

PROJECT REPORT
WATER QUALITY PREDICATION

Submitted in partial fulfilment of the requirements for the degree of

Bachelor of Engineering

in

Computer of Engineering

Submitted By

Avishek Mandal – 102203700

Yash Yadav– 102118035

Under the supervision of:

Dr. Arun Singh Pundir

Assistant Professor CSED TIET



**Computer Science and Engineering Department Thapar Institute of
Engineering and Technology Patiala – 147004**

November 2024

Table of Contents

Contents

1. Introduction	3
2. Related Work and Literature Review.....	4
3. Why we used this Dataset.....	5
4. Data Preprocessing.....	7
5. Machine Learning Models	13
6. Limitations	17
7. Conclusion and Future Work.....	19
8. Predictive System and Deployment	21
9. Limitations	23
10. Conclusion and Future Work.....	25

1. Introduction

Water quality plays a crucial role in maintaining public health, and the ability to predict whether water is safe for drinking is essential for environmental and health safety. Water quality is determined by various chemical and physical parameters, such as pH, turbidity, hardness, and concentrations of specific elements like sulfates, nitrates, and metals. Monitoring and ensuring water quality requires continuous assessment of these parameters.

In this project, we leverage machine learning algorithms to predict the potability (safe to drink or not) of water based on several water quality attributes. The goal is to create a predictive model that can classify water as potable or non-potable based on its chemical properties.

We used a dataset containing these water quality parameters, and applied multiple machine learning algorithms, such as Logistic Regression, Decision Tree, Random Forest, and K-Nearest Neighbors (KNN), to predict water potability. The model's accuracy is evaluated, and the best performing model is selected for deployment in real-world applications, providing valuable insights for water safety and quality monitoring.

2. Related Work and Literature Review

- **Lei Li, Shuming Rong, Rui Wang,** <https://www.sciencedirect.com/science/article/abs/pii/S1385894720328011>
- **Soma Safeer** <https://www.sciencedirect.com/science/article/abs/pii/S2214714422004184>.
- **DR ARVINDER KOUR1, V VIDYASAGAR** https://www.researchgate.net/profile/Arvinder-Mehta/publication/382799499_ENHANCING_WATER_PURIFICATION_EFFICIENCY_THROUGH_MACHINE_LEARNING-DRIVEN_MXENE_FUNCTIONALIZATION/links/66ac80fa299c327096a35c96/ENHANCING-WATER-PURIFICATION-EFFICIENCY-THROUGH-MACHINE-LEARNING-DRIVEN-MXENE-FUNCTIONALIZATION.pdf
- **An IEEE publication**
<https://ieeexplore.ieee.org/document/10017579>
- **IWA Publishing**
<https://iwaponline.com/wpt/article/18/12/3004/98806/Drinking-water-potability-prediction-using-machine>

3. Why we used this Dataset

This dataset, related to water potability, is typically used to determine whether water is safe for drinking based on various physical and chemical properties. The main objective of using this dataset is to build models or conduct analysis to predict water quality and assess its potability. Here are the typical reasons for using this dataset:

1. Predictive Modeling (Classification)

The dataset is often used to train machine learning models that can classify water as potable (safe to drink) or non-potable (unsafe to drink) based on its features. The target variable here is the Potability column (binary: 0 or 1).

2. Data Exploration and Analysis

Researchers or data scientists analyze the different water properties (like pH, hardness, solids, etc.) to:

Understand how these features affect water quality.

Identify correlations and trends among different chemical properties.

Spot anomalies, such as unusual combinations of properties in non-potable water.

3. Feature Engineering

By studying these water parameters, feature engineering can be performed to derive new insights or improve the model by creating new relevant features. For example, combinations of chemical concentrations may help in better predicting potability.

4. Water Quality Research

The dataset helps researchers or environmental scientists understand the conditions that make water safe or unsafe, aiding in water quality monitoring.

5. Imputation and Handling Missing Data

Since the dataset has missing values (e.g., in pH and Sulfate), it provides a good opportunity to practice techniques for handling missing data, like mean/mode imputation, or more advanced methods like KNN imputation or regression imputation.

6. Real-World Application

In environmental engineering, public health, and government sectors, data like this is essential for ensuring access to clean water. Modeling water potability can help governments and organizations predict water quality in various regions and improve drinking water standards.

4. Data Preprocessing

Data preprocessing is a crucial step to ensure that the dataset is clean, structured, and ready for analysis or model training. This involves checking for missing values, handling outliers, and visualizing data to understand its distribution and correlation. The following steps were followed in preprocessing the water purity dataset:

1. Loading the Dataset

The water quality dataset is loaded using the pandas `read_csv` function, which loads the data into a DataFrame (df). This dataset contains several features such as pH, hardness, sulfate, turbidity, and the target variable, Potability, which indicates whether the water is safe for drinking.

