Introduction to Machine Learning: Supervised Learning

Final Project

Machine Learning for Water Drinkability

Morgan I. MacKay

June 24th, 2025

Part 1: Project Introduction

This project explores the application of machine learning to assess drinking water potability as a classification problem. With approximately 3,400 observations and nine distinct water quality parameters in our dataset, the primary objective is to develop a robust predictive model capable of classifying water samples as either potable or non-potable. Leveraging decision trees, this analysis aims to create a clear and effective system for determining water safety based on its measured attributes. The goal is to create the most accurate prediction model possible, using a variety of machine learning models and approaches.

Access to clean and safe drinking water is fundamental to public health and global well-being. Contaminated water sources can lead to a myriad of health issues, ranging from gastrointestinal illnesses to severe chronic diseases, impacting vulnerable populations disproportionately. This project, utilizing a synthetic dataset, serves as an exercise in developing and refining the machine learning models necessary for such critical assessments. In real-world applications, a successfully developed predictive model for water potability could then be deployed to analyze real-time water quality data from sensors or lab tests. This would enable rapid assessment by water treatment plants, public health agencies, and environmental monitoring bodies, facilitating interventions before contamination becomes widespread. Furthermore, it could optimize monitoring efforts, helping allocate resources more efficiently to areas or parameters most indicative of safety risks, ultimately contributing to better public health outcomes.

Part 2: Data

The data for this project was obtained from a Kaggle repository that was opened up to the public domain. To find the dataset online, visit the following link: https://www.kaggle.com/datasets/adityakadiwal/water-potability/data(2021)[1]. The dataset was created and uploaded by Aditya Kadiwal. The dataset is available in CSV format, and takes up 525.19 kB of storage. The project's data is tabulated. There are 9 different input variables, all having to do with different measures of water quality (Ph, Hardness, Chloramines, Sulfates, etc.). There is one predictor variable, "Potability" which has a binary datatype. There are 3,275 observations in the dataset, with each observation representing a different body of water. The data is presented in a single table form, from a single source.

There are a couple key points to take into consideration at this initial stage of this project. The input variables for Ph, Sulfate and Trihalomethanes contain missing observations. Due to the synthetic nature of this dataset and the relative size of our observation pool, the observations with these missing values will be dropped during the cleaning stage.

Variable Name	Data Type
Ph (1-14)	Floating-Point
Hardness (mg/L)	Floating-Point
Solids (ppm)	Floating-Point
Chloramines (ppm)	Floating-Point
Sulfate (mg/L)	Floating-Point
Conductivity (μS/cm)	Floating-Point
Organic Carbon (ppm)	Floating-Point
Trihalomethanes (μg/L)	Floating-Point
Turbidity (NTU)	Floating-Point
Potability (Yes or No)	Boolean

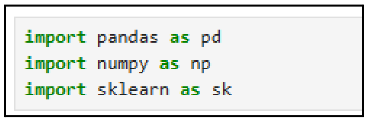
Part 3: Data Cleaning

For this project, the data cleaning stage was multi-faceted and essential for ensuring the quality and reliability of our analysis. I began by importing the raw CSV file into a JupyterLab environment, utilizing a Python kernel (Pyodide). To gain an initial understanding of the dataset's structure and content, I employed basic descriptive commands such as df.head() and df.info(), which provided an overview of data types, column names, and initial presence of non-null values. Following this exploratory phase, I addressed common data quality issues. This involved identifying and handling missing values, detecting and removing duplicate rows, correcting data type inconsistencies, and standardizing data formats where necessary to ensure uniformity. These are all issues that make it difficult or impossible to train machine learning models, so they have to be addressed early.

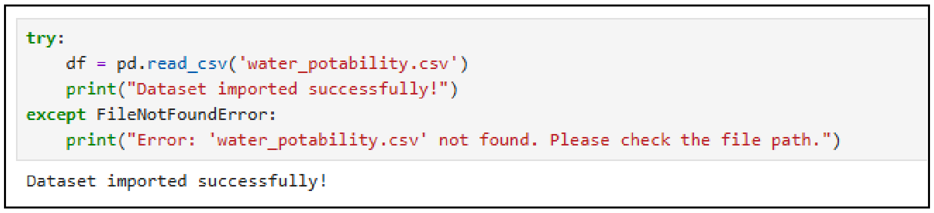
The subsequent sections will provide a more in-depth analysis of each cleaning step, clarifying how each operation was performed and why it was necessary to improve data quality for subsequent analysis. For the complete Jupyter Notebook, please refer to Appendix 2.

