A Project Report
on

Water Quality Index Prediction of River Ganga using Machine Learning Algorithms: A comparative Study

Submitted for
Undergraduate B.Tech program (2023-24) in
Computer Science & Engineering

By

RITIKA NASKAR

Under the supervision of

Dr. Sudip Kumar Adhikari
Assistant Professor
Department of Computer Science & Engineering
Cooch Behar Government Engineering College

Department of Computer Science & Engineering
Cooch Behar Government Engineering College
Vill – Harinchawra, P.O. - Ghughumari
Cooch Behar – 736170
2024

ACKNOWLEDGEMENT

The work summarized in this report, explores Water Quality Index Prediction of River Ganga using Machine Learning Algorithms: A comparative Study in terms of other Water Parameters (Like pH, Temperature, DO, BOD etc.) which works in really efficient way (both in case of accuracy & time consumption).

This is a combined endeavor of a number of people who directly or indirectly helped me in completing the work. This documentation would never have been more educative and efficient without the constant help and guidance of my guide ***Dr. Sudip Kumar Adhikari*** (Assistant Professor of Department of Computer Science & Engineering, Cooch Behar Govt. Engineering College). I would like to thank him for giving me the right guidance and encouraging me to complete this work within time.

I would also like to express my heartiest gratitude to ***Dr. Sourav De*** (Head of the Department of Computer Science & Engineering, Cooch Behar Govt. Engineering College).

Last but not the least, I would like to express my gratitude to the Office staffs of Department of Computer Science & Engineering, Cooch Behar Govt. Engineering College for their constant cooperation without which we would not be able to finish my work.

_____

Ritika Naskar

CERTIFICATE OF ORIGINALITY

The project entitled “Water Quality Index Prediction of River Ganga using Machine Learning: A comparative study” has been carried out by ourselves in partial fulfillment of the degree of Bachelor of Technology in Computer Science & Engineering of Cooch Behar Government Engineering College, Cooch Behar, under Maulana Abul Kalam Azad University of Technology during the academic year 2023 – 2024.

While developing this project no unfair means or illegal copies of software etc. have been used and neither any part of this project nor any documentation have been submitted elsewhere or copied as far in my knowledge.

Name: Ritika Naskar

University Roll No.: 34900120026

University Registration No: 203490100110025

CERTIFICATE OF APPROVAL

This is to certify that the project entitled “Water Quality Index Prediction of River Ganga using Machine Learning: A comparative study” has been carried out by Ritika Naskar under my supervision in partial fulfillment for the degree of Bachelor of Technology (B. TECH) in Computer Science & Engineering of Cooch Behar Government Engineering College, Cooch Behar affiliated to Maulana Abul Kalam Azad University of Technology during the academic year 2023-2024.

It is understood that by this approval the undersigned do not necessarily endorse any of the statements made or opinion expressed therein but approves it only for the purpose for which it is submitted.

Dr. Sourav De
Associate Professor and HOD
Department of Computer Sc. & Engg.

Dr. Sudip Kumar Adhikari
Assistant Professor and Supervisor
Department of Computer Sc. & Engg.

CONTENT

INTRODUCTION

Ganga River is considered as the most prominent river of India. However, the water quality of the Ganga has been a cause for concern due to various anthropogenic activities, industrial discharges, and urbanization along its banks. The Water Quality Index (WQI) assesses the overall quality of water based on various parameters. For the Ganga River, key parameters include dissolved oxygen, biochemical oxygen demand, pH, total coliform bacteria, and others. Calculation involves assigning weights to each parameter, normalizing values, and aggregating them into a single index. The goal is to provide a concise representation of water quality, aiding in decision-making for environmental management and public health. Here we are going to create a machine learning model which will predict the quality of the Ganga River water where the measuring parameters dataset is collected from the website of Central Pollution Control Board. Based on this collected data from different cities where Ganga riverbed is huge, the model will be trained and the prediction algorithm which will be most accurate and precise will be used. This study uses the Water Quality Index (WQI) to predict the water quality classification

The amount of oxygen dissolved in the water can tell a lot about its quality. As the dissolved oxygen (DO) increases, the temperature of water decreases. It is usually measured in both milligrams per litre and percentage saturation.

The relative amount of free hydrogen and hydroxyl ions determines the pH of water. In other words, it is a measure of how acidic or basic is the given water sample. pH ranges from 0-14, with 7 being neutral.

BOD measures the amount of dissolved oxygen required by microorganisms to decompose organic matter in water. High BOD levels indicate pollution and can impact WQI significantly.

Conductivity, a measure of water's ability to conduct an electric current, is an important parameter in the calculation of the Water Quality Index (WQI).

Nitrate parameters are crucial for Water Quality Index (WQI) prediction as they indicate the level of nitrogen pollution in water, often from agricultural runoff and sewage.

The dependency of E. coli on WQI is rooted in its role as an indicator of microbial pollution, impacting the suitability of water for various uses, including drinking and recreational activities

Using these physicochemical parameters, a machine learning model can be made to effectively assess the water quality of Ganga River water.

LITERATURE SURVEY

Several studies have been conducted to assess the quality and health status of the various river basin at various stations throughout India. From 1990s to the current date many assessments are carried out on various rivers all over the world. Therefore, surveillance of water quality is mandatory. Though water quality can be tested using traditional techniques such as collecting the water specimens manually and then analyzed it in a laboratory (Wu and Liu, 2012). But it can be considered time-consuming and expensive. Sensors can also be regarded as another conventional approach. However, using sensors is considered costly to test all the water quality variables and often show low precision (Oelen et al., 2018). Another solution for monitoring water quality is predictive modelling using machine learning and deep learning approaches. Compared to other conventional methods, it has several advantages: lower costs, efficient in terms of time required for travel and collection, enables prediction under various phases of a system, and predicts desirable values when accessing a site is inconvenient (Sinshaw et al., 2019).

Several studies have compared these algorithms for WQI prediction: Kumar et al. (2020) found RF and GBM to outperform LR and SVM in accuracy. Sharma et al. (2021) demonstrated ANN's superiority in handling non-linear relationships but noted its higher computational cost. Gupta et al. (2022) highlighted RF's balance between accuracy and interpretability, making it a preferred choice. Verma et al. in 2019 published "Predicting Water Quality Index of River Ganga Using Support Vector Regression and Ensemble Learning Techniques" by using algorithm Support Vector Regression (SVR) and ensemble learning techniques. Singh et al at 2018 publishing "Evaluation of Machine Learning Models for Water Quality Prediction in Ganga River" by using Algorithms Decision Trees, Random Forest, and Neural Networks.

Mostly the Water Quality is judged on various parameters like pH, Temperature, Potassium, Sulphate, Hardness, BOD, COD, Conductivity, E. coli etc.

In 2010, a study was carried out to assess the Ganga water quality at India, using various physicochemical parameters such as turbidity, total solids, hardness, free CO2, nitrate, BOD, COD, and phosphate. The values of these parameters are compared with WHO and ISI standards.

A study was conducted in Kanpur on Ganga using Pearson's correlation coefficient value which is determined using correlation matrix to identify highly correlated and inter-related quality parameters.

It has been found that in West Bengal, present day – Kolkata, is the most polluted station of Ganga River basin which is mainly known as River Hooghly. There are several studies that are already done on and in progress on this river. In that reason, Ganga is the most polluted River Basin in West Bengal.

River Ganga is the most polluted river in India, so we set it the main river of India and according to the West Bengal Region it is the main river. Mostly in Kolkata, Hooghly, Howrah the River Ganga is so polluted due to Industrial affairs and very congested locality and their various usage of water in many purposes and also the Oil, Thermal Industries are very much responsible for this pollution.

AIM

The Ganges River and the Ganges River Basin cover one-quarter of India's land. The selection of the river Ganga for this project stems from its profound ecological, cultural, and socioeconomic significance. The Ganges, revered as a sacred river in India, holds immense cultural importance, and its waters are considered purifying in various religious practices. The main aim of this project is to create a Water Quality Index (WQI) prediction using machine learning algorithms which will analyze the quality of Ganga River water and decide whether the water is used for human consumption or not. By leveraging machine learning algorithms, we intend to apply different machine learning algorithms in our dataset and compare which ML algorithm will give more accurate water quality levels based on relevant environmental parameters.

OBJECTIVES

The primary objective of this project is to develop a predictive model for the Water Quality Index of the River Ganga.

- Data Collection and Preprocessing:
    - Gather comprehensive datasets encompassing various water quality parameters along the Ganges. Preprocess the data to handle missing values, outliers, and ensure compatibility with machine learning algorithms.
- Feature Selection:
    - Identify key features impacting water quality and select a subset for model training to enhance efficiency and interpretability.
- Algorithm Selection:
    - Evaluate and compare the performance of diverse machine learning algorithms such as Random Forest, Support Vector Machines (SVM), and others ML algorithms for predicting the Water Quality Index.
- Model Training and Validation:
    - Train the selected models on historical data, utilizing a portion for validation to assess performance and fine-tune hyperparameters.
- Prediction and Visualization:
    - Predict which ML algo gives the most accurate water quality level of the and receive real-time predictions of the Water Quality Index.

Fig: Working procedure of machine learning model

MATERIALS AND METHODS

- **Water Quality Index:**

  Water quality index (WQI) is one of the most used tools to describe water quality. It is based on physical, chemical, and biological factors that are combined into a single value that ranges from 0 to 100 and involves 4 processes:

  1. Selection of parameters for measurement of water quality.

  2. Development of a rating scale to obtain the rating (V_r).

  3. Estimating the unit weight of each indicator parameter (Wi) by considering the weightage of each parameter.

  4. Determining the subindex value ($W_i * V_r$).

  5. Aggregating the subindices to obtain the overall WQI.

The Water Quality Index uses six parameters, which are BOD, DO, Conductivity, pH, E.coli. To obtain the WQI value, all parameters needed to be converted first into subindices (SI), namely SI_{BOD}, SI_{DO}, SI_{EC}, SI_{pH} and $SI_{E.coli}$. Then, the calculation of WQI was performed by substituting all sub-indices of the six parameters into the WQI formula.

$$WQI = \sum_{i=1}^{n} SIi$$

Where,

WQI is the Water Quality Index.

n is the number of water quality parameters.

&

$SI_i = V_r \times W_i$

Where,

"SI_i" is the subindex value, V_r is the rating which can be obtained from a rating scale.

Parameters	Range				
pH	7-8.5	8.5-8.6 6.8-6.9	8.6-8.8 6.7-6.8	8.8-9.0 6.5-6.7	>9.0 <6.5
DO	>6	5.1-6	4.1-5	3.0-4	<3
BOD	0-3	3.0-6	6.0-80	80.0-125	>125
Conductivity	0-75	75-150	150-225	225-300	>300
Nitrate	0-20	20.0-50	50.0-100	100-200	>200
Total Coliform	0-5	5.0-50	50-500	500-10000	>10000
Vr	100	80	60	40	0

Table1: Above we've shown the rating table based on all the parameters

"W_i" is the unit weight.