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import plotly.express as px
5 import seaborn as sns
6
7 from pyarrow.compute import scalar
8
9 from sklearn.preprocessing import StandardScaler
10 from sklearn.model_selection import train_test_split
11 from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
12
13 ✓ [2] 17ms
14
15 df = pd.read_csv('data/water_potability .csv')
16 df.head()
17 [202]
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	0
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180013	56.329076	4.500656	0
2	8.099124	224.236259	19909.541732	9.275804	NaN	410.606213	16.868637	66.420093	3.055934	0
3	8.316766	214.373394	22018.417441	8.059332	356.806136	363.266516	18.436524	100.341674	4.628771	0
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	0

```
1 df.columns
2 [203]
3 Index(['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity',
4        'Organic_carbon', 'Trihalomethanes', 'Turbidity', 'Potability'],
5        dtype='object')
```

1.1 Data Preprocessing

```
#Data preprocessing

1 df.describe()
[204]
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
count	2785.000000	3276.000000	3276.000000	3276.000000	2495.000000	3276.000000	3276.000000	3114.000000	3276.000000	3276.0000
mean	7.080795	196.369496	22014.092526	7.122277	333.775777	426.205111	14.284970	66.396293	3.966786	0.3901
std	1.594320	32.879761	8768.570828	1.583085	41.416840	80.824064	3.308162	16.175008	0.780382	0.4878
min	0.000000	47.432000	320.942611	0.352000	129.000000	181.483754	2.200000	0.738000	1.450000	0.0000
25%	6.093092	176.850538	15666.690297	6.127421	307.699498	365.734414	12.065801	55.844536	3.439711	0.0000
50%	7.036752	196.967627	20927.833607	7.130299	333.073546	421.884968	14.218338	66.622485	3.955028	0.0000
75%	8.062066	216.667456	27332.762127	8.114887	359.950170	481.792304	16.557652	77.337473	4.500320	1.0000
max	14.000000	323.124000	61227.196008	13.127000	481.030642	753.342620	28.300000	124.000000	6.739000	1.0000

Click to edit an empty Markdown cell

```
1 df.info
[205]
```

```
<bound method DataFrame.info of
0      NaN    204.890455    20791.318981    7.300212    368.516441
1    3.716080    129.422921    18630.057858    6.635246      NaN
2    8.099124    224.236259    19909.541732    9.275884      NaN
3    8.316766    214.373394    22018.417441    8.059332    356.886136
4    9.092223    181.101509    17978.986339    6.546600    310.135738
...      ...      ...      ...      ...      ...
3271    4.668102    193.681735    47580.991603    7.166639    359.948574
3272    7.808856    193.553212    17329.802160    8.061362      NaN
3273    9.419510    175.762646    33155.578218    7.350233      NaN
3274    5.126763    239.603758    11983.869376    6.303357      NaN
3275    7.874671    195.102299    17404.177061    7.509306      NaN
```

2. Checking for Missing Values

The first step in preprocessing is to check for missing values in the dataset. This is essential as missing data can distort the accuracy of machine learning models. We use `df.isnull()` to identify missing values and visualize the distribution of missing data using a heatmap.

```
1 df.info
[205]
```

```
<bound method DataFrame.info of
0      NaN    204.890455    20791.318981    7.300212    368.516441
1    3.716080    129.422921    18630.057858    6.635246      NaN
2    8.099124    224.236259    19909.541732    9.275884      NaN
3    8.316766    214.373394    22018.417441    8.059332    356.886136
4    9.092223    181.101509    17978.986339    6.546600    310.135738
...      ...      ...      ...      ...      ...
3271    4.668102    193.681735    47580.991603    7.166639    359.948574
3272    7.808856    193.553212    17329.802160    8.061362      NaN
3273    9.419510    175.762646    33155.578218    7.350233      NaN
3274    5.126763    239.603758    11983.869376    6.303357      NaN
3275    7.874671    195.102299    17404.177061    7.509306      NaN
```

```
1 df.isnull().sum()
[206]
```

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

```
1 plt.figure(figsize=(12,8))
```


This heatmap provides a clear view of which features have missing data, helping us decide whether to impute or drop those values.

3. Handling Missing Data

After identifying missing values, we apply various strategies for dealing with them:

- Imputation: For features with a small percentage of missing values, such as pH, missing values are replaced with the mean or median of the respective column.
- Advanced Imputation: For columns like Sulfate and Trihalomethanes, where a significant proportion of data is missing, we can use advanced imputation techniques (like KNN imputation or regression imputation) or decide to drop the column if it doesn't significantly impact model performance.

Example of filling missing values with the median:

```
df['pH'] = df['pH'].fillna(df['pH'].median())  
df['Sulfate'] = df['Sulfate'].fillna(df['Sulfate'].median())
```

4. Exploratory Data Analysis (EDA)

Once missing values are handled, we perform an exploratory analysis to understand the relationships between the features in the dataset. This involves creating a correlation matrix and visualizing it using a heatmap:

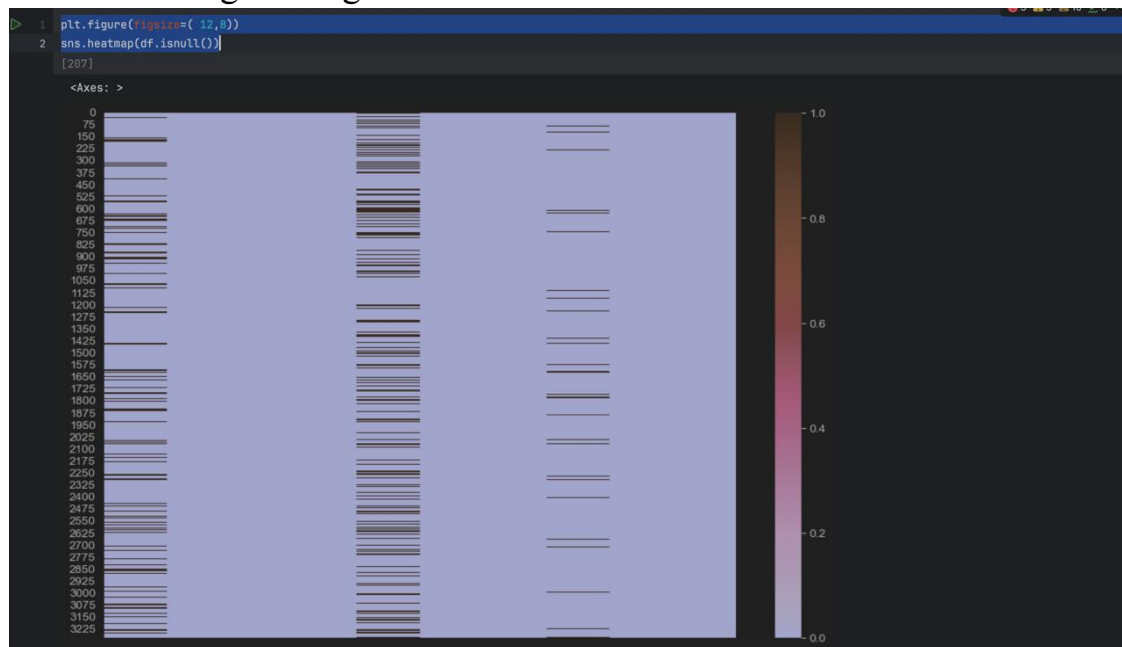
```
plt.figure(figsize=(12, 8))  
sns.heatmap(df.corr(), annot=True, cmap="coolwarm")
```

This helps identify how features like pH, sulfate, and turbidity are correlated with the target variable, Potability. Strong correlations between features and the target are critical for model performance.

5.Data Visualsization

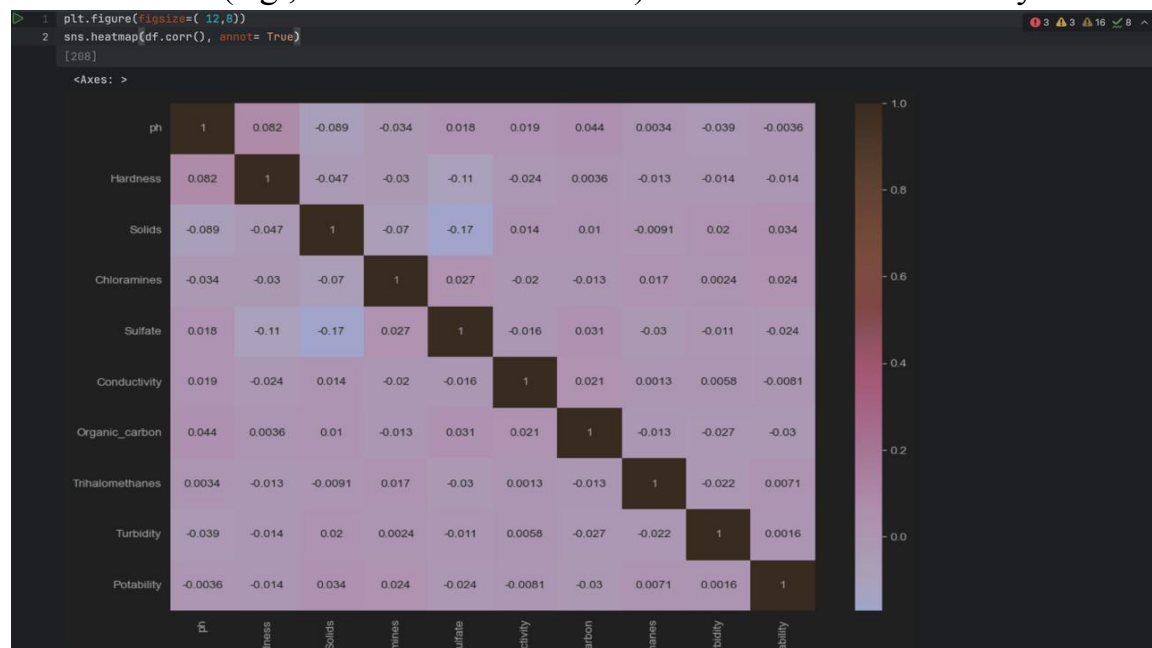
Before diving into modeling, data visualization plays an essential role in understanding the underlying structure and patterns in the data. Visualizations can help identify outliers, patterns, correlations, and distributions of the features, which are crucial for making informed decisions about data cleaning

and feature engineering.



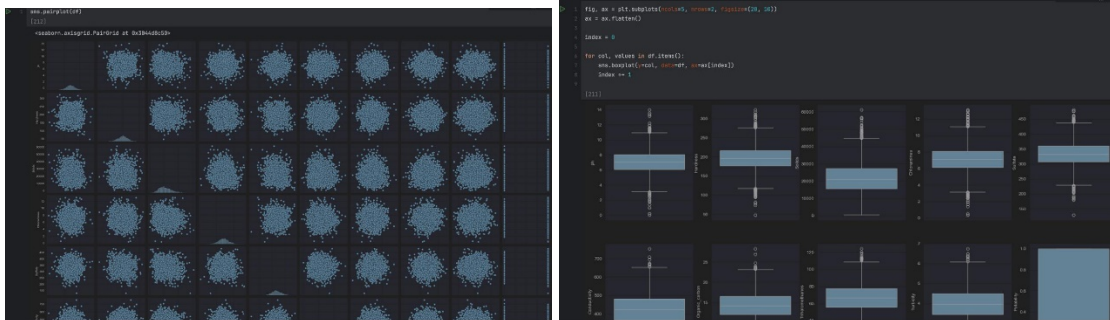
- Correlation Matrix Visualization:

A heatmap is generated to visualize the correlation between different features in the dataset. This allows us to identify highly correlated features, which can help in selecting or transforming variables for better model performance. Features that are highly correlated with each other might need to be handled (e.g., removed or combined) to avoid multicollinearity.



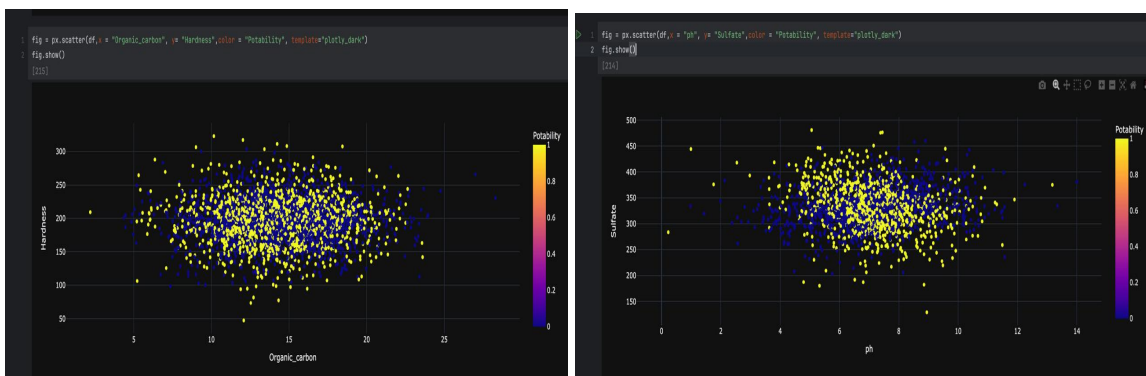
- Outliers:

We are checking for Outliers in data by using box plot and pair plot



- Relationship between Features

Now we actually creates an interactive scatter plot using the plotly.express library to visualize the relationship between the features ph and Sulfate, with points colored by the Potability column (indicating whether the water is potable or not).



- Pattern in Missing Data

Identify columns or rows with significant amounts of missing values. Observe if missing data occurs in a specific pattern (e.g., sequentially or in groups).



5. Feature Scaling and Data Training

Since the features in the dataset vary in scale (e.g., pH is between 0-14, while turbidity can be much larger), it is essential to scale the features to a standard range. This series of commands performs data preparation for machine learning by creating input features (x) and labels (y), scaling the features, and splitting the data into training and testing sets..

Purpose: Standardizes the features in x to have a mean of 0 and a standard deviation of 1. This ensures that features with large values do not dominate those with smaller values during model fits. Fits the StandardScaler to x (calculates mean and standard deviation). Transforms x by scaling it.

6. Data Splitting

Finally, the dataset is split into training and testing sets. This allows us to evaluate the performance of the model on unseen data. The split is done using `train_test_split` from scikit-learn:

```
df.head()
[222]
```

	ph	Hardness	Soilids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	7.080795	284.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	0
1	3.716080	129.422921	18630.057858	6.635246	333.775777	592.885359	15.180813	56.329076	4.500656	0
2	8.099124	224.236259	19909.541732	9.275884	333.775777	418.606213	16.868637	66.420093	3.055934	0
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.341674	4.628771	0
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	0

```
1 x = df.drop("Potability", axis=1)
2 y = df["Potability"]
[223]
```

```
1 x.shape, y.shape
[224]
```

```
((3276, 9), (3276,))
```

```
1 scaler = StandardScaler()
2 x = scaler.fit_transform(x)
3 x
[225]
```

```
array([[ -6.04313345e-16,  2.59194711e-01, -1.39470871e-01, ...,
        -1.18065057e+00,  1.30614943e+00, -1.28629758e+00],
       [-2.28933938e-00, -2.03641367e+00, -3.85986650e-01, ...,
        2.70597240e-01, -6.38479983e-01,  6.04217891e-01],
       [ 6.92867789e-01,  8.47664833e-01, -2.40047337e-01, ...,
        7.81116857e-01,  1.50940084e-03, -1.16736546e+00],
       ...,
       [ 1.59125368e+00, -6.26829230e-01,  1.27080989e+00, ...,
        2.81320271e-01,  2.40710024e-01,  0.51001700e+01]])
```