Data Cleaning: Importing

The import phase of the project was relatively straightforward. The following code chunks show how the CSV was loaded into JupyterLab.



```
import pandas as pd
import numpy as np
import sklearn as sk
```

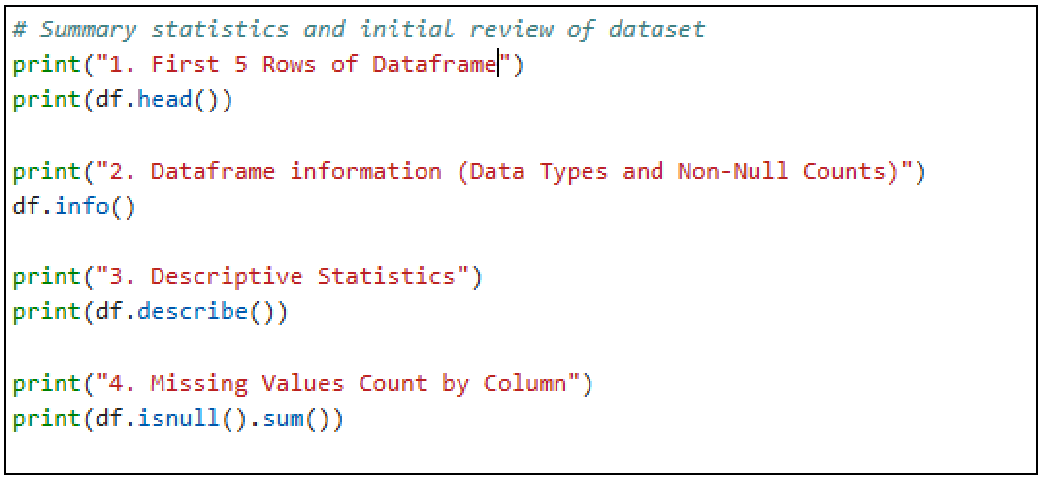


```
try:
    df = pd.read_csv('water_potability.csv')
    print("Dataset imported successfully!")
except FileNotFoundError:
    print("Error: 'water_potability.csv' not found. Please check the file path.")
```

```
Dataset imported successfully!
```

Data Cleaning: Data Inspection & Summary

Upon execution of these commands, several notable observations emerged. As indicated by Kaggle's data overview, the dataset contained numerous missing (NaN) values that would require handling. Additionally, the data exhibited excessive precision, specifically five decimal points, which was more than necessary for our analysis. The descriptive commands confirmed a total of 3275 observations within the dataset. Furthermore, the "Potability" output column was identified as an int64 datatype, necessitating a conversion to a boolean type during the cleaning phase. Overall, these initial results provided a comprehensive understanding of the summary statistics and clearly highlighted which columns contained missing values. The following code chunk shows the descriptive commands that I used to get a better understanding of the dataset. Please see appendix for output[3].

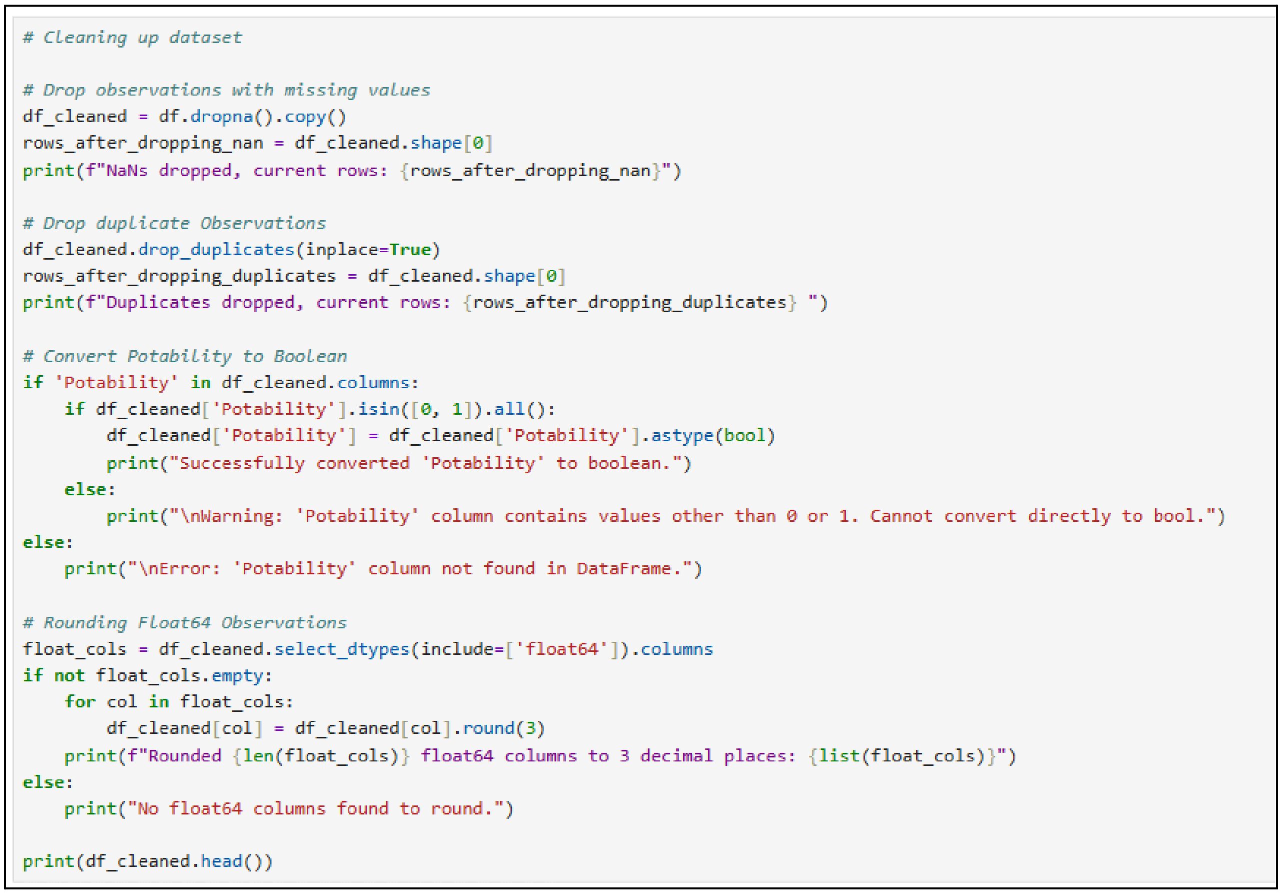


```
# Summary statistics and initial review of dataset
print("1. First 5 Rows of Dataframe")
print(df.head())

print("2. Dataframe information (Data Types and Non-Null Counts)")
df.info()

print("3. Descriptive Statistics")
print(df.describe())

print("4. Missing Values Count by Column")
print(df.isnull().sum())
```