Parameters	Unit Weights (W_i)
pH	0.165
DO	0.281
BOD	0.234
Conductivity	0.009
Nitrate	0.028
Total Coliform	0.281

Table2: Above we've shown the unit weight table of all parameters

Now, let's calculate the sub-indices for each parameter and the overall Water Quality Index (WQI):

1. **pH Sub-index:**

   If pH = 7.3(from the dataset), pH lies on the range of 7 – 8.5(from the rating scale). So V_r value of pH is 100 and W_{pH}=0.165

   SI_{pH} = 100*0.165 = 16.5

2. **DO Sub-Index:**

   If DO= 9.1(from the dataset), DO lies on range where DO is greater than 6(from the rating scale). So V_r value of DO is 100 and W_{DO}=0.281

   SI_{DO} = 100*0.281 = 28.1

3. **BOD Sub-index:**

   If BOD = 3.7(from the dataset), BOD lies on the range of 3.0 – 6(from the rating scale). So V_r value of BOD is 80 and W_{BOD}=0.234

   SI_{pH} = 80*0.234= 18.72

4. **Conductivity Sub-index:**

   If conductivity = 335(from the dataset), conductivity lies on the range where conductivity is greater than 300(from the rating scale). So V_r value of conductivity is 0 and $W_{conductivity}$ = 0.009

   $SI_{conductivity}$ = 0*0.009 = 0

5. **Nitrate Sub-Index:**

   If Nitrate= 0.815(from the dataset), Nitrate lies on the range of 0 - 20(from the rating scale). So V_r value of Nitrate is 100 and $W_{Nitrate}$ =0.028

   $SI_{Nitrate}$ = 100*0.028= 2.8

6. Total Coliform Sub-Index:

If Total Coliform = 1660(from the dataset), Total Coliform lies on the range of 500 – 10,000(from the rating scale). So V_r value of Total Coliform is 40 and $W_{\text{Total Coliform}}$ =0.281

$SI_{\text{Total Coliform}}$ = 40*0.281= 11.24

Now, sum up the individual sub-indices:

$WQI = SI_{PH} + SI_{DO} + SI_{BOD} + SI_{conductivity} + SI_{Nitrate} + SI_{\text{Total Coliform}}$

WQI = 16.5 + 28.1 + 18.72 + 0 + 2.8 + 11.24 = 77.36

- **Water Quality Classification:**

  Water quality classifications based on the Water Quality Index (WQI) can vary depending on the standards or guidelines set by different regulatory bodies. However, a common approach is to classify water into different categories based on the WQI value. Here is a general classification scheme:

  1. Poor (WQI < 25): Water quality is very poor, and it is not suitable for most uses without extensive treatment.
  2. Marginal (25 ≤ WQI < 50): Water quality is marginal, and it may not be suitable for certain uses without treatment
  3. Medium (50 ≤ WQI < 70): Water may have some significant impairments, and precautions may be needed.
  4. Good (70 ≤ WQI < 90): Water is generally safe for designated uses, with minor impairments.
  5. Excellent (90 ≤ WQI < 100): Water is of quality excellent, suitable for all designated uses.

- **Missing Values:**

  There are frequently many missing values in real-world data. It is common and may have significant impact on the decisions that can be made from the data. The cause for missing values may be data corruption or failure to save data. In general, missing values per variable, which range between 0.4% and 10%, are considered normal. Processing missing data is very important during the pre-processing of the dataset because many Machine Learning algorithms do not support missing values. A value must be present for every row and column of a data set for most ML algorithms to work properly. Therefore, it is common to identify missing values within a dataset and to replace them by the mean value of the parameter. This is referred to as data imputation. Data imputation is the process of replacing missing values with substitution values obtained from a statistical analysis to produce a complete dataset.

- **Imbalance Data Issue:**

  One of the main challenges of Machine Learning is the processing of imbalance data for the classification. An imbalanced dataset is a situation in which the occurrence of one outcome from two possible outcomes is very rare. The data are unevenly distributed in classes and certain classes have large samples (majority classes), while some have a few samples (minority classes). In this kind of dataset, not even a single sample of the minority class is classified correctly, and accuracy can reach up to 99%. It means that, when imbalanced data occur, classifiers have a tendency to make a biased model that has a poorer predictive accuracy over the minority class. Moreover, the gap between sensitivity and specificity may become large, especially in traditional classifiers. Therefore, the classification of imbalanced data becomes a highly explored issue because it creates a bias in the performance of traditional classifiers. They consider the error rate, but not the distribution of data, and the minority class samples are removed from the overall classification result. The modification of the existing classifier to accommodate imbalanced data, such as using ensemble methods, has been proven to be successful.

ALGORITHMS TO BE APPLIED

1. **Support Vector Machine (SVM):**

Support Vector Machine is a supervised machine learning algorithm used for classification and regression tasks. The primary goal of SVM is to find a hyperplane in an n-dimensional space (where n is the number of features) that distinctly classifies the data points into different classes.

2. **Random Forest (RF):**

The Random Forest algorithm is an ensemble learning method that combines multiple decision trees to create a more robust and accurate predictive model. Each tree in the forest is constructed using a random subset of the training data and a random subset of features at each split, reducing the risk of overfitting.

3. **KNN:**

This K-Nearest Neighbours method classifies samples by discovering the closest neigh boing given points and assigns the class of the majority of n neighbours. If there is a draw, different techniques might be used to resolve it. However, KNN is not suggested for a large dataset since all processing occurs during the testing, and it iterates through all training datasets and calculates the nearest neighbor each time.

4. **Decision Tree:**

The decision tree (DT) is an explicit, simple algorithm that makes decisions founded on the values from all pertinent input parameters. DT uses entropy for selecting the root variable and, depending on it, reviews the values of the other parameters. DT obtained all decisions of the parameters arranged in a top-down tree and plans the decision according to different values from different parameters. Decision tree models are frequently found in previous studies to perform well on imbalanced data. However, decision-tree based ensemble models, including Random Forest (RF) and Gradient Boosting (GB), almost always outperform the single decision tree. The advantages of decision-tree-based model are the fact that they are not sensitive to missing values, are able to manage both regular attributes and data, and are highly efficient. Compared to other ML models, decision tree-based models are more favorable for short-term predictions and may have a quicker calculation speed.

5. **Multivariate Regression:**

One of the most common types of predictive analysis is multiple linear regression. This type of analysis allows us to understand the relationship between a continuous dependent variable and two or more independent variables. The independent variables can be either continuous (like age and height) or categorical (like gender and occupation). It is important to note that if dependent variable is categorical, one should dummy code it before running the analysis. When making a prediction or forecasting, it is best to have as much data as possible. Multiple linear regression is a model that allows you to account for all of these potentially significant variables in one model. The benefits of this approach include a more accurate and detailed view of the relationship between each factor and the outcome. It means you can plan and monitor your data more effectively.

IMPLEMENTATION

- ***Packages and Libraries:***

PYTHON: We use PYTHON programming language throughout the project. It is very simple, robust object oriented and user friendly. Very complex algorithms can be implemented very easily in python. There are many user-friendly libraries available in python for beginner and most of them are free to use. So, we choose python as our main programming language for this project.

PANDAS: For Data frames and Data Extraction, Data Cleaning and Exploratory Analysis and Normalization we used Pandas Module and its Built-in functions written in Python.

NUMPY: This is very useful for numerical calculation. It provides Array class which is 50 times faster than traditional python list. It also provides many methods which helps in matrix. We can create array with as many dimensions we want with it. So, this is very helpful for us.

SEABORN: Seaborn is very useful for plotting data as graph developed on Top of MATPLOTLIB. It support many complex graphs which is very tricky to implement using Matplotlib alone. We mainly use it for image visualization and data visualization.

SKLEARN: Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering, and dimensionality reduction via a consistence

interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

Modules of SKLEARN Library*:*

1. sklearn.model_selection: It is used to split our data into train and test sets where feature variables are given as input in the method.

2. sklearn.svm: It is commonly employed in classification tasks because they are particularly efficient in high-dimensional fields.

3. sklearn.ensemble: The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm to improve generalizability / robustness over a single estimator.

4. sklearn.datasets: Generated sklearn datasets are synthetic datasets, generated using the sklearn library in Python.

5. sklearn.neighbors: It provides functionality for unsupervised and supervised neighbors-based learning methods.

6. sklearn.metrics: The sklearn.metrics module implements several loss, score, and utility functions to measure classification performance. Some metrics might require probability estimates of the positive class, confidence values, or binary decisions values.

7. sklearn. Preprocessing: The sklearn. preprocessing package provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators.

8. sklearn.linear_model: It is a class of the sklearn module if contain different functions for performing machine learning with linear models.

9. sklearn.tree: It is a non-parametric supervised learning method used for classification and regression.

MATPLOTLIB. PYPLOT: It is a collection of command style functions that make matplotlib work like MATLAB. It creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

- ***Data Collection on Ganga River****:*

  Here we discuss which dataset we use in our project. We collect the Dataset of Ganga River from Central Pollution Control Board Website and they provide values of various water variables (like DO, pH, BOD, Conductivity, Nitrate, Total Coliform etc.) of 16 different stations from different states from 2013 to 2020 in yearly basis.

  [*Website Link: https://cpcb.nic.in/*]

Station	Year												
		D.O. (mg/L)		pH		Conductivity (µSiemens/cm)		BOD (mg/L)		Total coliform		Nitrate	
		Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
Garhmukteshwar, UP	2013	7	11.2	6.8	7.8	190	480	2.2	5.2	920	2400	0.4	1.23
	2014	7.2	11.1	6.8	7.8	170	300	1.4	3.5	900	1500	0	0.68
	2015	6.5	9.5	7.2	8.4	210	290	2.1	3.6	1100	1500	0	0.5
	2016	7.1	9.65	7	7.6	129	246	1.6	3.5	900	1200	0.2	0.5
	2017	7.3	8.7	7.4	7.7	120	258	1.2	3	410	610	0.3	0.6
	2018	7.1	9.1	6.8	7.8	180	257	2.1	3.2	410	630	0.5	0.62
	2019	7.5	10.5	6.8	7.7	155	266	0.9	2.8	540	3500	0.33	0.82
	2020	7.1	10.8	7	8.5	150	275	BDL	9.8	280	9400	BDL	0.5
Bithoor, UP	2013	6.4	10.6	7.4	8.7	19	195	2.3	3.8	2300	4300	0.14	2.2
	2014	6.4	9.8	7.7	8.7	42	428	2.2	4.4	3500	4900	0.18	0.28
	2015	6.6	9.8	7.4	8.8	215	372	2.1	3.2	2400	5800	0	0.3
	2016	7	9.9	6.9	8.4	202	398	2.4	3.6	2700	5400	0.2	0.4
	2017	6.1	11.3	7.3	8.8	213	397	2.6	3.6	3800	6300	0.3	0.3
	2018	6.7	11.6	7.8	8.7	202	413	2.8	3.8	3500	5200	_	_
	2019	6.1	10.6	7.9	8.8	203	350	2.3	4.4	3300	5800	__	__
	2020	6.2	12	7.6	8.6	186	370	2.4	3.2	3300	9400	-	-
Allahabad(Rasoolabad), UP	2013	6.9	10.5	8	8.3	338	385	2.5	4.8	10000	33000	2.44	2.56
	2014	8.1	9.8	8.1	8.4	338	454	2.6	4.4	22000	47000	1.08	1.96
	2015	7.2	8.9	7.4	8.4	298	520	3.3	4.6	26000	35000	0	1.3
	2016	6.8	8.6	7.9	8.4	328	385	3.6	4.5	32000	58000	1	1.4
	2017	7.7	8.8	7.5	8.3	196	364	3.6	5.6	21000	43000	1.2	1.7
	2018	7.5	10.3	7.7	8.5	189	293	2.6	4.8	7900	32000	1.37	1.75
	2019	7.3	12.2	7.6	8.4	154	261	2.1	3.2	9400	24000	1.28	1.61
	2020	7.6	12.5	7.3	8.5	214	334	2	3.2	2100	4900	BDL	1.47
Varanasi (Assighat), UP	2013	6.9	8.8	7.6	8.4	268	370	2.9	3.2	7000	13000	0	0
	2014	7.3	9.2	8	8.4	282	392	2.4	3.3	3100	4900	0	0
	2015	7.5	8.7	7.9	8.8	245	776	2.5	8.6	2700	5000	0	0
	2016	7.4	9.8	7.4	8.5	320	496	2.8	3.5	2200	3600	0	0
	2017	7.5	8.8	7.8	8.6	340	535	2.8	3.3	2400	22000	0.3	0.3
	2018	7.4	9.2	8.2	8.4	345	488	2.5	3.2	1100	3400	0.1	0.32
	2019	7.2	10	8.1	8.4	364	512	1.7	3.3	1100	3400	0.12	0.24
	2020	6.5	10.4	7.9	8.6	242	512	2.1	6.5	1100	3400	BDL	0.6