```
LOGISTIC REGRESSION

1 from sklearn.linear_model import LogisticRegression
2
3 model_lr = LogisticRegression()
✓ [35] < 10 ms

1 model_lr.fit(x_train,y_train)
✓ [36] 33ms

LogisticRegression()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

1 # Making Prediction
2 pred_lr = model_lr.predict(x_test)
✓ [37] < 10 ms

1 accuracy_score_lr = accuracy_score(y_test,pred_lr)
2 accuracy_score_lr
✓ [38] < 10 ms

0.586896243982439
```

5.2 Decision Tree Classifier

- Overview: The Decision Tree algorithm splits the dataset into smaller subsets based on feature values to make predictions. It works well for non-linear data and interpretable decision-making.
- Accuracy: The Decision Tree model achieved an accuracy of 60.00%.

```
DECISION TREE CLASIFIER

1 from sklearn.tree import DecisionTreeClassifier
2
3 model_dt = DecisionTreeClassifier(max_depth = 4)
✓ [39] 193ms

1 #Training
2 model_dt.fit(x_train, y_train)
✓ [40] 25ms

DecisionTreeClassifier(max_depth=4)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

1 pred_dt = model_dt.predict(x_test)
✓ [41] < 10 ms

1 accuracy_score_dt = accuracy_score(y_test,pred_dt)
2 accuracy_score_dt
✓ [42] < 10 ms

0.6051829268292683

1 #confusion matrix
2 cm2 = confusion_matrix(y_test,pred_dt)
3 cm2
✓ [43] < 10 ms

2 rows x 2 columns
0 372 11
```

5.3 Random Forest Classifier

- Overview: Random Forest is an ensemble learning method that combines multiple decision trees. It improves prediction accuracy by averaging predictions or using majority voting.

- Accuracy: The Random Forest Classifier achieved the highest accuracy of 63.00%, making it the best-performing model in this project.

```
RANDOM FOREST CLASSIFIER

1 from sklearn.ensemble import RandomForestClassifier
2
3 model_rf = RandomForestClassifier()
✓ [44] 256ms

1 #TRAINING THE MODEL RF
2
3 import numpy as np
4
5 x_train = np.array(x_train)
6 y_train = np.array(y_train)
7
✓ [45] < 10 ms

1 model_rf.fit(x_train, y_train)
2
✓ [46] 1s 129ms

RandomForestClassifier()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

1 pred_rf = model_rf.predict(x_test)
2
✓ [47] 14ms

1 accuracy_score_rf = accuracy_score(y_test, pred_rf)
2 accuracy_score_rf*100
✓ [48] < 10 ms
63.5670731707317
```

5.4 K-Nearest Neighbors (KNN)

- Overview: KNN is a non-parametric, instance-based learning algorithm. It classifies new data points based on the majority class among its nearest neighbors.
- Accuracy: The KNN model achieved an accuracy of 60.00%, similar to the Decision Tree.

```
USING KNN MODEL

1 from sklearn.neighbors import KNeighborsClassifier
2
3 #model_knn = KNeighborsClassifier()
✓ [50] 11ms

4 for i in range(4,15):
5     model_knn = KNeighborsClassifier(n_neighbors=i)
6     model_knn.fit ( x_train, y_train)
7     pred_knn = model_knn.predict(x_test)
8     accuracy_score_knn = accuracy_score(y_test,pred_knn)
9     print(i,accuracy_score_knn)
10
11 ✓ [51] 454ms
12 4 0.6189924399243992
13 5 0.614329268292683
14 6 0.614329268292683
15 7 0.6112804878048781
16 8 0.6158536585365854
17 9 0.60975697569756
18 10 0.6112804878048781
19 11 0.6036585365853658
20 12 0.6112804878048781
21 13 0.6173789487894879
22 14 0.6173789487894879

