MINI-PROJECT

Water Quality Prediction with Decision Trees

UNIVERSITY OF MONASTIR HIGHER SCHOOL OF COMPUTER SCIENCE AND MATHEMATICS

SOFTWARE ENGINEERING – ING1 2023-2024

MARYEM CHAKROUN & REFKA MECHRI

1.Feature Engineering

we first loaded our data into a dataframe, to get a quick overview of the structure and contents of our dataset , we used df.head()

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	0
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180013	56.329076	4.500656	0
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.868637	66.420093	3.055934	0
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.341674	4.628771	0
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	0

Exploratory Data Analysis

we used df.shape to know the dimensions of our dataframe

```
(3276, 10)
```

the number of missing values

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

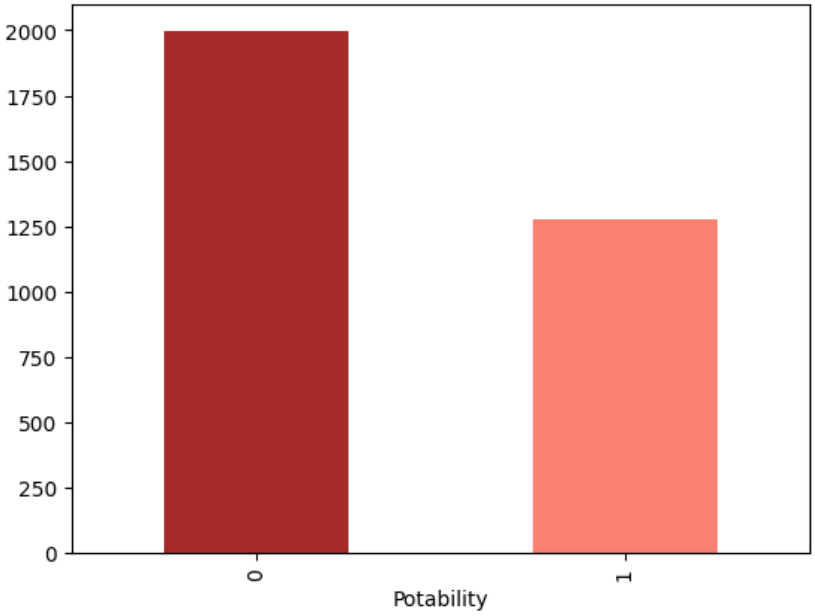
we notice that sulfate and ph are missing a of values

Upon using df.info(), we noticed that all our features are numerical. Therefore, we won't need variable encoding.