Table3: Dataset of Ganga River from Central Pollution Control Board website

- ***Editor: Google Colaboratory (Colab)***

Google Colaboratory, or Colab, is an as-a-service version of Jupyter Notebook that enables you to write and execute Python code through your browser. Jupyter Notebook is a free, open-source creation from the Jupyter Project. It provides free access to computing resources, including GPUs and TPUs. Colab is especially well suited to machine learning, data science, and education.

- ***Experimental Result and Discussion:***

Firstly, we collected the Ganga River dataset from Central Pollution Control Board. Then we structurally took the dataset on a Separate Excel Sheet and made a Final Datasheet. Then we import that Final Dataset on Colabortory for further experiments. We have given some of the results we got in those procedures with proper Visualization and Reasons mentioned below.

STEP I - Final Data frame after Extract & Cleaning:

DO	pH	Conductivity	BOD (mg/L)	Total coliform	Nitrate
9.1	7.3	335	3.7	1660	0.815
9.15	7.3	235	2.45	1200	0.34
8	7.8	250	2.85	1300	0.25
8.375	7.3	187.5	2.55	1050	0.35
8	7.55	189	2.1	510	0.45
8.1	7.3	218.5	2.65	520	0.56
9	7.25	210.5	1.85	2020	0.575
8.95	7.75	212.5	9.8	4840	0.5
8.5	8.05	107	3.05	3300	1.17
8.1	8.2	235	3.3	4200	0.23
8.2	8.1	293.5	2.65	4100	0.15
8.45	7.65	300	3	4050	0.3
8.7	8.05	305	3.1	5050	0.3
9.15	8.25	307.5	3.3	4350	
8.35	8.35	276.5	3.35	4550	
9.1	8.1	278	2.8	6350	
8.45	8.1	70	3.3	23400	1.23
7.8	8.25	369.5	3.45	4800	0.24
8.2	8.1	320	3.1	4550	0.2
8	7.7	338	3.7	4750	0.35
8.55	7.95	315.5	3.4	6500	0.3
8.8	8.15	325	3.65	4900	
8.05	8.25	285.5	3.25	4850	
9.15	8.01	284	4.6	10450	
8.7	8.15	361.5	3.65	21500	2.5
8.95	8.25	396	3.5	34500	1.52
8.05	7.9	409	3.95	30500	0.65

Table 4: Extracted Dataset of different parameters of river water

STEP II – Dataset upload on Colab & print the data frame:

First, we've to convert the excel sheet to .csv file and then upload it on Google Drive. After uploading the file, we open the Colab Library and print the dataset.

```
[1] from google.colab import drive
    drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[2] import pandas as pd
    import numpy as np
    from sklearn.model_selection import train_test_split,GridSearchCV
    from sklearn.feature_selection import RFE
    from sklearn.svm import SVC,SVR
    from sklearn.ensemble import RandomForestClassifier,RandomForestRegressor
    from sklearn.neighbors import KNeighborsClassifier,KNeighborsRegressor
    from sklearn.metrics import ConfusionMatrixDisplay, r2_score, mean_absolute_error, mean_squared_error, accuracy_score, precision_score, recall_score, f1_score,confusion_matrix
    from sklearn.preprocessing import StandardScaler, LabelEncoder,PolynomialFeatures
    import matplotlib.pyplot as plt
    import seaborn as sns
    from sklearn.linear_model import LinearRegression,Ridge
    from sklearn.tree import DecisionTreeRegressor,DecisionTreeClassifier
```

```
[3] df = pd.read_csv('/content/drive/MyDrive/input.csv')
    df
```

	DO	Conductivity	BOD (mg/L)	Total coliform	Nitrate	PH
0	9.100	335.0	3.700	1660	0.815	7.30
1	9.150	235.0	2.450	1200	0.340	7.30
2	8.000	250.0	2.850	1300	0.250	7.80
3	8.375	187.5	2.550	1050	0.350	7.30
4	8.000	189.0	2.100	510	0.450	7.55

Fig1: Data mount on Google Colab

STEP III - replace the missing values by the mean value of the parameter:

First, we checked that how many missing values are there in the data frame. Then we remove the missing values with mean values of the parameter. After replacing the missing fields with mean value there will zero missing values

```
+ Code  + Text

[4] df.isnull().sum()

DO                0
Conductivity      0
BOD (mg/L)        0
Total coliform    0
Nitrate           9
PH                0
dtype: int64

[5] df['Nitrate '] = df['Nitrate '].fillna(df['Nitrate '].mean())
```

Fig2: Checking missing values

```
[7] df.isnull().sum()

DO                0
Conductivity      0
BOD (mg/L)        0
Total coliform    0
Nitrate           0
PH                0
dtype: int64
```

Fig3: Replacing missing values

STEP IV - print WQI Values for each row:

Here, we take relative weights and standard values recommended by WHO. Then we get all WQI values for each row according to the WQI formula we'd described above.

```
[9] df
```

	DO	Conductivity	BOD (mg/L)	Total coliform	Nitrate	PH	WQI
0	9.100	335.0	3.700	1660	0.815	7.30	77.36
1	9.150	235.0	2.450	1200	0.340	7.30	82.40
2	8.000	250.0	2.850	1300	0.250	7.80	82.40
3	8.375	187.5	2.550	1050	0.350	7.30	82.58
4	8.000	189.0	2.100	510	0.450	7.55	82.58
...	...	...	...	...	...	...	...
122	4.800	500.0	3.750	115500	0.350	7.75	54.88
123	5.700	467.5	2.850	90500	0.500	7.55	65.18
124	6.150	416.5	2.950	116500	0.415	7.15	70.80
125	5.750	563.5	3.400	115500	3.185	7.60	60.50

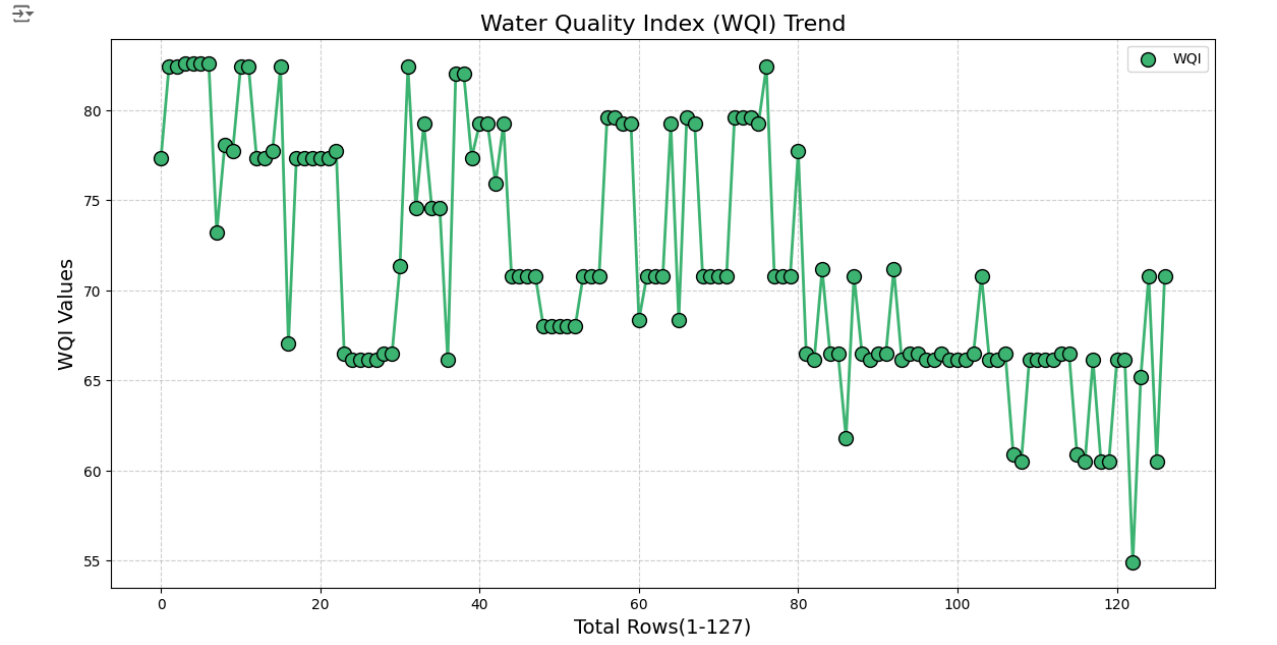


Fig4: Printing WQI values

STEP V – print the function to classify water quality based on WQI

Now, we can classify the quality (fair, good, marginal, poor) of the water based on the above WQI values. We are attaching the categorical distributions below.

```
[12] df
```

	DO	Conductivity	BOD (mg/L)	Total coliform	Nitrate	PH	WQI	WaterQuality
0	9.100	335.0	3.700	1660	0.815	7.30	77.36	Good
1	9.150	235.0	2.450	1200	0.340	7.30	82.40	Good
2	8.000	250.0	2.850	1300	0.250	7.80	82.40	Good
3	8.375	187.5	2.550	1050	0.350	7.30	82.58	Good
4	8.000	189.0	2.100	510	0.450	7.55	82.58	Good
...	...	...	...	...	...	...	...	...
122	4.800	500.0	3.750	115500	0.350	7.75	54.88	Medium
123	5.700	467.5	2.850	90500	0.500	7.55	65.18	Medium