Data Cleaning: Missing Values, Duplicate Rows, Data Type Standardization

Given the synthetic nature of the data and the large number of observations, I opted to drop rows containing missing values and duplicate entries. Missing, or "NaN" values can cause models to fail during the fitting stage. After these operations, the dataset contained 2011 unique observations. Subsequently, the "Potability" column was converted to a boolean datatype. Finally, the values in the nine float64 columns were rounded to three decimal places to enhance readability. Based on other summary metrics and how the code was performing, I felt comfortable moving forward with the data after these cleaning steps were taken.

The following code chunk presents the data cleaning operations. (For code output, please see Appendix 4.)



```
# Cleaning up dataset

# Drop observations with missing values
df_cleaned = df.dropna().copy()
rows_after_dropping_nan = df_cleaned.shape[0]
print(f"NaNs dropped, current rows: {rows_after_dropping_nan}")

# Drop duplicate Observations
df_cleaned.drop_duplicates(inplace=True)
rows_after_dropping_duplicates = df_cleaned.shape[0]
print(f"Duplicates dropped, current rows: {rows_after_dropping_duplicates} ")

# Convert Potability to Boolean
if 'Potability' in df_cleaned.columns:
    if df_cleaned['Potability'].isin([0, 1]).all():
        df_cleaned['Potability'] = df_cleaned['Potability'].astype(bool)
        print("Successfully converted 'Potability' to boolean.")
    else:
        print("\nWarning: 'Potability' column contains values other than 0 or 1. Cannot convert directly to bool.")
else:
    print("\nError: 'Potability' column not found in DataFrame.")

# Rounding Float64 Observations
float_cols = df_cleaned.select_dtypes(include=['float64']).columns
if not float_cols.empty:
    for col in float_cols:
        df_cleaned[col] = df_cleaned[col].round(3)
    print(f"Rounded {len(float_cols)} float64 columns to 3 decimal places: {list(float_cols)}")
else:
    print("No float64 columns found to round.")

print(df_cleaned.head())
```

Part 4: Exploratory Data Analysis

The code for this section was substantially longer than for other sections, please see Appendix 5 for the full code.

Exploratory Data Analysis (EDA) is crucial because it helps you gain a deep understanding of your dataset's characteristics, including patterns, anomalies, and relationships between variables. This foundational understanding informs appropriate analytical approaches and ultimately leads to more robust and accurate insights.

KDE (Kernel Density Estimation) plots visualize the probability density function of continuous data, providing a smoothed representation of the underlying distribution. When I generated KDE plots for the water potability data, a couple of key points stood out. Each of the nine input variables appeared to be roughly normally distributed, which is highly unusual for real-world data. Furthermore, the data for each input variable consistently showed its highest density around the midpoint of the acceptable value range. Both of these observations strongly suggest the data was synthetically generated, a fact already known. Another aspect indicating synthetic data (and a potential issue if this were real-world data) is that the distributions for drinkable and non-drinkable water showed almost no discernible difference across the variables. While this lacks logical sense in a real-world context, it is not an issue for synthetic data designed for training purposes. For KDE visualizations, please see Appendix 6.