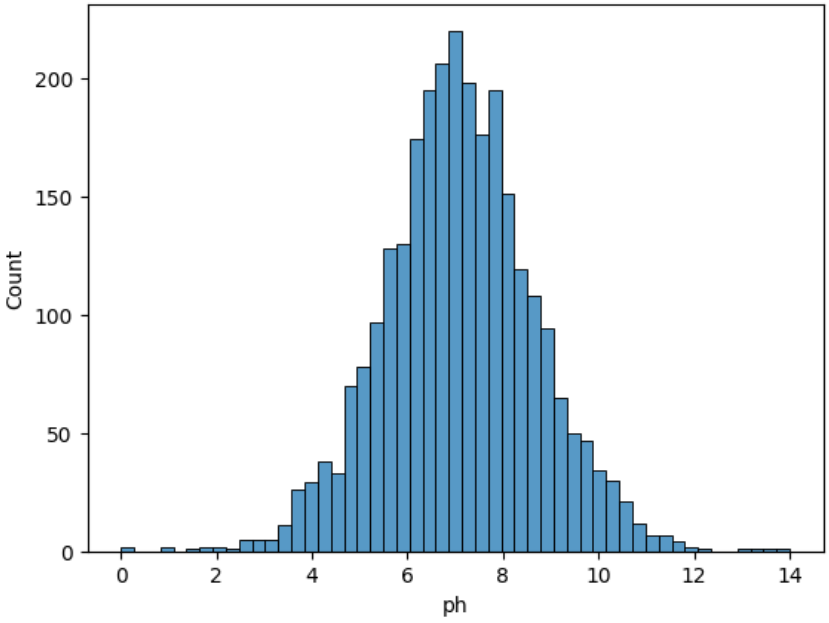
```
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ph               2785 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3114 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

Count the frequency of each unique value in the 'Potability' column

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

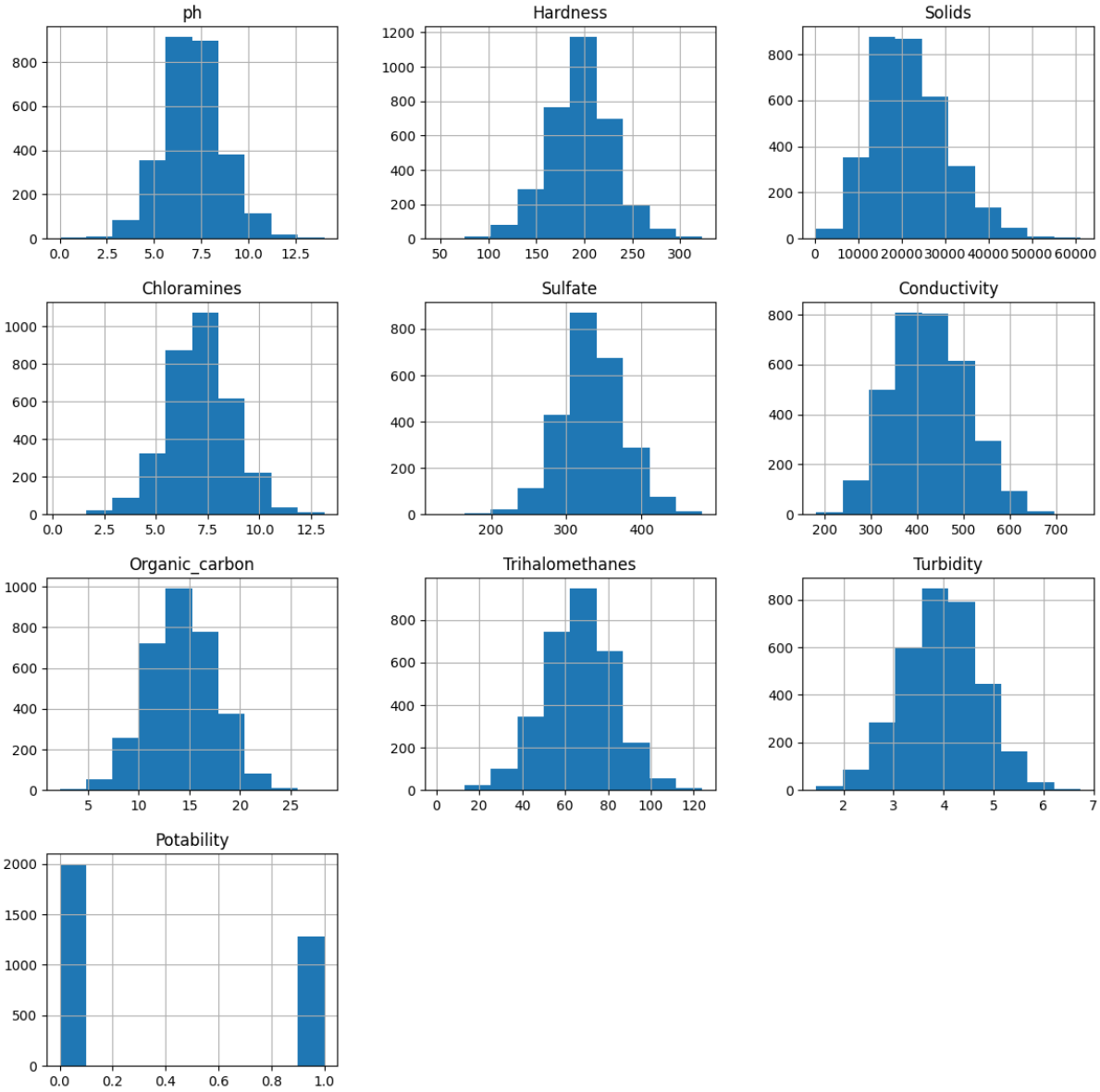


histogram of the 'ph' column using Seaborn



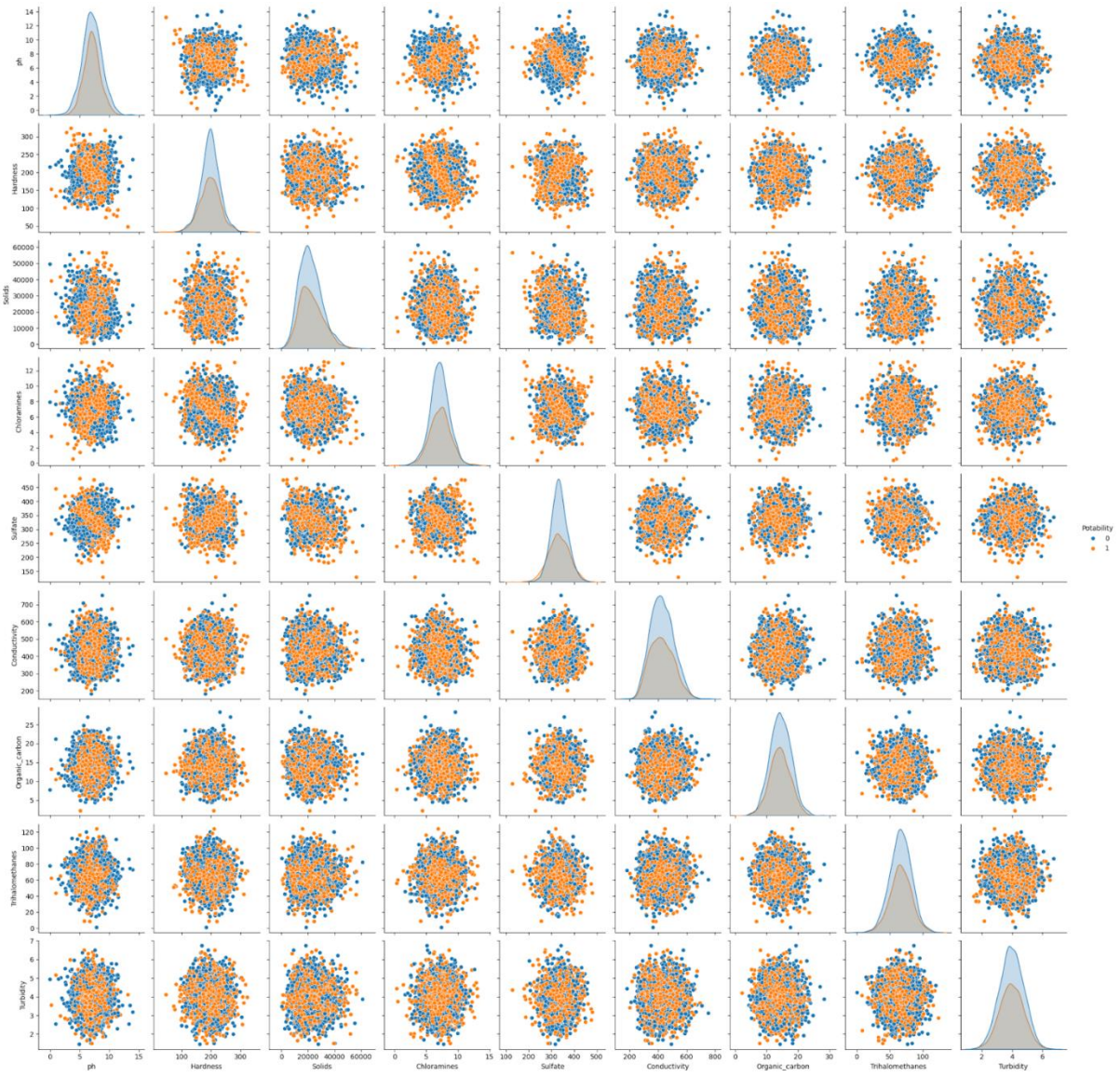
We noticed that the majority of pH values fall within the range of 6 to 8, which is logical considering that the average pH of water is expected to be around 7.

