159-4eb9-8a0b-a87e12eefa36	base
do_py3.7	9447fa8b-2051-4d24-9eef-5acb0e3c59f8	base
spark-mllib_3.0-r_3.6	94bb6052-c837-589d-83f1-f4142f219e32	base
cuda-py3.7-opence	94e9652b-7f2d-59d5-ba5a-23a414ea488f	base
nlp-py3.8	96e60351-99d4-5a1c-9cc0-473ac1b5a864	base
cuda-py3.7	9a44990c-1aa1-4c7d-baf8-c4099011741c	base
hybrid_0.2	[illegible]	base

```
cuda-py3.7                          9a44990c-1aa1-4c70-baf8-c4099011741c  base
hybrid_0.2                          9b3f9040-9cee-4ead-8d7a-780600f542f7  base
spark-mllib_3.0-py38                9f7a8fc1-4d3c-5e65-ab90-41fa8de2d418  base
autoai-kb_3.3-py3.7                 a545cca3-02df-5c61-9e88-998b09dc79af  base
spark-mllib_3.0-py39                a6082a27-5acc-5163-b02c-6b96916eb5e0  base
runtime-22.1-py3.9-do               a7e7dbf1-1d03-5544-994d-e5ec845ce99a  base
default_py3.8                       ab9e1b80-f2ce-592c-a7d2-4f2344f77194  base
tensorflow_rt22.1-py3.9             acd9c798-6974-5d2f-a657-ce06e986df4d  base
kernel-spark3.2-py3.9               ad7033ee-794e-58cf-812e-a95f4b64b207  base
autoai-obm_2.0 with Spark 3.0       af10f35f-69fa-5d66-9bf5-acb58434263a  base
default_py3.7_opence                c2057dd4-f42c-5f77-a02f-72bdbd3282c9  base
tensorflow_2.1-py3.7                c4032338-2a40-500a-beef-b01ab2667e27  base
do_py3.7_opence                     cc8f8976-b74a-551a-bb66-6377f8d865b4  base
spark-mllib_3.3                     d11f2434-4fc7-58b7-8a62-755da64fdaf8  base
autoai-kb_3.0-py3.6                 d139f196-e04b-5d8b-9140-9a10ca1fa91a  base
spark-mllib_3.0-py36                d82546d5-dd78-5fbb-9131-2ec309bc56ed  base
autoai-kb_3.4-py3.8                 da9b39c3-758c-5a4f-9cfd-457dd4d8c395  base
kernel-spark3.2-r3.6                db2fe4d6-d641-5d05-9972-73c654c60e0a  base
autoai-kb_rt22.1-py3.9              db6afe93-665f-5910-b117-d879897404d9  base
tensorflow_rt22.1-py3.9-horovod     dda170cc-ca67-5da7-9b7a-cf84c6987fae  base
autoai-ts_1.0-py3.7                 deef04f0-0c42-5147-9711-89f9904299db  base
tensorflow_2.1-py3.7-horovod        e384fce5-fdd1-53f8-bc71-11326c9c635f  base
default_py3.7                       e4429883-c883-42b6-87a8-f419d64088cd  base
do_22.1                             e51999ba-6452-5f1f-8287-17228b88b652  base
autoai-obm_3.2                      eae86aab-da30-5229-a6a6-1d0d4e368983  base
tensorflow_rt22.2-py3.10            f65bd165-f057-55de-b5cb-f97cf2c0f393  base
do_20.1                             f686cdd9-7904-5f9d-a732-01b0d6b10dc5  base
pytorch-onnx_rt22.2-py3.10-edt      f8a05d07-e7cd-57bb-a10b-23f1d4b837ac  base
scikit-learn_0.19-py3.6             f963fa9d-4bb7-5652-9c5d-8d9289ef6ad9  base
tensorflow_2.4-py3.8                fe185c44-9a99-5425-986b-59bd1d2eda46  base
------------------------------      ------------------------------------  ----
```

In [58]:
```python
import sklearn
sklearn.__version__
```

Out[58]: '1.0.2'

In [59]:
```python
MODEL_NAME = 'quality_prediction' #here the model name should be the name which u give for ur project
DEPLOYMENT_NAME = 'quality_prediction_water' #deployment name should be the same which is given in the deployments section
DEMO_MODEL = neigh
```

In [60]:
```python
software_spec_uid = wml_client.software_specifications.get_id_by_name('runtime-22.1-py3.9')
```

In [61]:
```python
# Setup model meta
model_props = {
    wml_client.repository.ModelMetaNames.NAME: MODEL_NAME,
    wml_client.repository.ModelMetaNames.TYPE: 'scikit-learn_1.0',
    wml_client.repository.ModelMetaNames.SOFTWARE_SPEC_UID: software_spec_uid
}
#no changes in the above section execute as it is
```