Heatmaps utilize a gradient of colors to represent the magnitude of a phenomenon, making them excellent tools for visualizing correlation matrices or large datasets to quickly identify patterns, relationships, and clusters between variables. In the context of this project, the heatmap generated for our dataset revealed notably weak linear correlation among the variables. This observation could suggest two possibilities: either the data exhibits non-linear relationships that are not captured by a standard correlation analysis, or its synthetic nature contributes to a higher degree of randomness rather than strong linear associations. For the detailed heatmap visualization, please refer to Appendix 7.

Boxplots (also known as box-and-whisker plots) graphically display the distribution of numerical data by illustrating their quartiles, median, and potential outliers. For this project, the most notable observation derived from the boxplots was the presence of numerous outliers distributed across various variables. Apart from these outliers, the boxplots generally indicated that the data within each variable was roughly evenly distributed. As discussed previously, the existence of these outliers and the consistent distribution are not problematic for a synthetic training dataset as their primary purpose is to facilitate model learning. However, in a real-world data analysis scenario, such characteristics would warrant further investigation and potentially require specific handling during the data cleaning and preprocessing phases to avoid skewing results or misrepresenting phenomena. For the detailed boxplot visualizations, please refer to Appendix 8.

Part 5: Models

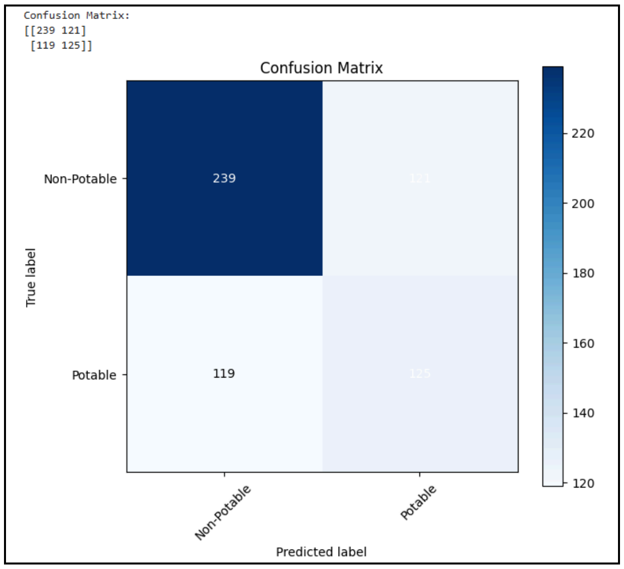
Given that the dataset aims to predict a boolean outcome, I selected the decision tree model as the foundational machine learning approach. Decision trees are particularly well-suited for classification problems and are adept at uncovering the non-linear relationships that our previous exploratory analysis indicated are present within this dataset. The base decision tree will be the first of the following sections.

Using a decision tree as the base model, several improvements can be implemented. The first step involves adjusting its hyperparameters. Additionally, ensemble methods such as Random Forests and Gradient Boosting can be applied to enhance accuracy. Finally, a Support Vector Machine (SVM) approach will be utilized to determine if it yields comparable or superior results. Each approach will be broken down in the following sections.

Models: Decision Tree & Hyperparameter Tuning

The initial decision tree was built using the DecisionTreeClassifier function within the sk.learn.tree package. To understand the results more, a classification report and confusion matrix were generated as well. The model's accuracy was .6026, or 60.26%. For the detailed code implementation, please refer to Appendix 9.

The results had a few interesting points. The main one was the accuracy rate. We can expect this number to rise as we try improved models, but a baseline of roughly 60% would be considered unacceptably low in the real world. For something as important as if water is safe to drink or not, we would want a useful model to have upwards of 90% accuracy. The F1 score of the model was higher for non-potable water (.67) than it was for potable water (.51). This means the model was better at predicting unsafe drinking water than it was for predicting safe drinking water. Finally, the support was 604, meaning we had 604 test cases.



```
Confusion Matrix:
[[239 121]
 [119 125]]
```

The initial decision tree was then improved upon by adjusting the model's hyperparameters. This was done automatically through the GridSearchCV function in the sklearn.model_selection package. Here are the parameters tested:

```
param_grid = {
    'max_depth': [None, 5, 10, 15, 20],
    'min_samples_split': [2, 10, 20],
    'min_samples_leaf': [1, 5, 10],
    'criterion': ['gini', 'entropy']
}
```

Tuning the tree didn't provide better results. The highest accuracy recorded on the training data was 65.46%, but the highest testing data accuracy was 60.26%. Reviewing the F1 scores indicates that the model got better at identifying non-potable water, but at the cost of identifying potable water. This is a trade off that would have to be weighed in a real life scenario. According to GridSearch, the best criterion for the tree were the following:

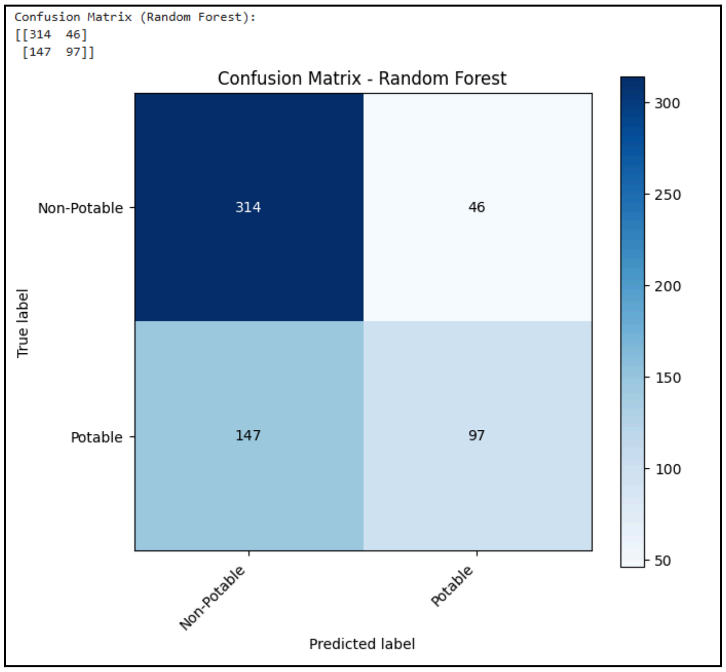
- Criterion: Gini
- Max Depth: 5
- Min. Samples Leaf: 10
- Min. Samples Split: 2

To see the code for the GridSearchCV function, see Appendix 10. To see the full results from the GridSearchCV function, please see Appendix 11.

Models: Random Tree

A Random Forest ML model is an ensemble learning method that builds and combines predictions from multiple individual decision trees, trained on different subsets of the data and features, which collectively reduces overfitting and improves the overall predictive accuracy and robustness compared to a single decision tree. For this dataset, the random forest was generated using the RandomForestClassifier function from the sklearn.ensemble package.

The random forest model offered an increase in accuracy, this time up to 68.05%. Evaluating the F1 scores indicates that the model is still lopsided, and correctly identifies non-potable water more often (.76) than it correctly identifies potable water(.50). The support, or number of test cases was the same as in the previous step. Overall, this model performed significantly better than the "tuned" decision tree from the previous step. See Appendix 12 for full random tree outputs.

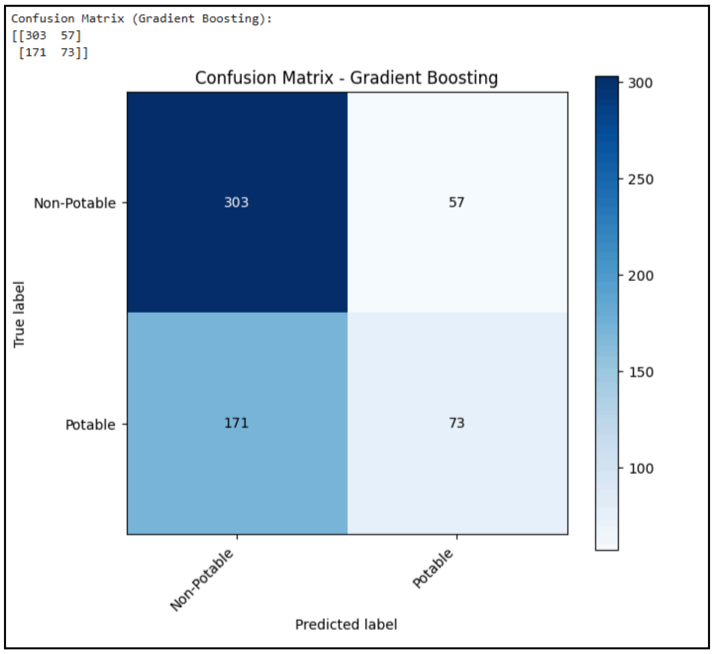


```
Confusion Matrix (Random Forest):
[[314  46]
 [147  97]]
```

Models: Boosting

Boosting is a powerful ensemble learning technique that sequentially combines multiple "weak" or "base" learners (often simple models like shallow decision trees) to create a single, more robust "strong" learner. For this project, the boosting model was trained using the GradientBoostingClassifier function from the sklearn.ensemble package.