histograms for each numerical column in the DataFrame



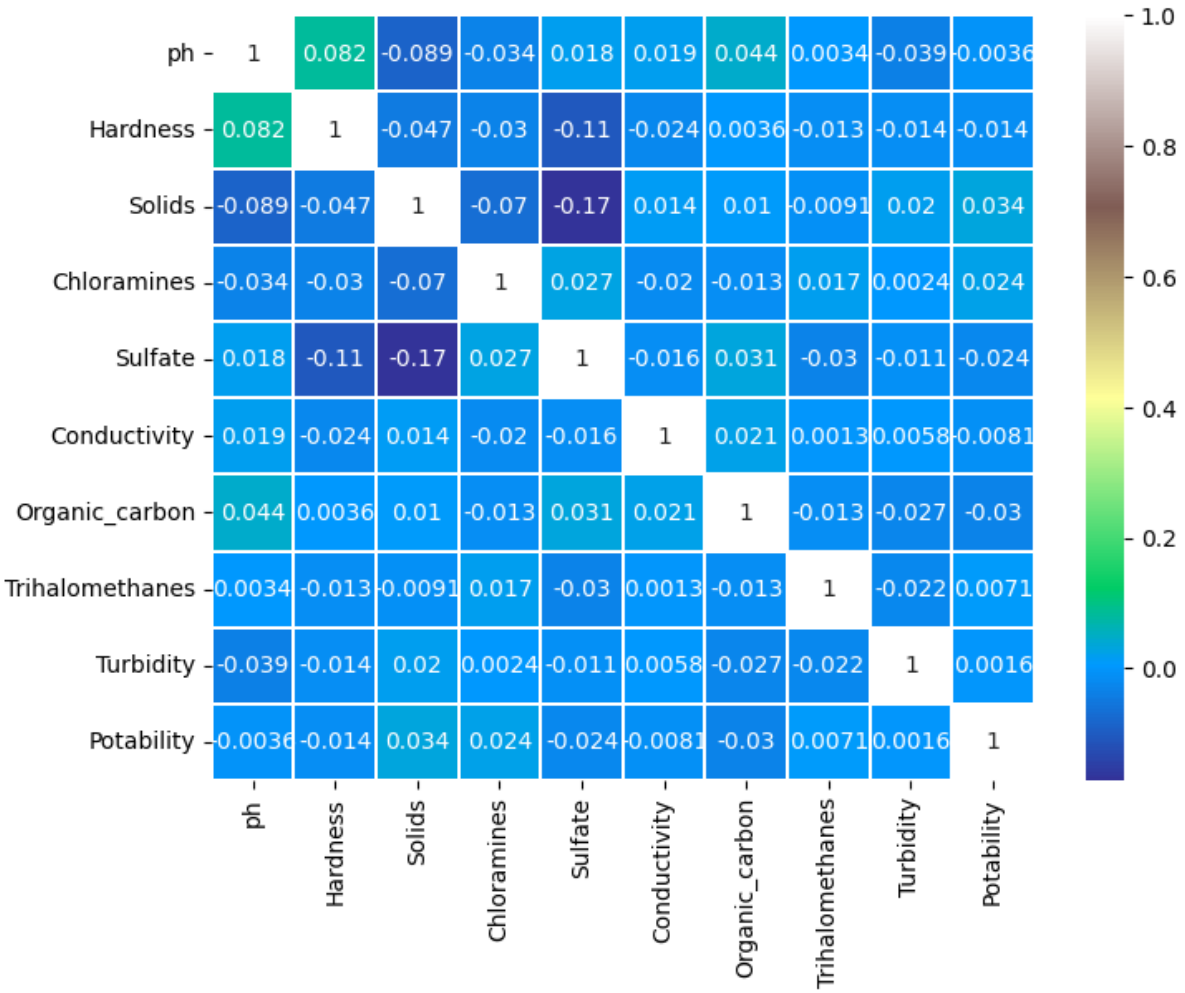
Most of the histograms exhibit a bell-shaped distribution without any skewness. However, the potability histogram stands out as a two-category histogram, representing the presence or absence of potability in water samples.

pairwise scatterplots colored by 'Potability'



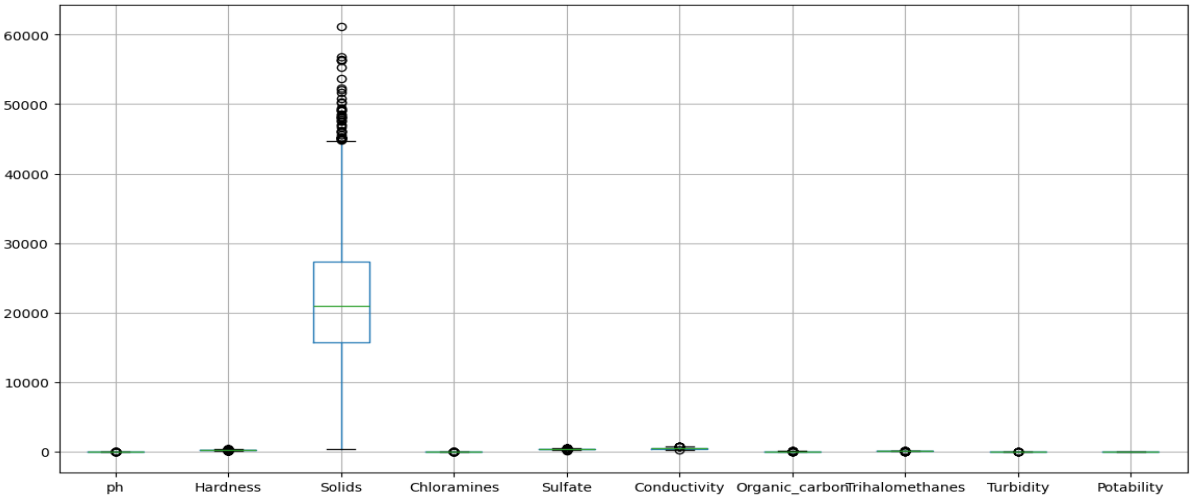
The pairwise scatterplots colored by 'Potability' offer a visual and intuitive way to explore the relationship between water quality variables and potability, enabling better understanding and decision-making regarding water safety. Additionally, The circular shape formed by the points suggests that there may be a non-linear relationship between the variables

the correlation heatmap



the correlation coefficients in the heatmap are low, indicating a weak linear relationship, it's possible that the variables have a more complex, non-linear association that is not captured by the correlation coefficient.

the boxplot of the feature "solids"