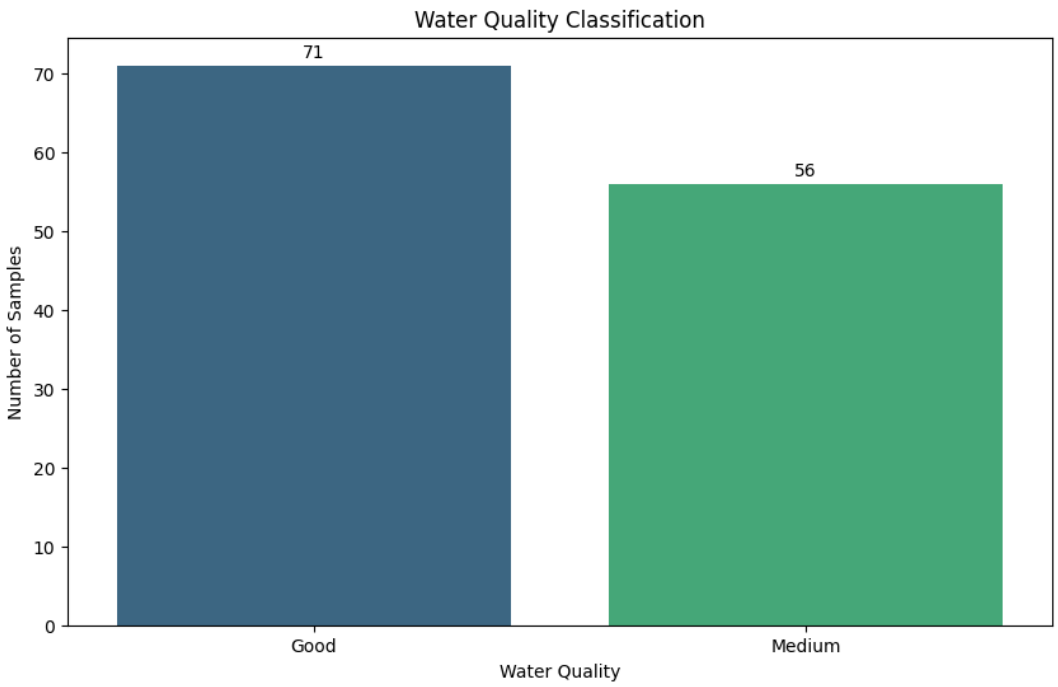
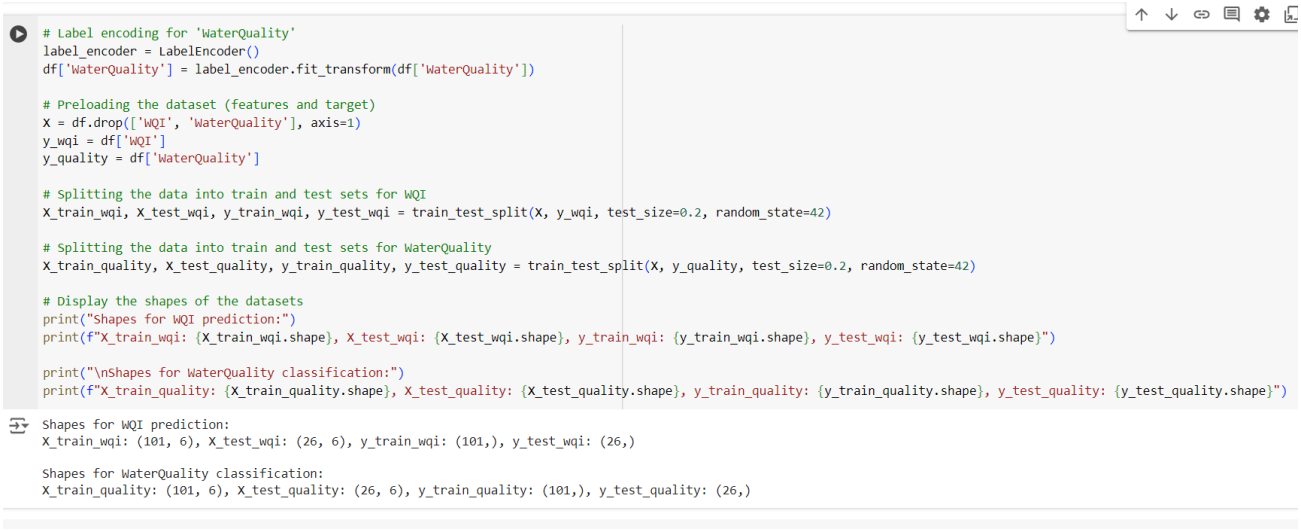


Fig5: Classifying water quality

STEP VI – Splitting the dataset into feature & target variables

Here, we split the dataset into two datasets- i) Training Dataset, ii) Testing Dataset. We take 80% of the actual dataset for training and 20% is used for testing. Here we Label encoding technique to convert categorical variables into numerical format



```
# Label encoding for 'WaterQuality'
label_encoder = LabelEncoder()
df['WaterQuality'] = label_encoder.fit_transform(df['WaterQuality'])

# Preloading the dataset (features and target)
X = df.drop(['WQI', 'WaterQuality'], axis=1)
y_wqi = df['WQI']
y_quality = df['WaterQuality']

# Splitting the data into train and test sets for WQI
X_train_wqi, X_test_wqi, y_train_wqi, y_test_wqi = train_test_split(X, y_wqi, test_size=0.2, random_state=42)

# Splitting the data into train and test sets for WaterQuality
X_train_quality, X_test_quality, y_train_quality, y_test_quality = train_test_split(X, y_quality, test_size=0.2, random_state=42)

# Display the shapes of the datasets
print("Shapes for WQI prediction:")
print(f"X_train_wqi: {X_train_wqi.shape}, X_test_wqi: {X_test_wqi.shape}, y_train_wqi: {y_train_wqi.shape}, y_test_wqi: {y_test_wqi.shape}")

print("\nShapes for WaterQuality classification:")
print(f"X_train_quality: {X_train_quality.shape}, X_test_quality: {X_test_quality.shape}, y_train_quality: {y_train_quality.shape}, y_test_quality: {y_test_quality.shape}")
```

```
Shapes for WQI prediction:
X_train_wqi: (101, 6), X_test_wqi: (26, 6), y_train_wqi: (101,), y_test_wqi: (26,)

Shapes for WaterQuality classification:
X_train_quality: (101, 6), X_test_quality: (26, 6), y_train_quality: (101,), y_test_quality: (26,)
```

Fig6: Splitting dataset into training & testing sets

STEP VII – Applying machine learning algorithms

Now we are applying mentioned ML Algorithms (KNN, SVM, Multi Variate Regression, Decision Tree & Random Forest) one by one and finding model's performance criteria such as

1. **R2 Squared:** It is a regression error metric that justifies the performance of the model. It represents the value of how much the independent variables can describe the value for the response/target variable.

   $R^2 = 1 – (SSE/SST)$

   - SSE is the sum of the squared differences between the actual dependent variable values and the predicted values from the regression model.
   - SST is the total variation in the dependent variable and is calculated by summing the squared differences between each actual dependent variable value and the mean of all dependent variable values.

2. **MSE:** Mean Squared Error (MSE) is defined as Mean or Average of the square of the difference between actual and estimated values.

   $MSE = (1/n) * \Sigma (actual – forecast)^2$

where:

- Σ – a symbol that means “sum”
- n – sample size
- actual – the actual data value
- forecast – the predicted data value

3. **MAE:** It provides a measure of the average magnitude of errors without considering their direction. MAE is less sensitive to outliers compared to RMSE.

   MAE = (1/n) Σ (i=1 to n) |y_i – ŷ_i|

   where:

   - n is the number of observations in the dataset.
   - y_i is the true value.
   - ŷ_i is the predicted value

4. **Accuracy:** Accuracy is the most common measure used for classifier assessment. It assesses the overall efficiency of the algorithm by estimating the likelihood of the actual value of the class label.

   Accuracy=(TP+TN)/(TP+TN+FP+FN)

5. **Precision:** Precision refers to the number of true positives divided by the total number of positive predictions (i.e., the number of true positives plus the number of false positives).

   Precision = TP/(TP+FP)

6. **Recall:** Recall is a metric that measures how often a machine learning model correctly identifies positive instances (true positives) from all the actual positive samples in the dataset.

   Recall = TP / (TP) + (FN)

7. **F1 Score**: The F1 score or F-measure is described as the harmonic mean of the precision and recall of a classification model.

2 * (Precision*Recall) / (Precision+Recall)

Definition of the Terms:

- True Positive (TP): Observation is positive, and is predicted to be positive.
- False Negative (FN): Observation is positive, but is predicted negative.
- True Negative (TN): Observation is negative, and is predicted to be negative.
- False Positive (FP): Observation is negative, but is predicted positive.

Based on the model's performance we obtain three diagrams such as

1. Actual vs. Predicted WQI: Actual and predicted values plot is a visualization technique used to compare the actual and predicted values by the model. This helps in evaluating the performance of the model and seeing how close the predicted results are to the actual values.
2. Confusion Matrix: A confusion matrix is a performance evaluation tool in machine learning, representing the accuracy of a classification model. It displays the number of true positives, true negatives, false positives, and false negatives.
3. Evaluation Metrics for Training & Testing: It refers to the set of metrics used to measure the effectiveness of a machine-learning model. These metrics help to determine how well a model can make accurate predictions or classifications on unseen data.

STEP VII – Applying machine learning algorithms

Here we obtained model performance criteria of all the ML Algorithms based on the specific parameters like classification (Accuracy, Precision, Recall, F1 Score) and regression (MAE, MSE, R2 Squared) metrices.

	Model	R2	MAE	MSE
0	KNN Regressor	82.7722	2.0646	6.5137
1	SVM Regressor	45.4213	3.3562	20.6360
2	Multivariate Linear Regression	45.1635	3.8735	20.7335
3	Random Forest Regressor	94.5030	0.7514	2.0784
4	Decision Tree Regressor	90.9593	0.7115	3.4182

	Model	Accuracy	Precision	Recall	F1 Score
0	KNN Classifier	0.8462	0.8846	0.8462	0.8443
1	SVM Classifier	0.8077	0.8220	0.8077	0.8033
2	Random Forest Classifier	0.9615	0.9641	0.9615	0.9614
3	Decision Tree Classifier	0.9615	0.9641	0.9615	0.9614

Fig7: Comparison chart of all the algorithms

ANALYSIS

1. Random Forest Classifier has the highest scores in all metrics (Accuracy, Precision, Recall, and F1 Score), indicating it perfectly predicts the water quality classifications in the testing set.

2. Decision Tree Classifier also performs very well, with high scores in all metrics, but not as perfect as the Random Forest.

3. KNN Classifier and Multivariate Regression have similar performance, with decent accuracy and other metric scores.

4. SVM Classifier has the lowest scores among the five models, indicating it is less suitable for this classification task compared to the other models.