In [62]:
```python
#Save model
model_details = wml_client.repository.store_model(
    model=DEMO_MODEL,
    meta_props=model_props,
    training_data=X_train,
    training_target=y_train
)
```

In [63]:
```
model_details
```

Out[63]:
```
{'entity': {'hybrid_pipeline_software_specs': [],
  'label_column': 'WQI clf',
  'schemas': {'input': [{'fields': [{'name': 'f0', 'type': 'float'},
      {'name': 'f1', 'type': 'float'},
      {'name': 'f2', 'type': 'float'},
      {'name': 'f3', 'type': 'float'},
      {'name': 'f4', 'type': 'float'},
      {'name': 'f5', 'type': 'float'},
      {'name': 'f6', 'type': 'float'},
      {'name': 'f7', 'type': 'float'}],
     'id': '1',
     'type': 'struct'}],
   'output': []},
  'software_spec': {'id': '12b83a17-24d8-5082-900f-0ab31fbfd3cb',
   'name': 'runtime-22.1-py3.9'},
  'type': 'scikit-learn_1.0'},
 'metadata': {'created_at': '2022-11-16T06:13:42.920Z',
  'id': '8997759c-9a50-452e-bcb2-a2f697eb10f5',
  'modified_at': '2022-11-16T06:13:45.876Z',
  'name': 'quality_prediction',
  'owner': 'IBMid-6620043R89',
  'resource_key': '4edeb366-5662-47f4-8d79-3dfa1c29e08c',
  'space_id': 'c870c0b4-f040-4f8f-8778-410c757a8e21'},
 'system': {'warnings': []}}
```

In [64]:
```
model_id = wml_client.repository.get_model_id(model_details)
model_id
```

Out[64]:
```
'8997759c-9a50-452e-bcb2-a2f697eb10f5'
```

In [65]:
```
# Set meta
deployment_props = {
    wml_client.deployments.ConfigurationMetaNames.NAME:DEPLOYMENT_NAME,
    wml_client.deployments.ConfigurationMetaNames.ONLINE: {}
}
```

In [66]:
```
# Deploy
deployment = wml_client.deployments.create(
    artifact_uid=model_id,
    meta_props=deployment_props
)
```

```
#######################################################################################

Synchronous deployment creation for uid: '8997759c-9a50-452e-bcb2-a2f697eb10f5' started

#######################################################################################


initializing
Note: online_url is deprecated and will be removed in a future release. Use serving_urls instead.

ready

------------------------------------------------------------------------------------------------


------------------------------------------------------------------------------------------------
Successfully finished deployment creation, deployment_uid='0c97875a-5c13-4907-9920-77315392d368'
------------------------------------------------------------------------------------------------
```

In [67]:
```
X_train
```

Out[67]:
```
array([[ 3.89478618e-01,  1.32515603e+00,  1.32502300e+00, ...,
        -4.11114409e-01,  9.17913602e-04, -6.64761952e-02],
       [ 2.34686398e-01,  8.48406053e-01, -4.81377707e-02, ...,
        -5.21609938e-01, -2.40085223e-01, -1.51883820e-01],
       [-2.58253201e+00,  3.71656072e-01, -4.81377707e-02, ...,
         9.45525136e-01, -2.72541926e-01, -1.79259015e-01],
       ...,
       [ 1.16343972e+00, -3.20396878e+00, -1.24965344e+00, ...,
        -4.72500814e-01,  6.92197261e-01,  1.73411073e-01],
       [-1.15844358e+00, -7.40760550e-01, -5.63073058e-01, ...,
        -3.68143926e-01, -2.46019471e-01, -1.79830886e-01],
       [ 1.38014883e+00, -1.29696886e+00,  6.38442613e-01, ...,
        -6.87353231e-01, -2.73752997e-01, -1.79855750e-01]])
```

In [ ]:
```
'''Temp          float64
DO            float64
PH            float64
Conductivity  float64
BOD           float64
NI            float64
Fec_col       float64
Tot_col       float64'''
```

In [68]:
```
X_test[0]
```

Out[68]:
```
array([ 0.54427084,  0.21273941,  1.325023  , -0.25411375, -0.34534496,
       -0.5768577 , -0.09257677, -0.09181255])
```

GITHUB and project demo link:

GITHUB LINK:

https://github.com/IBM-EPBL/IBM-Project-51358-1660978492

PROJECT DEMO LINK:

https://drive.google.com/file/d/1MxagvwnoYv8qp9bHxZrixGR_wToDx_1T/view?usp=sharing