```
Click here to ask Blackbox to help you code faster
df['Solids'].describe()
```

```
19]
```

```
count     3276.000000
mean     22014.092526
std       8768.570828
min        320.942611
25%      15666.690297
50%      20927.833607
75%      27332.762127
max      61227.196008
Name: Solids, dtype: float64
```

we notice that a huge gap exists between the feature "solids" and the other features

Cleaning and data preparation:

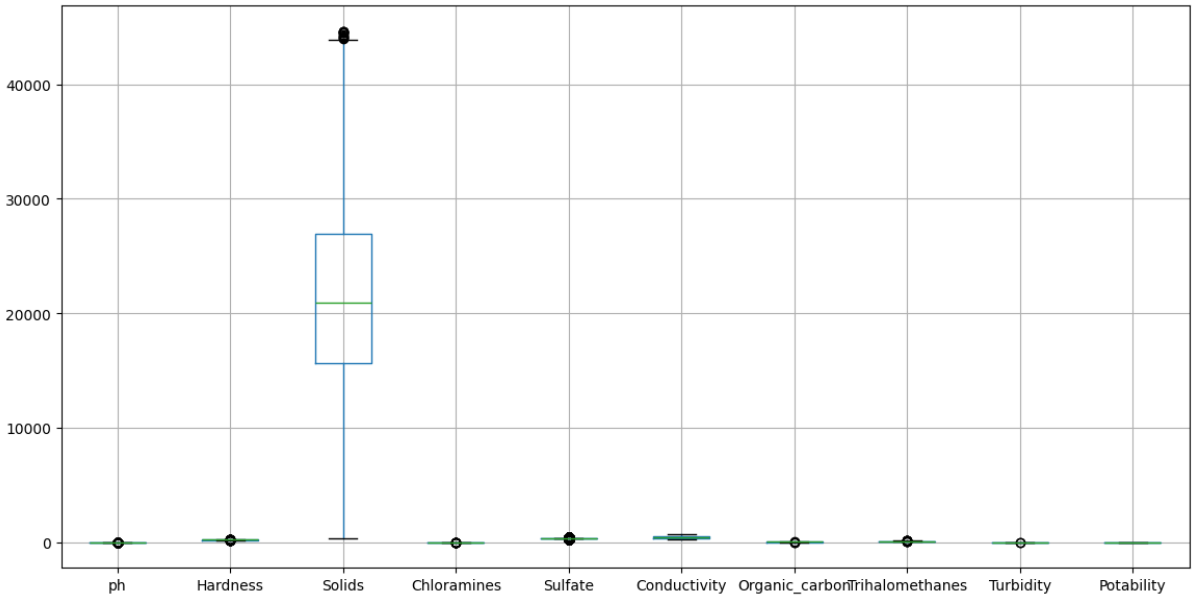
handling missing values:

Since there are a lot of missing values, we decided to replace the missing values with the mean of each feature.

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

handling outliers :

we identified outliers in each column of the DataFrame df,and replaced them with the mean of the respective column.



Normalization (Z-score normalization):

We chose Z-score normalization because it is less sensitive to outliers compared to Min-Max Scaling.

```
      Chloramines  Conductivity  Hardness  Organic_carbon    Solids   Sulfate  \
0        0.122299      1.747388  0.284779       -1.223469 -0.103048  1.358913
1       -0.337732      2.107127 -2.282372        0.279961 -0.368679 -0.001476
2        1.489093     -0.086792  0.942858        0.808837 -0.211423 -0.001476
3        0.647468     -0.783437  0.607357        1.299899  0.047769  0.903488
4       -0.399059     -0.341022 -0.524441       -0.854364 -0.448699 -0.927182
...           ...           ...       ...             ...       ...       ...
3271     0.029892      1.270478 -0.096504       -0.122687  0.047238  1.023409
3272     0.648872     -0.416065 -0.100876        1.759269 -0.528487 -0.001476
3273     0.156905      0.082380 -0.706051       -1.016981  1.416588 -0.001476
3274    -0.567337     -0.284723  1.159459       -0.976303 -1.185532 -0.001476
3275     0.266953     -1.234192 -0.048181        0.580744 -0.519346 -0.001476

      Trihalomethanes  Turbidity        ph  Potability
0            1.404960  -1.322438  0.005345           0
1           -0.694107   0.702485  0.005345           0
2           -0.003291  -1.200222  0.845312           0
3            2.318929   0.871213  1.024833           0
4           -2.359776   0.141993  1.664468           0
...               ...        ...       ...         ...
3271         0.015029   0.617097 -1.984760           1
3272        -0.004920  -1.539602  0.605885           1
3273         0.231200  -0.880266  1.934430           1
3274         0.754416   0.976425 -1.606435           1
3275         0.837267  -2.183741  0.660172           1

[3276 rows x 10 columns]
```

2.Model Engineering

Data partitioning:

To prepare our data for training, we split our dataset into training and test sets using the train_test_split function in the scikit-learn library.

Training set (X_train, Y_train):

X_train: Set of features used to train our model.

Y_train: Set of labels corresponding to the training set, used to train the model to make predictions.

Test set (X_test, Y_test) :

X_test: Set of features used to evaluate the performance of our model on unseen data.

Y_test: Set of labels corresponding to the test set, used to evaluate the model's predictions.

We used the following parameters for splitting the data:

test_size: 0.2, which means that 20% of the data is reserved for the test set.

random_state: 101, used to guarantee the reproducibility of the results.

shuffle: True, to shuffle the data before splitting, which is particularly useful if the data is sorted or grouped.