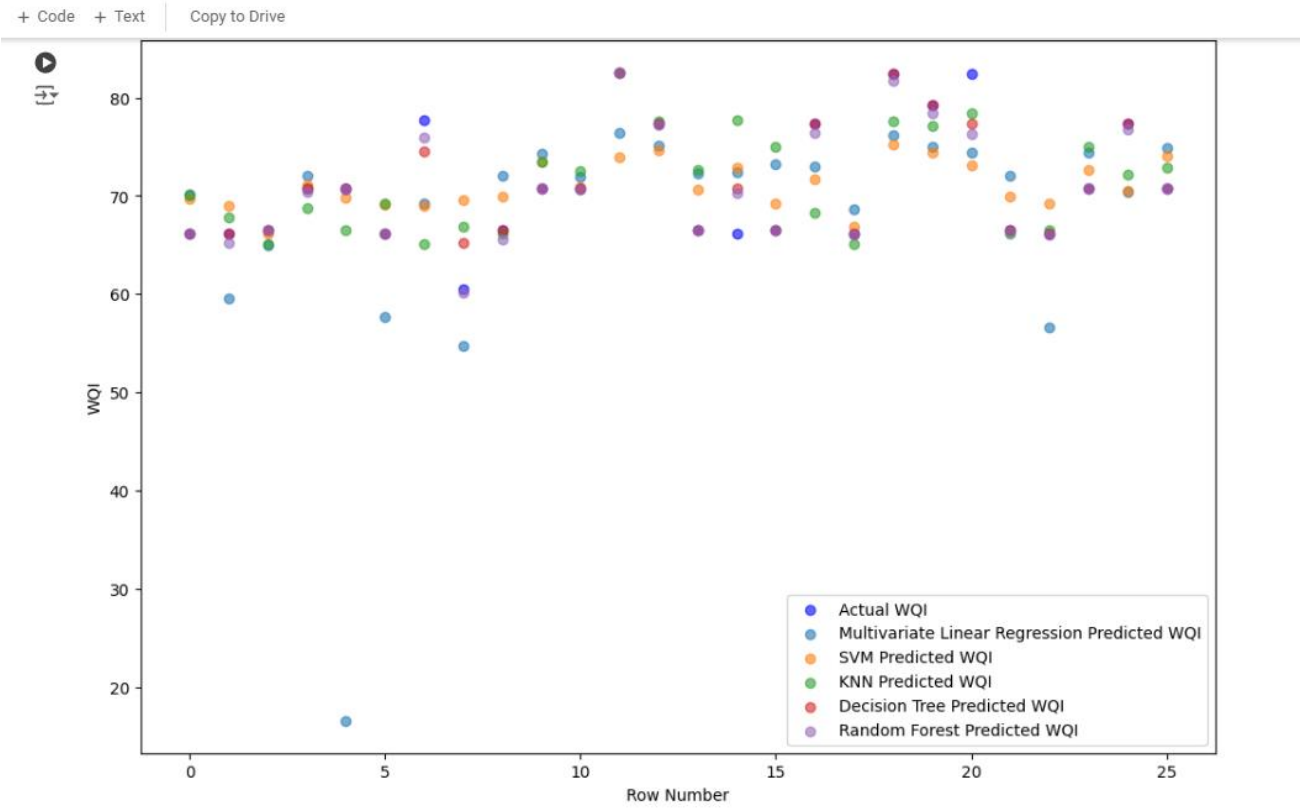


Fig8: Row no. Vs. actual & predicted WQI

CONCLUSION

In our complete analysis of Water Quality prediction using machine learning model, we have come across at some points. We have applied K Nearest Neighbor, Support Vector Machine, Multi Variate Linear Regression, Random Forest, Decision Tree algorithms for training & Testing and got the predicted WQI. Furthermore, we have obtained through the error calculation between the predicted and actual value of Water Quality Index, we predicted which algorithm best suits our collected dataset.

Based on the testing metrics, the Random Forest Classifier is the best model for water quality classification in your dataset, as it achieves perfect scores in all evaluation metrics (Accuracy, Precision, Recall, and F1 Score).

REFERENCES

I. Kosha A. Shah1 Geeta S. Joshi "Evaluation of water quality index for River Sabarmati", Gujarat, India

II. Shivani Chauhan "GANGA WATER QUALITY ASSESSMENT USING DEMPSTER SHAFER THEORY," INDIAN INSTITUTE OF INFORMATION TECHNOLOGY ALLAHABAD Prayagraj, UP 211015, India. July, 2020.

III. Armin Azad, Hojat Karami, Saeed Farzin, Sayed-Farhad Mousavi, Ozgur Kisi "Modeling River water quality parameters using modified adaptive neuro fuzzy inference system", Faculty of Civil Engineering, Semnan University, Semnan 35131-19111, Iran, Faculty of Natural Sciences and Engineering, Ilia State University, Tbilisi 0162, Georgia. Available online 11 April 2019.

IV. Gozen Elkirana, Vahid Nourania "Multi-step ahead modelling of river water quality parameters using ensemble artificial intelligence-based approach" Journal of Hydrology, a Faculty of Civil and Environmental Engineering, Near East University, Near East Boulevard, via Mersin 10, 99138 Nicosia, North Cyprus, Turkey.

V. K. Shukla, C. S. P. Ojha, and R. D. Garg "Surface water quality assessment of Ganga River Basin, India using Index mapping."

VI. Gagan Matta, Sachin Srivastava, R. R. Pandey, K. K. Saini "Assessment of physicochemical characteristics of Ganga canal water quality in Uttarakhand."

VII. K. Haritash, Shalini Gaur, Sakshi Garg "Assessment of water quality & suitability analysis of River Ganga in Rishikesh, India".

VIII. Anbarivan, "Indian-water-quality-analysis-and-prediction"

IX. Water Quality Index Formula by CCME (Canadian Council for Ministers and Environments) and YU_GN (For detailed calculation steps).