The Gradient Boosting model achieved an overall accuracy of 62.25%. A more detailed examination of its performance revealed F1 scores of 0.73 for the non-potable class and 0.39 for the potable class. These metrics consistently indicate that the boosting model, similar to the previously evaluated models, demonstrated a stronger predictive capability for non-potable water. See appendix 13 for full boosting tree outputs.

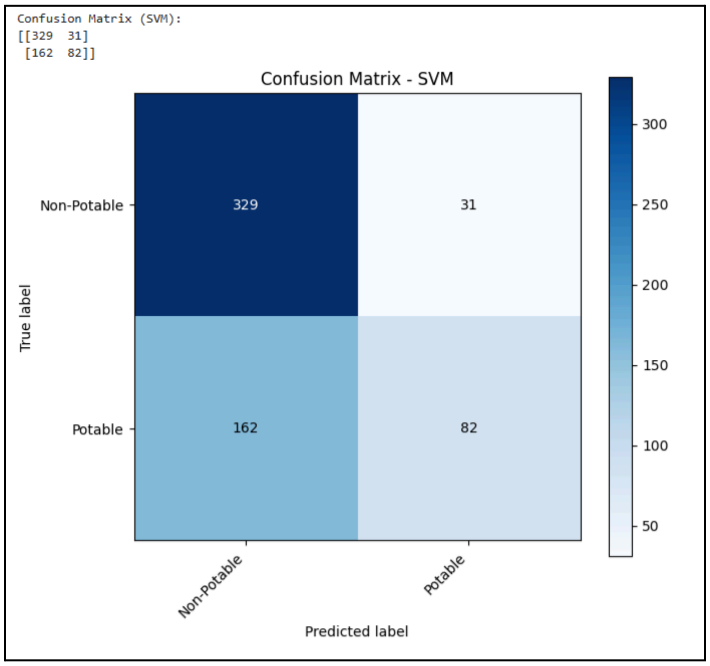


```
Confusion Matrix (Gradient Boosting):
[[303  57]
 [171  73]]
```

Models: SVM

Support Vector Machines (SVMs) are powerful supervised learning models used for both classification and regression tasks, though they are most commonly applied to classification. The core idea behind SVMs is to find the optimal hyperplane that best separates different classes in the feature space. To implement an SVM for this project the StandardScaler and SVC functions were imported from sklearn.preprocessing and sklearn.svm.

The SVM model achieved the same accuracy as the random forest model, 68.05%. When it comes to F1 scores, the SVM model had .77 and .46 for non-potable and potable, respectively. Overall, the SVM model performed similarly to the random forest model, which has been the best so far. See appendix 14 for full SVM outputs.



```
Confusion Matrix (SVM):
[[329  31]
 [162  82]]
```

Part 6: Results and Analysis

Based on the evaluation of the various machine learning models for water potability, it's crucial to consider the trade-offs between overall accuracy and the F1-score, particularly given the dataset's class imbalance. While accuracy provides a straightforward measure of total correct predictions, it can be misleading when one class significantly outnumbers the other, as was the case with our non-potable and potable water samples. The F1-score, as the harmonic mean of precision and recall, offers a more robust indicator of a model's performance by balancing the concerns of false positives and false negatives, making it especially valuable for assessing performance on the minority "Potable" class. For instance, the Tuned Decision Tree showed a high overall accuracy but significantly sacrificed recall for the potable class, a trade-off revealed by its low F1-score for that specific class. In contrast, the Random Forest model achieved the highest overall accuracy while also demonstrating a more balanced and higher F1-score for both classes, indicating a more robust performance across the board. The SVM model presented a similar overall accuracy to the Random Forest but with a slightly different balance in F1-scores, suggesting its strengths lay more in precision for potable water. Ultimately, while overall accuracy gives a general sense of model effectiveness, the F1-score is essential for understanding performance nuances.

Model	**Accuracy**
Decision Tree	60.26%
Decision Tree (opt. hyperparameters)	60.26%
Random Forest	68.05%
Boosting	62.25%
SVM	68.05%

Model	F1 Scores (Non-Potable / Potable)
Decision Tree	.67 / .51
Decision Tree (opt. hyperparameters)	.73 / .25
Random Forest	.76 / .50
Boosting	.73 / .39
SVM	.77 / .46

Despite exploring various machine learning models, the Random Forest classifier emerged as the best performer in this project, achieving the highest overall accuracy of 68.05% and demonstrating the most balanced F1-scores across both non-potable and potable water classes. While this level of accuracy is a significant improvement over the base Decision Tree and Gradient Boosting models, it is crucial to recognize that an accuracy of approximately 68% would be largely unacceptable for a critical, real-world application such as determining water potability, where misclassifications could have severe health implications. However, for the specific purpose of this project, which involved a synthetic dataset designed for model training and exploration, this accuracy score is beneficial. It effectively demonstrates the learning capabilities of the model and provides valuable insights into the patterns, however subtle, that can be extracted from such data to inform future analyses.