```
Y_train.value_counts()
```

```
Potability
0    1596
1    1024
Name: count, dtype: int64
```

```
Y_test.value_counts()
```

```
Potability
0    402
1    254
Name: count, dtype: int64
```

Modelling with a Decision Tree:

Learning on Training Data

For this step, we used a decision tree classification model using the scikit-learn library. Here are the parameters we used to implement the model:

Node separation criterion: 'entropy', which uses entropy as a criterion to measure the quality of splits.

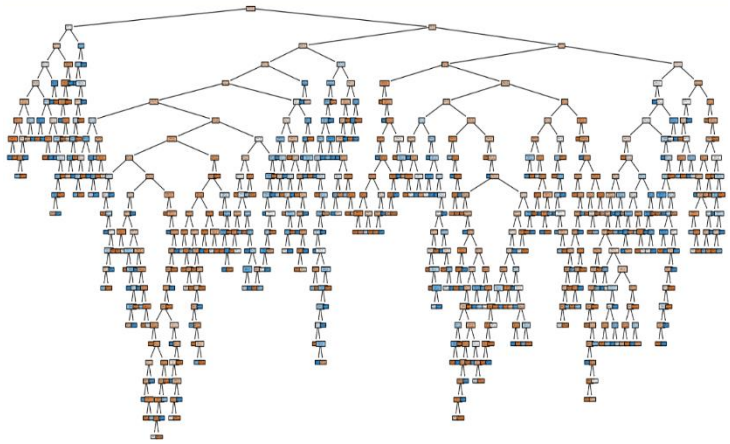
Minimum number of samples required to split a node: 10, specified with min_samples_split=10.

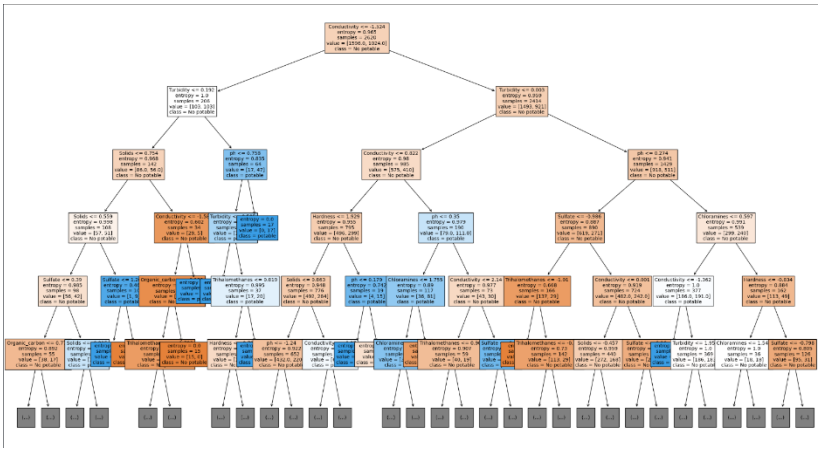
Split selection strategy: 'best', which indicates that the best strategy will be used to select the split at each node.

Random seed for reproducibility of results: 101, specified with random_state=101.

After implementing the model with these parameters, we trained it on the training set (X_train, Y_train) using the fit method.

Following this training, we obtained a learned decision tree. To visualise this decision tree, we used graphical and textual methods.