APPENDIX: SOURCE CODE

```
#dataset mount

from google.colab import drive

drive.mount('/content/drive')

#libraries

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split,GridSearchCV

from sklearn.feature_selection import RFE

from sklearn.svm import SVC,SVR

from sklearn.ensemble import RandomForestClassifier,RandomForestRegressor

from sklearn.neighbors import KNeighborsClassifier,KNeighborsRegressor

from sklearn.metrics import ConfusionMatrixDisplay, r2_score, mean_absolute_error, mean_squared_error,
accuracy_score, precision_score, recall_score, f1_score,confusion_matrix

from sklearn.preprocessing import StandardScaler, LabelEncoder,PolynomialFeatures

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.linear_model import LinearRegression,Ridge

from sklearn.tree import DecisionTreeRegressor,DecisionTreeClassifier

df = pd.read_csv('/content/drive/MyDrive/input.csv')

df

#missing value calculation

df.isnull().sum()

df['Nitrate '] = df['Nitrate '].fillna(df['Nitrate '].mean())

df

df.isnull().sum()

#wqi values calculation

# Define unit weights for each parameter

unit_weights = {


    'PH': 0.165,

    'DO': 0.281,
```

```
    'Conductivity': 0.009,
    'BOD (mg/L)': 0.234,
    'Total coliform': 0.281,
    'Nitrate ': 0.028

}

# Define Vr ranges for each parameter
vr_ranges = {
    'PH': {
        (float('-inf'), 6.5): 0,    # <6.5
        (6.5, 6.7): 40,             # 6.5–6.7
        (6.7, 6.8): 60,             # 6.7–6.8
        (6.8, 6.9): 80,             # 6.8–6.9
        (6.9, 7): 80,               # 6.9–7
        (7, 8.5): 100,              # 7–8.5
        (8.5, 8.6): 80,             # 8.5–8.6
        (8.6, 8.8): 60,             # 8.6–8.8
        (8.8, 9.0): 40,             # 8.8–9.0
        (9.0, float('inf')): 0      # >9.0
    },
    'DO': [(6, float('inf')),(5.1, 6),(4.1, 5),(3, 4),(0, 3)],
    'Conductivity': [(0, 75), (75, 150), (150, 225), (225, 300), (300, float('inf'))],
    'BOD (mg/L)': [(0, 3), (3, 6), (6, 80), (80, 125), (125, float('inf'))],
    'Total coliform': [(0, 5), (5, 50), (50, 500), (500, 10000), (10000, float('inf'))],
    'Nitrate ': [(0, 20), (20, 50), (50, 100), (100, 200), (200, float('inf'))]

}

vr_values = [100, 80, 60, 40, 0]

def get_vr_value(parameter, value):
```

```
    # Check if the parameter is 'pH'
    if parameter == 'PH':
        for (start, end), vr_value in vr_ranges['PH'].items():
            if start <= value < end:
                return vr_value
        return None

    # Check if the parameter is in the vr_ranges dictionary
    if parameter in vr_ranges:
        # Iterate over the Vr ranges for the parameter
        for i, (start, end) in enumerate(vr_ranges[parameter]):
            if start < value <= end:
                return vr_values[i]

    return None

def calculate_wqi(row):
    # Initialize total WQI
    total_wqi = 0

    # Iterate over each parameter in the row
    for parameter, value in row.items():
        # Get the Vr value for the parameter and value
        vr_value = get_vr_value(parameter, float(value))

        # If Vr value exists, calculate WQI for the parameter and add to total WQI
        if vr_value is not None:
            total_wqi += unit_weights[parameter] * vr_value

    return total_wqi

def main():
```

```
        reader = csv.DictReader(csvfile)
        for row in reader:
            # Calculate WQI for the current row
            wqi = calculate_wqi(row)

            # Add WQI of current row to total aggregate WQI
            #total_aggregate_wqi += wqi

            # Print WQI for current row
            print(f"WQI for row {reader.line_num}: {wqi}")

        # Print total aggregate WQI
        # print(f"\nTotal Aggregate WQI: {total_aggregate_wqi}")

df['WQI'] = df.apply(calculate_wqi, axis=1)
df
df.describe()
# Create the scatter plot with improved styling
plt.figure(figsize=(14, 7))

# Use a single color for both scatter points and line
color = 'mediumseagreen'

# Scatter plot with customized markers
plt.scatter(df.index, df['WQI'], color=color, edgecolor='black', s=100, zorder=5, label='WQI')
# Line plot with the same color
plt.plot(df.index, df['WQI'], color=color, linewidth=2, linestyle='-', zorder=3)

# Add titles and labels
plt.title('Water Quality Index (WQI) Trend', fontsize=16)
plt.xlabel('Total Rows(1-127)', fontsize=14)
plt.ylabel('WQI Values', fontsize=14)
```

```
# Add grid lines
plt.grid(True, linestyle='--', alpha=0.6)

# Add a legend
plt.legend()

# Show the plot
plt.show()
# Function to classify water quality based on WQI
def classify_water_quality(wqi):
    if wqi < 25:
        return 'Poor'
    elif 25 <= wqi < 50:
        return 'Marginal'
    elif 50 <= wqi < 70:
        return 'Medium'
    elif 70 <= wqi < 90:
        return 'Good'
    else:
        return 'Excellent'

# Apply water quality classification to each row in the dataset
df['WaterQuality'] = df['WQI'].apply(classify_water_quality)
df
# Apply water quality classification to each row in the dataset
df['WaterQuality'] = df['WQI'].apply(classify_water_quality)

# Filter for Medium and Good classifications
filtered_df = df[df['WaterQuality'].isin(['Medium', 'Good'])]

# Count the occurrences of each classification
```

```
quality_counts = filtered_df['WaterQuality'].value_counts()

# Plot the bar chart
plt.figure(figsize=(10, 6))
bar_plot = sns.barplot(x=quality_counts.index, y=quality_counts.values, palette='viridis')

# Annotate the bars with the frequency counts
for index, value in enumerate(quality_counts.values):
    bar_plot.text(index, value + 0.5, str(value), ha='center', va='bottom')

plt.title('Water Quality Classification')
plt.xlabel('Water Quality')
plt.ylabel('Number of Samples')
plt.show()
df.to_excel("df1.xlsx", "df1")
# Label encoding for 'WaterQuality'
label_encoder = LabelEncoder()
df['WaterQuality'] = label_encoder.fit_transform(df['WaterQuality'])

# Preloading the dataset (features and target)
X = df.drop(['WQI', 'WaterQuality'], axis=1)
y_wqi = df['WQI']
y_quality = df['WaterQuality']

# Splitting the data into train and test sets for WQI
X_train_wqi, X_test_wqi, y_train_wqi, y_test_wqi = train_test_split(X, y_wqi, test_size=0.2, random_state=42)

# Splitting the data into train and test sets for WaterQuality
X_train_quality, X_test_quality, y_train_quality, y_test_quality = train_test_split(X, y_quality, test_size=0.2,
random_state=42)

# Display the shapes of the datasets
```

```
print("Shapes for WQI prediction:")

print(f"X_train_wqi: {X_train_wqi.shape}, X_test_wqi: {X_test_wqi.shape}, y_train_wqi: {y_train_wqi.shape}, y_test_wqi: {y_test_wqi.shape}")


print("\nShapes for WaterQuality classification:")

print(f"X_train_quality: {X_train_quality.shape}, X_test_quality: {X_test_quality.shape}, y_train_quality: {y_train_quality.shape}, y_test_quality: {y_test_quality.shape}")

# KNN for WQI Prediction

knn_regressor = KNeighborsRegressor(7)

knn_regressor.fit(X_train_wqi, y_train_wqi)

# Predictions for training and testing sets

train_preds_wqi = knn_regressor.predict(X_train_wqi)

test_preds_wqi = knn_regressor.predict(X_test_wqi)

# Evaluation metrics for testing set

print("Testing Metrics:")

print(f"R2: {(r2_score(y_test_wqi, test_preds_wqi) * 100):.4f}")

print(f"MAE: {mean_absolute_error(y_test_wqi, test_preds_wqi):.4f}")

print(f"MSE: {mean_squared_error(y_test_wqi, test_preds_wqi):.4f}")

# KNN for WaterQuality Classification

knn_classifier = KNeighborsClassifier()

knn_classifier.fit(X_train_quality, y_train_quality)

# Predictions for training and testing sets

train_preds_quality = knn_classifier.predict(X_train_quality)

test_preds_quality = knn_classifier.predict(X_test_quality)

# Evaluation metrics for testing set

print("Testing Metrics:")

print(f"Accuracy: {accuracy_score(y_test_quality, test_preds_quality):.4f}")

print(f"Precision: {precision_score(y_test_quality, test_preds_quality, average='weighted'):.4f}")

print(f"Recall: {recall_score(y_test_quality, test_preds_quality, average='weighted'):.4f}")

print(f"F1 Score: {f1_score(y_test_quality, test_preds_quality, average='weighted'):.4f}")

# Actual vs. Predicted Plot for Regression

plt.figure(figsize=(10, 6))
```

```
    plt.scatter(y_train_wqi, train_preds_wqi, label='Training data', alpha=0.6)
    plt.scatter(y_test_wqi, test_preds_wqi, label='Testing data', alpha=0.6)
    plt.plot([min(y_train_wqi), max(y_train_wqi)], [min(y_train_wqi), max(y_train_wqi)], color='red')
    plt.xlabel('Actual WQI')
    plt.ylabel('Predicted WQI')
    plt.legend()
    plt.title('Actual vs. Predicted WQI')
    plt.show()
    # Confusion Matrix for Testing
    plt.figure(figsize=(10, 6))
    ConfusionMatrixDisplay.from_estimator(knn_classifier, X_test_quality, y_test_quality, cmap=plt.cm.Blues)
    plt.title('Confusion Matrix for KNN Classifier - Testing Data')
    plt.show()
    # Calculate metrics for training data
    train_preds_quality = knn_classifier.predict(X_train_quality)
    train_accuracy = accuracy_score(y_train_quality, train_preds_quality)
    train_precision = precision_score(y_train_quality, train_preds_quality, average='weighted')
    train_recall = recall_score(y_train_quality, train_preds_quality, average='weighted')
    train_f1 = f1_score(y_train_quality, train_preds_quality, average='weighted')
# Calculate metrics for testing data
    test_preds_quality = knn_classifier.predict(X_test_quality)
    test_accuracy = accuracy_score(y_test_quality, test_preds_quality)
    test_precision = precision_score(y_test_quality, test_preds_quality, average='weighted')
    test_recall = recall_score(y_test_quality, test_preds_quality, average='weighted')
    test_f1 = f1_score(y_test_quality, test_preds_quality, average='weighted')
    # Bar chart for metrics
    metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
    train_metrics = [train_accuracy, train_precision, train_recall, train_f1]
    test_metrics = [test_accuracy, test_precision, test_recall, test_f1]
    x = np.arange(len(metrics))  # the label locations
    width = 0.35  # the width of the bars
    fig, ax = plt.subplots(figsize=(10, 6))
```

```
    rects1 = ax.bar(x - width/2, train_metrics, width, label='Training')
    rects2 = ax.bar(x + width/2, test_metrics, width, label='Testing')
    # Add some text for labels, title and custom x-axis tick labels, etc.
    ax.set_xlabel('Metrics')
    ax.set_ylabel('Scores')
    ax.set_title('Evaluation Metrics for Training and Testing')
    ax.set_xticks(x)
    ax.set_xticklabels(metrics)
    ax.legend()
# Attach a text label above each bar in rects, displaying its height.
    def autolabel(rects):
        for rect in rects:
            height = rect.get_height()
            ax.annotate(f'{height:.4f}',
                        xy=(rect.get_x() + rect.get_width() / 2, height),
                        xytext=(0, 3),  # 3 points vertical offset
                        textcoords="offset points",
                        ha='center', va='bottom')

    autolabel(rects1)
    autolabel(rects2)

    fig.tight_layout()
    plt.show()
    # SVM for WQI Prediction
    svm_regressor = SVR()
    svm_regressor.fit(X_train_wqi, y_train_wqi)
    # Predictions for training and testing sets
    train_preds_wqi = svm_regressor.predict(X_train_wqi)
    test_preds_wqi = svm_regressor.predict(X_test_wqi)
    # Evaluation metrics for testing set
    print("Testing Metrics:")
```

```
print(f"R2: {(r2_score(y_test_wqi, test_preds_wqi)*100):.4f}")
print(f"MAE: {mean_absolute_error(y_test_wqi, test_preds_wqi):.4f}")
print(f"MSE: {mean_squared_error(y_test_wqi, test_preds_wqi):.4f}")
# SVM for WaterQuality Classification
svm_classifier = SVC()
svm_classifier.fit(X_train_quality, y_train_quality)
# Predictions for training and testing sets
train_preds_quality = svm_classifier.predict(X_train_quality)
test_preds_quality = svm_classifier.predict(X_test_quality)
# Evaluation metrics for testing set
print("Testing Metrics:")
print(f"Accuracy: {accuracy_score(y_test_quality, test_preds_quality):.4f}")
print(f"Precision: {precision_score(y_test_quality, test_preds_quality, average='weighted'):.4f}")
print(f"Recall: {recall_score(y_test_quality, test_preds_quality, average='weighted'):.4f}")
print(f"F1 Score: {f1_score(y_test_quality, test_preds_quality, average='weighted'):.4f}")
# Actual vs. Predicted Plot for Regression
plt.figure(figsize=(10, 6))
plt.scatter(y_train_wqi, train_preds_wqi, label='Training data', alpha=0.6)
plt.scatter(y_test_wqi, test_preds_wqi, label='Testing data', alpha=0.6)
plt.plot([min(y_train_wqi), max(y_train_wqi)], [min(y_train_wqi), max(y_train_wqi)], color='red')
plt.xlabel('Actual WQI')
plt.ylabel('Predicted WQI')
plt.legend()
plt.title('Actual vs. Predicted WQI')
plt.show()
# Confusion Matrix for Testing
plt.figure(figsize=(10, 6))
ConfusionMatrixDisplay.from_estimator(svm_classifier, X_test_quality, y_test_quality, cmap=plt.cm.Blues)
plt.title('Confusion Matrix for SVM Classifier - Testing Data')
plt.show()
# Calculate metrics for testing data
test_preds_quality = svm_classifier.predict(X_test_quality)
```

```
test_accuracy = accuracy_score(y_test_quality, test_preds_quality)
test_precision = precision_score(y_test_quality, test_preds_quality, average='weighted')
test_recall = recall_score(y_test_quality, test_preds_quality, average='weighted')
test_f1 = f1_score(y_test_quality, test_preds_quality, average='weighted')
# Bar chart for metrics
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
train_metrics = [train_accuracy, train_precision, train_recall, train_f1]
test_metrics = [test_accuracy, test_precision, test_recall, test_f1]
x = np.arange(len(metrics))  # the label locations
width = 0.35  # the width of the bars
fig, ax = plt.subplots(figsize=(10, 6))
rects1 = ax.bar(x - width/2, train_metrics, width, label='Training')
rects2 = ax.bar(x + width/2, test_metrics, width, label='Testing')
# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_xlabel('Metrics')
ax.set_ylabel('Scores')
ax.set_title('Evaluation Metrics for Training and Testing')
ax.set_xticks(x)
ax.set_xticklabels(metrics)
ax.legend()
# Attach a text label above each bar in rects, displaying its height.
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate(f'{height:.4f}',
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')
autolabel(rects1)
autolabel(rects2)
fig.tight_layout()
```

```
plt.show()

# Multivariate Linear Regression for WQI Prediction
linear_regressor = LinearRegression()
linear_regressor.fit(X_train_wqi, y_train_wqi)
# Predictions for training and testing sets
train_preds_wqi = linear_regressor.predict(X_train_wqi)
test_preds_wqi = linear_regressor.predict(X_test_wqi)
# Evaluation metrics for testing set
print("Testing Metrics:")
print(f"R2: {(r2_score(y_test_wqi, test_preds_wqi)*100):.4f}")
print(f"MAE: {mean_absolute_error(y_test_wqi, test_preds_wqi):.4f}")
print(f"MSE: {mean_squared_error(y_test_wqi, test_preds_wqi):.4f}")
# Standardize the features for Logistic Regression
scaler_quality = StandardScaler()
X_train_quality_scaled = scaler_quality.fit_transform(X_train_quality)
X_test_quality_scaled = scaler_quality.transform(X_test_quality)
# Logistic Regression for WaterQuality Classification
logistic_classifier = LogisticRegression(max_iter=1000)
logistic_classifier.fit(X_train_quality_scaled, y_train_quality)
# Predictions for training and testing sets
train_preds_quality = logistic_classifier.predict(X_train_quality_scaled)
test_preds_quality = logistic_classifier.predict(X_test_quality_scaled)
# Evaluation metrics for testing set
print("Testing Metrics:")
print(f"Accuracy: {accuracy_score(y_test_quality, test_preds_quality):.4f}")
print(f"Precision: {precision_score(y_test_quality, test_preds_quality, average='weighted'):.4f}")
print(f"Recall: {recall_score(y_test_quality, test_preds_quality, average='weighted'):.4f}")
print(f"F1 Score: {f1_score(y_test_quality, test_preds_quality, average='weighted'):.4f}")
# Actual vs. Predicted graph for Multivariate Linear Regression
plt.figure(figsize=(10, 5))
plt.scatter(y_test_wqi, test_preds_wqi, color='blue', label='Test Data')
```

```
plt.scatter(y_train_wqi, train_preds_wqi, color='orange', label='Training Data')

plt.plot([min(y_test_wqi), max(y_test_wqi)], [min(y_test_wqi), max(y_test_wqi)], color='red', linestyle='--', label='Ideal Fit')

plt.xlabel('Actual WQI')

plt.ylabel('Predicted WQI')

plt.title('Actual vs. Predicted WQI')

plt.legend()

plt.show()

# Confusion Matrix for Logistic Regression

train_cm = confusion_matrix(y_train_quality, train_preds_quality)

test_cm = confusion_matrix(y_test_quality, test_preds_quality)

# Plotting Confusion Matrix for Testing Set

plt.subplot(1, 2, 2)

sns.heatmap(test_cm, annot=True, fmt='d', cmap='Blues')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix - Testing Set')

plt.show()

# Bar chart for accuracy, precision, recall, and F1 score for Logistic Regression

metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']

train_scores = [accuracy_score(y_train_quality, train_preds_quality),

                precision_score(y_train_quality, train_preds_quality, average='weighted'),

                recall_score(y_train_quality, train_preds_quality, average='weighted'),

                f1_score(y_train_quality, train_preds_quality, average='weighted')]

test_scores = [accuracy_score(y_test_quality, test_preds_quality),

               precision_score(y_test_quality, test_preds_quality, average='weighted'),

               recall_score(y_test_quality, test_preds_quality, average='weighted'),

               f1_score(y_test_quality, test_preds_quality, average='weighted')]

x = np.arange(len(metrics))  # the label locations

width = 0.35  # the width of the bars

fig, ax = plt.subplots(figsize=(10, 5))

rects1 = ax.bar(x - width/2, train_scores, width, label='Train')
```

```
rects2 = ax.bar(x + width/2, test_scores, width, label='Test')
# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Scores')
ax.set_title('Evaluation Metrics for Training and Testing')
ax.set_xticks(x)
ax.set_xticklabels(metrics)
ax.legend()
# Attach a text label above each bar in rects, displaying its height.
def autolabel(rects):
    """Attach a text label above each bar in rects, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate(f'{height:.2f}',
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(rects1)
autolabel(rects2)

fig.tight_layout()
plt.show()
```