23 model_knn = KNeighborsClassifier(n_neighbors=11)
24 model_knn.fit(x_train, y_train)
25 pred_knn = model_knn.predict(x_test)
26 accuracy_score_knn = accuracy_score(y_test, pred_knn)
27 print(accuracy_score_knn)
```

5.5 Performance Comparison

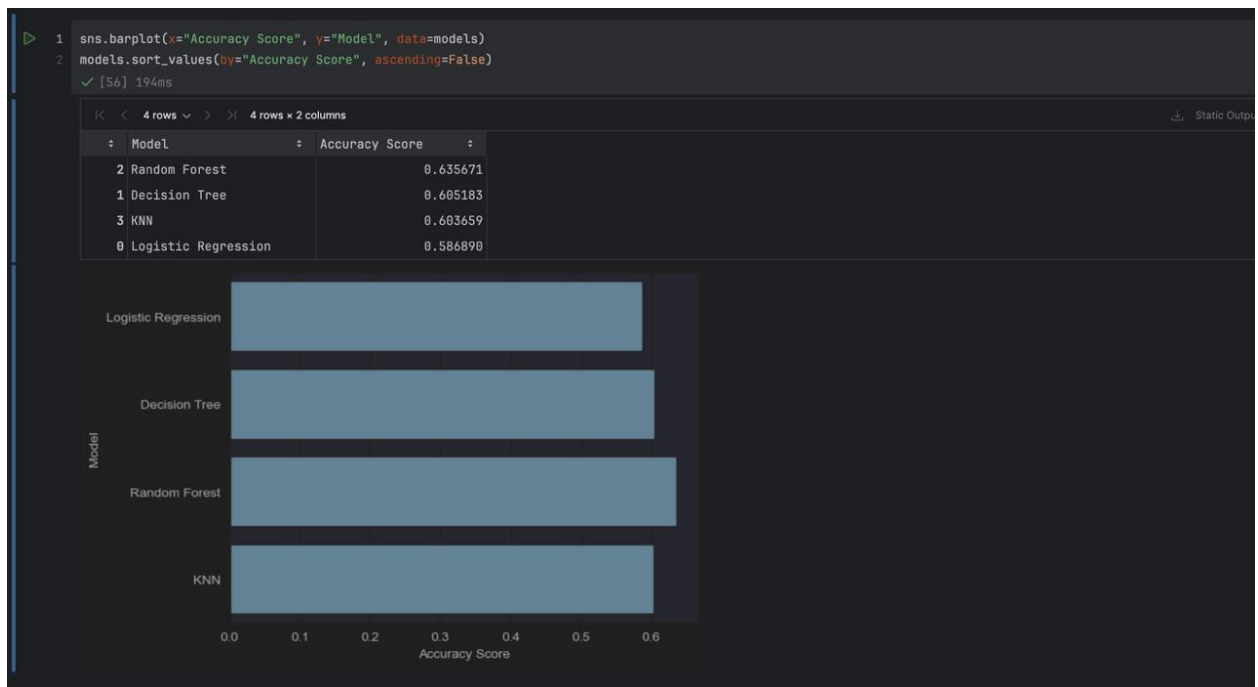
A summary of the performance of all models is as follows:

Model	Accuracy Score (%)
Logistic Regression	58.85
Decision Tree	60.00
Random Forest	63.00
K-Nearest Neighbors	60.00

The performance comparison highlights that the Random Forest Classifier outperformed other models in terms of accuracy.

5.6 Visualization of Model Performance

To better understand the performance of each model, we created a bar plot comparing their accuracy scores. This visualization highlights the superior performance of the Random Forest Classifier and provides a quick reference for comparing the models.



6. Limitations

While the analysis and predictive models developed for water purity detection provide valuable insights, there are several limitations that must be acknowledged:

6.1 Data Limitations

- **Sample Size:** The dataset used for training and testing may not comprehensively represent the diverse range of water quality conditions globally. This could limit the generalizability of the results.
- **Imbalance in Classes:** The dataset has more non-potable water samples (class 0) than potable samples (class 1), which may bias the models' predictions.

6.2 Missing Data

- Some features, such as pH, Sulfate, and Trihalomethanes, contained a significant proportion of missing values. While imputation techniques were applied, they may have introduced some inaccuracies in the dataset.

6.3 Feature Limitations

- The dataset does not include certain critical parameters, such as biological contaminants, heavy metals, or real-time environmental conditions, which are essential for a more comprehensive analysis of water quality.

6.4 Model Limitations

- **Overfitting:** Certain models, like Random Forest, are prone to overfitting, especially with smaller datasets or datasets with redundant features.
- **Model Accuracy:** The highest model accuracy achieved was 63%, which leaves significant room for improvement to make the system more reliable for real-world applications.

6.5 Developing and Using Random Forest

- While the Random Forest Classifier gave the best result in the project (63% accuracy), further testing and fine-tuning the model's hyperparameters can help achieve better performance.
- Future efforts should focus on increasing the depth and number of trees, optimizing features selected for each split, and validating the results on additional datasets.

6.6 Lack of Qualitative Analysis

- The models focused solely on quantitative data and did not incorporate qualitative aspects, such as the physical appearance, odor, or taste of the water, which are also critical indicators of water quality.

6.7 Deployment and Real-World Application

- The predictive system was tested on a dataset but has not been validated in real-world scenarios, which may involve unseen variations in water quality data.

7. Conclusion and Future Work

7.1 Conclusion

The water purity detection project successfully utilized machine learning models to predict the potability of water based on chemical parameters. Key findings include:

- The Random Forest Classifier outperformed other models with an accuracy of 63%, making it the most effective model for this dataset.

- The dataset revealed a higher prevalence of non-potable water samples, emphasizing the importance of water quality monitoring.
- Data preprocessing, including handling missing values and scaling features, played a critical role in improving model performance.

7.2 Future Work

To enhance the project's impact and overcome its limitations, several avenues for future work have been identified:

1. Incorporating Additional Features
 - Include more comprehensive water quality parameters, such as biological contaminants, heavy metals, and temperature, for a holistic analysis.
2. Improving Model Accuracy
 - Experiment with advanced machine learning algorithms, such as Gradient Boosting, XGBoost, or neural networks, to achieve higher accuracy.
 - Address class imbalance using techniques like oversampling (SMOTE) or class-weighted models.
3. Optimizing the Random Forest Classifier
 - Further develop and optimize the Random Forest Classifier by fine-tuning hyperparameters, such as the number of estimators, maximum depth, and minimum samples per split, to improve its predictive accuracy.
4. Real-World Validation
 - Test the predictive system with real-world water samples to assess its practical applicability and reliability.
5. Interactive User Interface
 - Develop an interactive web application where users can input water quality parameters and receive real-time potability predictions.
6. Dynamic Feature Selection

- Implement feature selection techniques to identify the most influential parameters and reduce noise in the dataset.

7. Data Collection Expansion

- Collaborate with organizations to gather larger, more diverse datasets representing various geographic regions and environmental conditions.

8. Integration with IoT Systems

- Integrate the system with IoT devices for real-time monitoring and prediction of water quality.

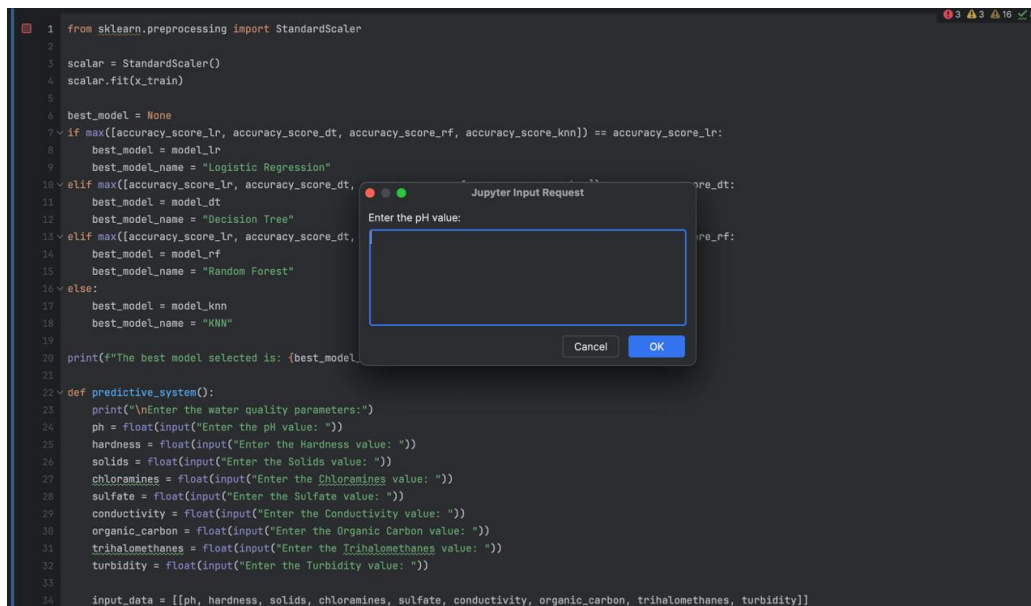
8. Predictive System and Deployment

The final part of the project focuses on developing a predictive system that allows users to input various water quality parameters and obtain real-time predictions about the potability of water. This section highlights the functionality, workflow, and potential applications of the predictive system.

8.1 System Workflow

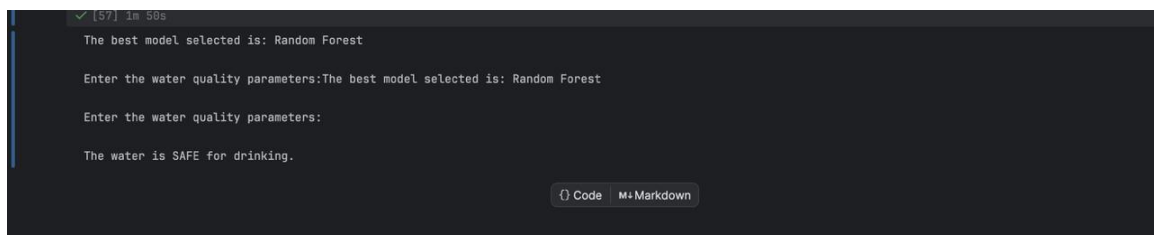
The predictive system follows these steps:

1. User Input:
Users provide inputs for chemical parameters such as pH, Hardness, Sulfate, Turbidity, and others.
2. Data Standardization:
Input data is standardized using the same StandardScaler that was applied during model training to ensure consistent scaling.
3. Prediction:
The Random Forest Classifier, identified as the best-performing model with an accuracy of 63%, predicts whether the water is safe or unsafe for drinking.
4. Output:
The system displays the result, indicating the water's potability status.