```
|--- Conductivity <= -1.52
|    |--- Turbidity <= 0.19
|    |    |--- Solids <= 0.75
|    |    |    |--- Solids <= 0.56
|    |    |    |    |--- Sulfate <= 0.39
|    |    |    |    |    |--- Organic_carbon <= 0.78
|    |    |    |    |    |    |--- Chloramines <= -0.12
|    |    |    |    |    |    |    |--- Organic_carbon <= -2.40
|    |    |    |    |    |    |    |    |--- class: 1
|    |    |    |    |    |    |    |--- Organic_carbon >  -2.40
|    |    |    |    |    |    |    |    |--- class: 0
|    |    |    |    |    |    |--- Chloramines >  -0.12
|    |    |    |    |    |    |    |--- Turbidity <= -0.04
|    |    |    |    |    |    |    |    |--- Turbidity <= -0.99
|    |    |    |    |    |    |    |    |    |--- class: 1
|    |    |    |    |    |    |    |    |--- Turbidity >  -0.99
|    |    |    |    |    |    |    |    |    |--- class: 0
|    |    |    |    |    |    |    |--- Turbidity >  -0.04
|    |    |    |    |    |    |    |    |--- class: 1
|    |    |    |    |    |--- Organic_carbon >  0.78
|    |    |    |    |    |    |--- Hardness <= 0.41
|    |    |    |    |    |    |    |--- class: 0
|    |    |    |    |    |    |--- Hardness >  0.41
|    |    |    |    |    |    |    |--- class: 1
|    |    |    |    |--- Sulfate >  0.39
---
|    |    |    |    |    |    |    |    |    |--- class: 0
|    |    |    |    |    |    |--- Organic_carbon >  1.55
|    |    |    |    |    |    |    |--- class: 0
```

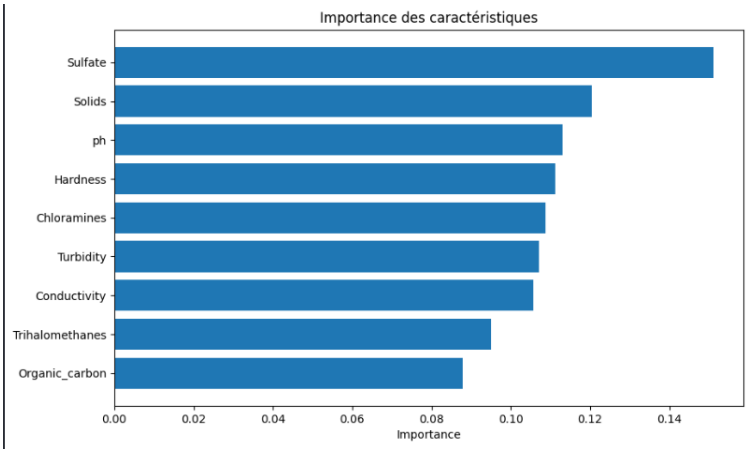
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings

⇒These visualizations will allow us to understand the structure of the learned decision tree and the decision rules that have been created by the model.

Feature Importance:

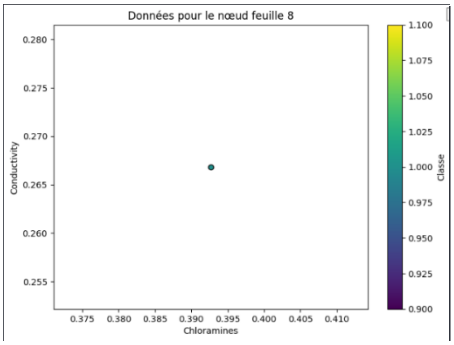
The importance of features in a decision tree allows us to understand which features have the most influence on the model's decision making. In our case, we used the decision tree classification model to evaluate feature importance for the classification task.

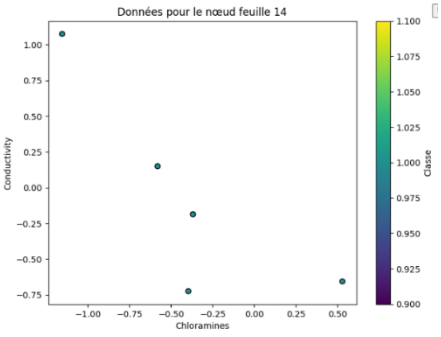
After training our model, we extracted feature importances using the feature_importances_ attribute of the decision tree model

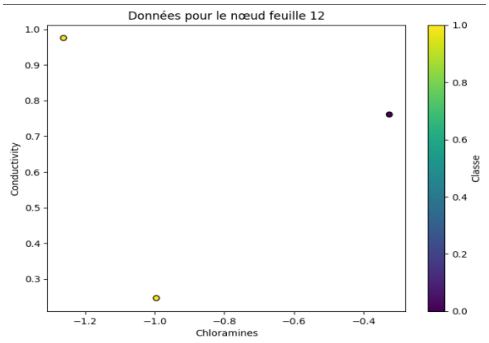


The importance of a feature in a decision tree is determined by the mean reduction in the decision criterion (such as entropy or the Gini criterion) that it provides across the tree. A feature that is used more frequently to divide the data is considered to be more important. The sum total of the importance of the features is 1, representing 100% of the total importance. These importances help us to identify the features with the highest impact on water quality, thus guiding interpretation and decision-making based on the model.

Data Separation for 3 Leaf Nodes







Model evaluation

After training our decision tree model, it is crucial to evaluate its performance on unseen datasets to understand how well it is able to generalize to new data. We evaluated the performance of our model on the training set and the test set using the accuracy metric.