Part 7: Discussion and Conclusion

This project provided valuable hands-on experience in the complete machine learning pipeline, from initial data exploration and rigorous cleaning to model selection, training, and evaluation. A key learning outcome centered on understanding the profound impact of data characteristics on model performance. Our initial Exploratory Data Analysis, particularly the KDE plots and heatmap, revealed that the input variables were unusually normally distributed and exhibited limited linear correlation, with very little distinction between the distributions for potable and non-potable water. These observations strongly indicated the synthetic nature of the dataset. The subsequent model evaluation across Decision Trees (untuned and tuned), Random Forest, Gradient Boosting, and Support Vector Machines clearly demonstrated how these inherent data characteristics became a primary limiting factor for predictive accuracy. Despite implementing advanced ensemble methods like Random Forest and SVM, which achieved the highest accuracy of 68.05%, the models consistently struggled with the "Potable" class, often exhibiting lower recall and F1-scores. This persistent challenge was not a failure of the methodologies themselves, but rather a direct consequence of the synthetic data's design, where the features did not provide sufficiently distinct boundaries to differentiate between drinkable and non-drinkable water with high confidence.

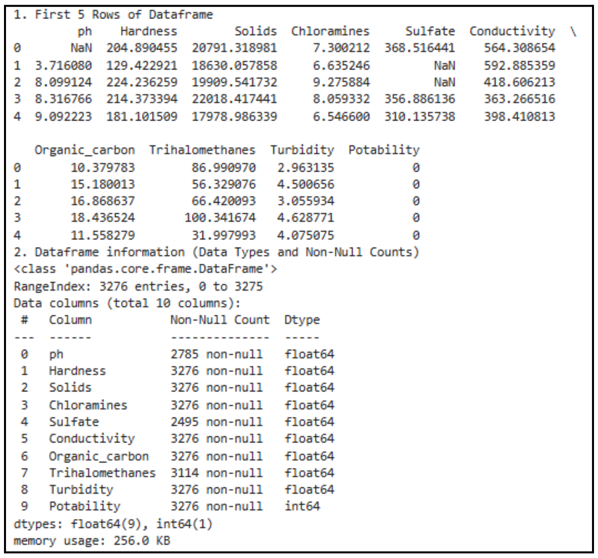
Crucially, the methodologies employed throughout this project are directly transferable and highly effective for real-world datasets. The systematic approach to data cleaning is fundamental to any robust data analysis. Similarly, the process of iteratively selecting, training, and evaluating diverse machine learning models, including hyperparameter tuning and the use of ensemble techniques, represents best practices in the field. Had this project been conducted with a real-world water quality dataset, which would likely exhibit more complex distributions, clearer class separation, and potentially non-normal data, these same methodologies would undoubtedly yield higher predictive performance and more actionable insights. This project served as an excellent practical exercise, solidifying the understanding that while sophisticated models are powerful, their ultimate effectiveness is inextricably linked to the quality and inherent separability of the underlying data.

Appendix

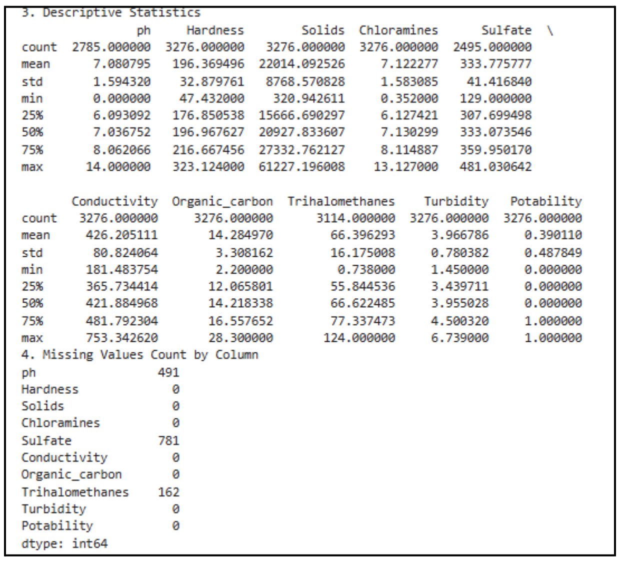
Appendix 1, link to dataset webpage)
https://www.kaggle.com/datasets/adityakadiwal/water-potability/data

Appendix 2, link to Jupyter Notebook)
https://jupyter.org/try-jupyter/lab/index.html?path=Supervised+Learning.ipynb

Appendix 3, outcomes of summary statistics measures)