 # Random Forest for WQI Prediction

```
#random Forest
# Define a pipeline for Random Forest Regressor
rf_regressor = RandomForestRegressor(random_state=42)
param_grid_rf = {
    'max_depth': [3, 5],
    'min_samples_split': [5, 10],
    'min_samples_leaf': [4, 6],
```

```
    'n_estimators': [50, 75]
}
grid_search_rf = GridSearchCV(rf_regressor, param_grid_rf, cv=5, scoring='r2')
grid_search_rf.fit(X_train_wqi_scaled, y_train_wqi)
# Get the best model from grid search
best_rf_regressor = grid_search_rf.best_estimator_
# Testing metrics for Random Forest Regressor
test_preds_wqi_rf = best_rf_regressor.predict(X_test_wqi_scaled)
test_r2_rf = r2_score(y_test_wqi, test_preds_wqi_rf) * 100
test_mae_rf = mean_absolute_error(y_test_wqi, test_preds_wqi_rf)
test_mse_rf = mean_squared_error(y_test_wqi, test_preds_wqi_rf)
print("Testing Metrics:")
print(f"R2: {test_r2_rf:.4f}")
print(f"MAE: {test_mae_rf:.4f}")
print(f"MSE: {test_mse_rf:.4f}")
# Define a pipeline for Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)
param_grid_rf_clf = {
    'max_depth': [3, 5],
    'min_samples_split': [5, 10],
    'min_samples_leaf': [4, 6],
    'n_estimators': [50, 75]
}
grid_search_rf_clf = GridSearchCV(rf_classifier, param_grid_rf_clf, cv=5, scoring='accuracy')
grid_search_rf_clf.fit(X_train_quality, y_train_quality)
# Get the best model from grid search
best_rf_classifier = grid_search_rf_clf.best_estimator_
# Testing metrics for Random Forest Classifier
test_preds_quality_rf = best_rf_classifier.predict(X_test_quality)
print("Testing Metrics:")
print(f"Accuracy: {accuracy_score(y_test_quality, test_preds_quality_rf):.4f}")
print(f"Precision: {precision_score(y_test_quality, test_preds_quality_rf, average='weighted'):.4f}")
```

```
print(f"Recall: {recall_score(y_test_quality, test_preds_quality_rf, average='weighted'):.4f}")

print(f"F1 Score: {f1_score(y_test_quality, test_preds_quality_rf, average='weighted'):.4f}")

# Create a DataFrame to show Actual vs Predicted WQI values

wqi_comparison = pd.DataFrame({

   'Row Number': range(len(y_test_wqi)),

   'Actual WQI': y_test_wqi,

   'Predicted WQI (Random Forest)': test_preds_wqi_rf

})

#random forest bar

# Assuming X and y are already defined

# Example splitting the data into training and testing sets

X_train_wqi, X_test_wqi, y_train_wqi, y_test_wqi = train_test_split(X, y_wqi, test_size=0.2, random_state=42)

X_train_quality, X_test_quality, y_train_quality, y_test_quality = train_test_split(X, y_quality, test_size=0.2,
random_state=42)

# Standardize the features

scaler_wqi = StandardScaler()

X_train_wqi_scaled = scaler_wqi.fit_transform(X_train_wqi)

X_test_wqi_scaled = scaler_wqi.transform(X_test_wqi)

# Define and fit a Random Forest Classifier

rf_classifier = RandomForestClassifier(random_state=42)

param_grid_rf_clf = {

   'max_depth': [3, 5],

   'min_samples_split': [5, 10],

   'min_samples_leaf': [4, 6],

   'n_estimators': [50, 75]

}

grid_search_rf_clf = GridSearchCV(rf_classifier, param_grid_rf_clf, cv=5, scoring='accuracy')

grid_search_rf_clf.fit(X_train_quality, y_train_quality)

# Get the best model from grid search

best_rf_classifier = grid_search_rf_clf.best_estimator_

# Calculate metrics for training data

train_preds_quality_rf = best_rf_classifier.predict(X_train_quality)
```

```
train_accuracy_rf = accuracy_score(y_train_quality, train_preds_quality_rf)
train_precision_rf = precision_score(y_train_quality, train_preds_quality_rf, average='weighted')
train_recall_rf = recall_score(y_train_quality, train_preds_quality_rf, average='weighted')
train_f1_rf = f1_score(y_train_quality, train_preds_quality_rf, average='weighted')
# Calculate metrics for testing data
test_preds_quality_rf = best_rf_classifier.predict(X_test_quality)
test_accuracy_rf = accuracy_score(y_test_quality, test_preds_quality_rf)
test_precision_rf = precision_score(y_test_quality, test_preds_quality_rf, average='weighted')
test_recall_rf = recall_score(y_test_quality, test_preds_quality_rf, average='weighted')
test_f1_rf = f1_score(y_test_quality, test_preds_quality_rf, average='weighted')
# Plot confusion matrix for testing data
test_conf_matrix = confusion_matrix(y_test_quality, test_preds_quality_rf)
plt.figure(figsize=(10, 6))
sns.heatmap(test_conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix (Testing)')
plt.show()
# Plotting bar chart for evaluation metrics
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
train_metrics_rf = [train_accuracy_rf, train_precision_rf, train_recall_rf, train_f1_rf]
test_metrics_rf = [test_accuracy_rf, test_precision_rf, test_recall_rf, test_f1_rf]
x = np.arange(len(metrics))  # the label locations
width = 0.35  # the width of the bars
fig, ax = plt.subplots(figsize=(10, 6))
rects1 = ax.bar(x - width/2, train_metrics_rf, width, label='Training')
rects2 = ax.bar(x + width/2, test_metrics_rf, width, label='Testing')
# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_xlabel('Metrics')
ax.set_ylabel('Scores')
ax.set_title('Evaluation Metrics for Training and Testing')
ax.set_xticks(x)
```

```
ax.set_xticklabels(metrics)
ax.legend()
# Attach a text label above each bar in rects, displaying its height.
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate(f'{height:.4f}',
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')


autolabel(rects1)
autolabel(rects2)
fig.tight_layout()
plt.show()
# Assuming best_rf_regressor is defined and trained as per your earlier code
# Plot Actual vs. Predicted for Training and Testing Set
train_preds_wqi = best_rf_regressor.predict(X_train_wqi_scaled)
test_preds_wqi = best_rf_regressor.predict(X_test_wqi_scaled)
plt.figure(figsize=(12, 8))
# Training set
plt.scatter(y_train_wqi, train_preds_wqi, color='blue', alpha=0.5, label='Training Set')
# Testing set
plt.scatter(y_test_wqi, test_preds_wqi, color='green', alpha=0.5, label='Testing Set')
# Line of best fit
plt.plot([min(y_train_wqi.min(), y_test_wqi.min()), max(y_train_wqi.max(), y_test_wqi.max())],
         [min(y_train_wqi.min(), y_test_wqi.min()), max(y_train_wqi.max(), y_test_wqi.max())],
         color='red', linewidth=2)
plt.xlabel('Actual WQI')
plt.ylabel('Predicted WQI')
plt.title('Actual vs Predicted WQI')
```

```
plt.legend()
plt.show()
# Decision Tree for WQI Prediction
dt_regressor = DecisionTreeRegressor(random_state=42)
dt_regressor.fit(X_train_wqi, y_train_wqi)
# Predictions for training and testing sets
train_preds_wqi = dt_regressor.predict(X_train_wqi)
test_preds_wqi = dt_regressor.predict(X_test_wqi)
# Evaluation metrics for testing set
print("Testing Metrics:")
print(f"R2: {(r2_score(y_test_wqi, test_preds_wqi)*100):.4f}")
print(f"MAE: {mean_absolute_error(y_test_wqi, test_preds_wqi):.4f}")
print(f"MSE: {mean_squared_error(y_test_wqi, test_preds_wqi):.4f}")
# Decision Tree for WaterQuality Classification
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train_quality, y_train_quality)
# Predictions for training and testing sets
train_preds_quality = dt_classifier.predict(X_train_quality)
test_preds_quality = dt_classifier.predict(X_test_quality)
# Evaluation metrics for testing set
print("Testing Metrics:")
print(f"Accuracy: {accuracy_score(y_test_quality, test_preds_quality):.4f}")
print(f"Precision: {precision_score(y_test_quality, test_preds_quality, average='weighted'):.4f}")
print(f"Recall: {recall_score(y_test_quality, test_preds_quality, average='weighted'):.4f}")
print(f"F1 Score: {f1_score(y_test_quality, test_preds_quality, average='weighted'):.4f}")
# Actual vs. Predicted for Training and Testing Set (WQI Regression)
plt.figure(figsize=(12, 8))
# Training set
plt.scatter(y_train_wqi, train_preds_wqi, color='blue', alpha=0.5, label='Training Set')
# Testing set
plt.scatter(y_test_wqi, test_preds_wqi, color='green', alpha=0.5, label='Testing Set')
# Line of best fit
```

```
    plt.plot([min(y_train_wqi.min(), y_test_wqi.min()), max(y_train_wqi.max(), y_test_wqi.max())],
             [min(y_train_wqi.min(), y_test_wqi.min()), max(y_train_wqi.max(), y_test_wqi.max())],
             color='red', linewidth=2)

    plt.xlabel('Actual WQI')
    plt.ylabel('Predicted WQI')
    plt.title('Actual vs Predicted WQI')
    plt.legend()
    plt.show()
# Confusion Matrix for Testing Set
    cm_test = confusion_matrix(y_test_quality, test_preds_quality)
    disp_test = ConfusionMatrixDisplay(confusion_matrix=cm_test)
    disp_test.plot(cmap=plt.cm.Blues)
    plt.title('Confusion Matrix for Testing Set')
    plt.show()
    # Store evaluation metrics in lists
    metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
    training_scores = [
        accuracy_score(y_train_quality, train_preds_quality),
        precision_score(y_train_quality, train_preds_quality, average='weighted'),
        recall_score(y_train_quality, train_preds_quality, average='weighted'),
        f1_score(y_train_quality, train_preds_quality, average='weighted')
    ]
    testing_scores = [
        accuracy_score(y_test_quality, test_preds_quality),
        precision_score(y_test_quality, test_preds_quality, average='weighted'),
        recall_score(y_test_quality, test_preds_quality, average='weighted'),
        f1_score(y_test_quality, test_preds_quality, average='weighted')
    ]
    x = np.arange(len(metrics))  # label locations
    width = 0.35  # width of the bars
    fig, ax = plt.subplots(figsize=(10, 6))
```

```
bars1 = ax.bar(x - width/2, training_scores, width, label='Training')
bars2 = ax.bar(x + width/2, testing_scores, width, label='Testing')
# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_xlabel('Metrics')
ax.set_ylabel('Scores')
ax.set_title('Training and Testing Scores by Metrics')
ax.set_xticks(x)
ax.set_xticklabels(metrics)
ax.legend()
# Attach a text label above each bar in bars, displaying its height.
def autolabel(bars):
    """Attach a text label above each bar in bars, displaying its height."""
    for bar in bars:
        height = bar.get_height()
        ax.annotate(f'{height:.2f}',
                    xy=(bar.get_x() + bar.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(bars1)
autolabel(bars2)
fig.tight_layout()
plt.show()
```