```
Accuracy on training set: 0.922
Accuracy on test set: 0.578
```

⇒Training set accuracy:

The accuracy of the training set is 0.922. This measure shows us the proportion of correct predictions made by the model on the training set. A high training set accuracy may indicate that the model is learning the training data very efficiently.

⇒Test set accuracy

The accuracy on the test set is 0.578. This measure shows us the proportion of correct predictions made by the model on the test set . A lower accuracy on the test set compared to the training set may indicate an over-fitting of the model to the training data, which may affect its ability to generalize to new data.

Pruning the Decision Tree

Pre-pruning:

Pre-pruning is a technique used to optimize the structure of the decision tree by limiting its size and avoiding overfitting to the training data. To do this, we used grid search (GridSearchCV) to find the best hyperparameters for our decision tree model.

Parameters used for pr-pruning:

Node separation criterion: 'entropy' was chosen as the node separation criterion because it gave better results than 'gini'.

Maximum tree depth (max_depth): A maximum depth of 7 was chosen to avoid overfitting while allowing the tree to capture important information in the data.

Minimum number of samples required to split a node (min_samples_split): A minimum of 2 samples is required to avoid splitting on samples that are too small.

Minimum number of samples required to be at a leaf node (min_samples_leaf): A minimum of 5 samples is required to avoid leaves with very few samples.
Maximum number of leaves (max_leaf_nodes): A maximum of 15 leaves has been chosen to limit the complexity of the tree.
Splitting strategy: 'random' was chosen as the splitting strategy to introduce variability when choosing splits.

```
Depth of the original tree: 23
Number of leaves in the original tree: 368
Depth of the pruned tree: 7
Number of leaves in the pruned tree: 15
```

⇒The original tree was much larger, with a high number of leaves and a greater depth.
But after pr-pruning, the pruned tree is shorter, with fewer leaves and a limited depth, making it less susceptible to over-fitting.

without pruning

```
Accuracy on training set: 0.922
Accuracy on test set: 0.578
```

=====>

with pruning

```
Accuracy on training set: 0.618
Accuracy on test set: 0.630
```

⇒Pre-pruning has improved accuracy over the test set while reducing the risk of overfitting. It therefore provides better generalization to new data, which is essential for model reliability in real-world scenarios.

Post-pruning:

Post-pruning is a technique that consists of reducing the size of an implemented decision tree by removing some of its leaves in order to improve its generalization and avoid over-fitting.
Post-pruning method chosen:
We chose the error reduction method for posterior pruning. This method consists of evaluating the accuracy of the tree on a validation set after deleting each node in the tree. If the accuracy of the pruned tree on the validation set is equal to or greater than that of the original tree, the node is permanently removed from the tree.
The need for a validation set for post pruning
The use of a validation set is essential for post pruning as it allows the accuracy of the pruned tree to be evaluated on data not seen during training. This ensures that posterior pruning does not focus only on fitting the training data, but also aims to improve the model's ability to generalize to new data.

without pruning

```
Accuracy of the original tree on validation set: 0.744
```

```
Accuracy on training set: 0.922
Accuracy on test set: 0.578
```

====>

with pruning

```
Accuracy of the pruned tree on validation set: 0.778
Accuracy of the pruned tree on test set: 0.706
Accuracy of the pruned tree on training set: 0.802
```

⇒ Post pruning succeeded in improving the ability of our decision tree model to generalize to new data while reducing overfitting. This confirms the importance of this step in the process of building and optimising decision tree models to ensure reliable and robust performance in real scenarios.

Comparison between Pre-pruning and Post-pruning:

Pre-pruning post-pruning

```
Accuracy on training set: 0.618
Accuracy on test set: 0.630
```

====>

```
Accuracy of the pruned tree on validation set: 0.778
Accuracy of the pruned tree on test set: 0.706
```

⇒The performance comparison shows that a post-pruning is more efficient than a pre-pruning for our dataset. Although pre-pruning performs slightly worse on the training set, post-pruning offers a better balance between learning and generalization, making it more reliable for accurate predictions on new data.

Recommendations and conclusions

Choosing the Best Decision Tree

After an analysis of the performance of pruned and unpruned decision trees, we recommend the use of the pruned decision tree as the best model for data classification. This model offers an optimal compromise between accuracy on training data and the ability to generalize to new data

Combination of Pre and Post Pruning

The relevance of combining these two types of pruning lies in their ability to balance the accuracy of the training data and the generalizability of the model. Preliminary pruning allows the growth of the tree to be controlled from the outset, while post-pruning adjusts the tree to optimise its performance on a specific validation set.

Method used to combine the two types of pruning:

To combine preliminary and posterior pruning, we first applied preliminary pruning by defining specific parameters to control tree growth during initial training. Then, we used posterior pruning to adjust the pruned tree by removing non-essential nodes that do not improve model performance on a specific validation set.

Evaluation of the combined model

After combining preliminary and posterior pruning, we evaluated the performance of the pruned decision tree model on the test set and the training set to measure its accuracy.

```
Accuracy of the pruned tree on test set: 0.619
Accuracy of the pruned tree on training set: 0.620
```

⇒The accuracy of 0.619 on the test set indicates a moderate ability to generalize the pruned decision tree model. This performance is relatively consistent with the accuracy of 0.620 on the training set, suggesting good balance and an absence of over-fitting.