```
1 from sklearn.preprocessing import StandardScaler
2
3 scalar = StandardScaler()
4 scalar.fit(x_train)
5
6 best_model = None
7 if max([accuracy_score_lr, accuracy_score_dt, accuracy_score_rf, accuracy_score_knn]) == accuracy_score_lr:
8     best_model = model_lr
9     best_model_name = "Logistic Regression"
10 elif max([accuracy_score_lr, accuracy_score_dt, accuracy_score_rf, accuracy_score_knn]) == accuracy_score_dt:
11     best_model = model_dt
12     best_model_name = "Decision Tree"
13 elif max([accuracy_score_lr, accuracy_score_dt, accuracy_score_rf, accuracy_score_knn]) == accuracy_score_rf:
14     best_model = model_rf
15     best_model_name = "Random Forest"
16 else:
17     best_model = model_knn
18     best_model_name = "KNN"
19
20 print(f"The best model selected is: {best_model_name}")
21
22 def predictive_system():
23     print("\nEnter the water quality parameters:")
24     ph = float(input("Enter the pH value: "))
25     hardness = float(input("Enter the Hardness value: "))
26     solids = float(input("Enter the Solids value: "))
27     chloramines = float(input("Enter the Chloramines value: "))
28     sulfate = float(input("Enter the Sulfate value: "))
29     conductivity = float(input("Enter the Conductivity value: "))
30     organic_carbon = float(input("Enter the Organic Carbon value: "))
31     trihalomethanes = float(input("Enter the Trihalomethanes value: "))
32     turbidity = float(input("Enter the Turbidity value: "))
33
34     input_data = [[ph, hardness, solids, chloramines, sulfate, conductivity, organic_carbon, trihalomethanes, turbidity]]
```

○ Result