#Comparison between algorithms

```
import pandas as pd
# Define the testing metrics for each model
models = [
    "KNN Regressor",
    "SVM Regressor",
    "Multivariate Linear Regression",
```

```
    "Random Forest Regressor",
    "Decision Tree Regressor"
]
r2_scores = [
    82.7722,
    45.4213,
    45.1635,
    94.5030,
    90.9593
]
mae_scores = [
    2.0646,
    3.3562,
    3.8735,
    0.7514,
    0.7115
]

mse_scores = [
    6.5137,
    20.6360,
    20.7335,
    2.0784,
    3.4182
]
# Create a DataFrame with the above data
df = pd.DataFrame({
    "Model": models,
    "R2": r2_scores,
    "MAE": mae_scores,
    "MSE": mse_scores
})
```

```
# Print the DataFrame
print(df)
# If using Jupyter Notebook or similar, display the DataFrame
try:
    from IPython.display import display
    display(df)
except ImportError:
    pass
import pandas as pd
# Data for the table
data = {
    'Model': [
        'KNN Classifier',
        'SVM Classifier',
        'Random Forest Classifier',
        'Decision Tree Classifier'
    ],
    'Accuracy': [
        0.8462,
        0.8077,
        0.9615,
        0.9615
    ],
    'Precision': [
        0.8846,
        0.8220,
        0.9641,
        0.9641
    ],
    'Recall': [
        0.8462,
        0.8077,
```

```
        0.9615,
        0.9615
    ],
    'F1 Score': [
        0.8443,
        0.8033,
        0.9614,
        0.9614
    ]
}
# Creating the DataFrame
df = pd.DataFrame(data)
# Displaying the table
print(df)
#plotting criteria
import matplotlib.pyplot as plt
# Define the algorithms
algorithms = ['KNN', 'SVM', 'Random Forest', 'Decision Tree']

# Define the testing metrics for each algorithm
accuracy_scores = [0.8462, 0.8077, 0.9615, 0.9615]
precision_scores = [0.8846, 0.8220, 0.9641, 0.9641]
recall_scores = [0.8462, 0.8077, 0.9615, 0.9615]
f1_scores = [0.8443, 0.8033, 0.9614, 0.9614]
# Plotting accuracy
plt.figure(figsize=(10, 6))
plt.bar(algorithms, accuracy_scores, color='skyblue')
plt.xlabel('Algorithms')
plt.ylabel('Accuracy')
plt.title('Testing Accuracy for Different Algorithms')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
```

```
plt.show()
# Plotting precision
plt.figure(figsize=(10, 6))
plt.bar(algorithms, precision_scores, color='lightgreen')
plt.xlabel('Algorithms')
plt.ylabel('Precision')
plt.title('Testing Precision for Different Algorithms')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
# Plotting recall
plt.figure(figsize=(10, 6))
plt.bar(algorithms, recall_scores, color='salmon')
plt.xlabel('Algorithms')
plt.ylabel('Recall')
plt.title('Testing Recall for Different Algorithms')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
# Plotting F1 score
plt.figure(figsize=(10, 6))
plt.bar(algorithms, f1_scores, color='lightcoral')
plt.xlabel('Algorithms')
plt.ylabel('F1 Score')
plt.title('Testing F1 Score for Different Algorithms')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
# Define the algorithms
algorithms = ['Multivariate Linear Regression', 'SVM', 'KNN', 'Decision Tree', 'Random Forest']
# Define the testing metrics for each algorithm
r2_scores = [45.1635, 35.2248, 30.5440, 89.9152, 94.0460]
```

```
mae_scores = [3.8735, 4.0919, 3.8411, 0.7631, 0.6572]
mse_scores = [20.7335, 24.4912, 26.2610, 3.8130, 2.2512]
# Plotting R2
plt.figure(figsize=(10, 6))
plt.bar(algorithms, r2_scores, color='skyblue')
plt.xlabel('Algorithms')
plt.ylabel('R2 Score')
plt.title('R2 Score for Different Algorithms')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
# Plotting MAE
plt.figure(figsize=(10, 6))
plt.bar(algorithms, mae_scores, color='lightgreen')
plt.xlabel('Algorithms')
plt.ylabel('MAE')
plt.title('MAE for Different Algorithms')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
# Plotting MSE
plt.figure(figsize=(10, 6))
plt.bar(algorithms, mse_scores, color='salmon')
plt.xlabel('Algorithms')
plt.ylabel('MSE')
plt.title('MSE for Different Algorithms')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

#comparison between WQI Values

```
# Assuming X_train_wqi, X_test_wqi, y_train_wqi, y_test_wqi are already defined
# Scaling the data
```

```
scaler = StandardScaler()
X_train_wqi_scaled = scaler.fit_transform(X_train_wqi)
X_test_wqi_scaled = scaler.transform(X_test_wqi)
# Multivariate Linear Regression for WQI Prediction (same as Linear Regression)
linear_regressor = LinearRegression()
linear_regressor.fit(X_train_wqi_scaled, y_train_wqi)
test_preds_wqi_lr = linear_regressor.predict(X_test_wqi_scaled)
# SVM for WQI Prediction
svm_regressor = SVR()
svm_regressor.fit(X_train_wqi_scaled, y_train_wqi)
test_preds_wqi_svm = svm_regressor.predict(X_test_wqi_scaled)
# KNN for WQI Prediction
knn_regressor = KNeighborsRegressor()
knn_regressor.fit(X_train_wqi_scaled, y_train_wqi)
test_preds_wqi_knn = knn_regressor.predict(X_test_wqi_scaled)

# Decision Tree for WQI Prediction
dt_regressor = DecisionTreeRegressor()
dt_regressor.fit(X_train_wqi_scaled, y_train_wqi)
test_preds_wqi_dt = dt_regressor.predict(X_test_wqi_scaled)
# Random Forest for WQI Prediction
rf_regressor = RandomForestRegressor()
rf_regressor.fit(X_train_wqi_scaled, y_train_wqi)
test_preds_wqi_rf = rf_regressor.predict(X_test_wqi_scaled)
# Evaluation metrics for all models (testing phase)
models = {
    'Multivariate Linear Regression': test_preds_wqi_lr,
    'SVM': test_preds_wqi_svm,
    'KNN': test_preds_wqi_knn,
    'Decision Tree': test_preds_wqi_dt,
    'Random Forest': test_preds_wqi_rf
}
```

```
print("Evaluation Metrics for Testing Phase:")
for model_name, test_preds in models.items():
    print(f"\n{model_name}:")
    print(f"R2: {(r2_score(y_test_wqi, test_preds)*100):.4f}")
    print(f"MAE: {mean_absolute_error(y_test_wqi, test_preds):.4f}")
    print(f"MSE: {mean_squared_error(y_test_wqi, test_preds):.4f}")
# Create a DataFrame to show Actual vs Predicted WQI values for all models
wqi_comparison = pd.DataFrame({
    'Row Number': range(len(y_test_wqi)),
    'Actual WQI': y_test_wqi,
    'Multivariate Linear Regression': test_preds_wqi_lr,
    'SVM': test_preds_wqi_svm,
    'KNN': test_preds_wqi_knn,
    'Decision Tree': test_preds_wqi_dt,
    'Random Forest': test_preds_wqi_rf
})
# Display the table
print("\nActual vs Predicted WQI values:")
print(wqi_comparison)
# Plot Actual vs Predicted WQI values for all models
plt.figure(figsize=(12, 8))
plt.scatter(range(len(y_test_wqi)), y_test_wqi, color='blue', label='Actual WQI', alpha=0.6)
for model_name, test_preds in models.items():
    plt.scatter(range(len(y_test_wqi)), test_preds, alpha=0.6, label=f'{model_name} Predicted WQI')
plt.xlabel('Row Number')
plt.ylabel('WQI')
plt.title('Row Number vs Actual and Predicted WQI')
plt.legend()
plt.show()
```