```
✓ [57] 1m 58s
The best model selected is: Random Forest

Enter the water quality parameters:The best model selected is: Random Forest

Enter the water quality parameters:

The water is SAFE for drinking.
```

8.2 Benefits of the Predictive System

- **Ease of Use:** The system provides a user-friendly interface for non-experts to determine water quality.
- **Real-Time Predictions:** The system ensures quick and accurate results.
- **Decision Support:** It helps users and authorities make informed decisions regarding water treatment and usage.

8.3 Limitations in Current Deployment

- The current system is based solely on a dataset and has not been validated on real-world water samples.
- Features like user-friendly interfaces, error handling, and additional functionality, such as saving predictions, are yet to be implemented.

8.4 Future Enhancements

To enhance the predictive system:

1. Add support for additional input parameters to improve prediction accuracy.
2. Optimize the Random Forest Classifier and explore alternative algorithms for better performance.
3. Develop a mobile application for wider accessibility.
4. Include data visualization for better interpretation of water quality trends.

9. Limitations

While the water quality analysis and prediction system has shown promising results, it is essential to recognize certain limitations that may impact its overall performance and real-world applicability.

9.1 Data Limitations

- **Sample Size:** The analysis is based on the provided dataset, which may not comprehensively represent all water quality conditions globally.

- **Imbalanced Data:** The dataset shows a significant imbalance, with more samples labeled as non-potable (0) compared to potable (1). This imbalance might bias the model's predictions.

9.2 Missing Data Handling

- **Imputation Methods:** Missing values in parameters like **pH**, **Sulfate**, and **Trihalomethanes** were filled using statistical imputation. While effective, this approach may not fully capture the true variability in these parameters, potentially affecting prediction accuracy.

9.3 Feature Constraints

- **Limited Parameters:** The dataset includes only a specific set of chemical features (e.g., pH, Hardness, etc.), and other critical water quality factors, such as biological contamination or heavy metals, are not considered.
- **Feature Scaling Dependency:** The system relies on standardized data. Poorly scaled input data could result in erroneous predictions.

9.4 Model-Specific Issues

- **Moderate Accuracy:** Despite using advanced machine learning models, the best-performing model (Random Forest Classifier) achieved an accuracy of 63%. This indicates that there is room for improvement in predicting water potability.
- **Overfitting Risks:** Complex models like Random Forest may be prone to overfitting, especially with smaller datasets.

9.5 Deployment Challenges

- **Real-World Data:** The predictive system has not yet been validated with real-world water samples, which might contain additional variability.
- **Resource Dependency:** Deployment of the system in low-resource settings may face challenges due to the requirement for computational power and real-time data collection infrastructure.

10. Conclusion and Future Work

10.1 Conclusions

The water quality analysis and prediction system successfully provides insights into water potability based on chemical parameters. Key outcomes include:

1. The **Random Forest Classifier** demonstrated the best performance with an accuracy of **63%**, making it the preferred model for predictions.
2. Data preprocessing techniques, including missing value imputation and feature scaling, played a critical role in improving model performance.

3. Insights derived from the dataset, such as the correlation between chemical parameters and potability, can assist in making informed decisions about water treatment processes.

10.2 Future Work

To enhance the system's accuracy, usability, and applicability, the following improvements are recommended:

- 1. Integration of Additional Parameters:**

- Include biological and physical water quality factors to make predictions more comprehensive.

- 2. Advanced Models and Techniques:**

- Explore deep learning models and ensemble techniques to improve accuracy and robustness.

- 3. Real-World Validation:**

- Test the system with real-world water quality datasets to ensure practical applicability.

- 4. Dynamic Visualizations:**

- Implement user-friendly dashboards with interactive visualizations for better interpretation of water quality trends.

- 5. Scalability:**

- Develop mobile and web applications to increase accessibility for end-users.