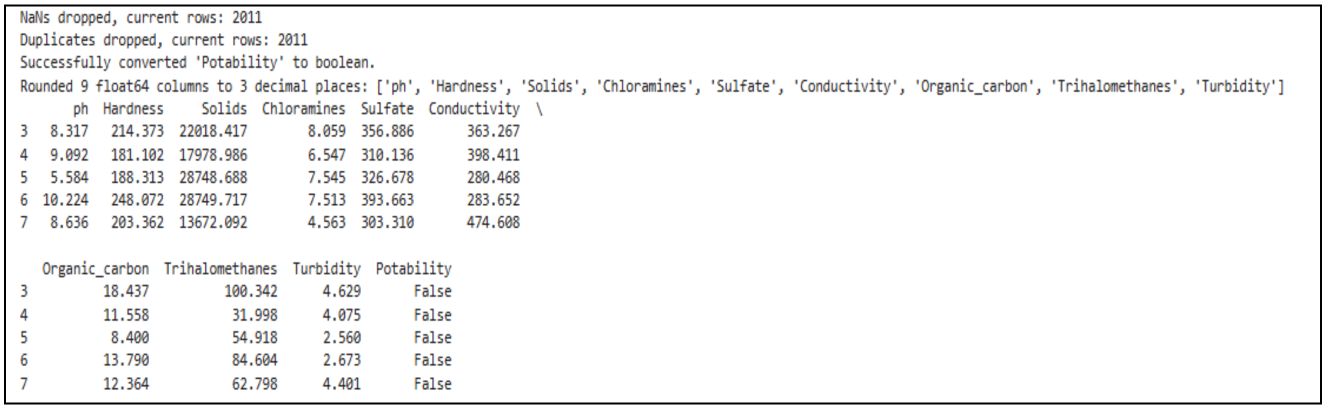


```
1. First 5 Rows of Dataframe
         ph    Hardness        Solids  Chloramines     Sulfate  Conductivity  \
0       NaN  204.890455  20791.318981     7.300212  368.516441    564.308654
1  3.716080  129.422921  18630.057858     6.635246         NaN    592.885359
2  8.099124  224.236259  19909.541732     9.275884         NaN    418.606213
3  8.316766  214.373394  22018.417441     8.059332  356.886136    363.266516
4  9.092223  181.101509  17978.986339     6.546600  310.135738    398.410813

   Organic_carbon  Trihalomethanes  Turbidity  Potability
0       10.379783        86.990970   2.963135           0
1       15.180013        56.329076   4.500656           0
2       16.868637        66.420093   3.055934           0
3       18.436524       100.341674   4.628771           0
4       11.558279        31.997993   4.075075           0
2. Dataframe information (Data Types and Non-Null Counts)
<class 'pandas.core.frame.DataFrame'>
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: 256.0 KB
```



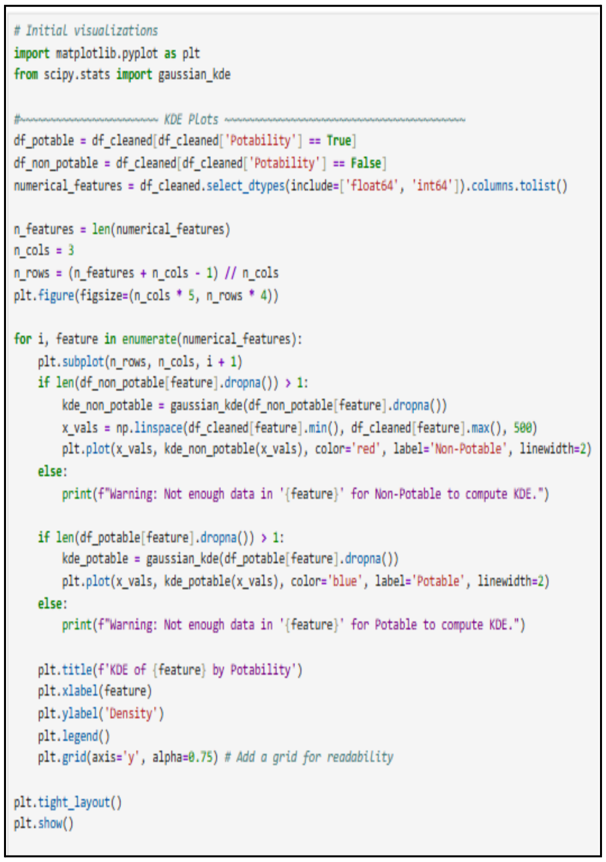
```
3. Descriptive Statistics
                ph     Hardness        Solids  Chloramines      Sulfate  \
count  2785.000000  3276.000000   3276.000000  3276.000000  2495.000000
mean      7.080795   196.369496  22014.092526     7.122277   333.775777
std       1.594320    32.879761   8768.570828     1.583085    41.416840
min       0.000000    47.432000    320.942611     0.352000   129.000000
25%       6.093092   176.850538  15666.690297     6.127421   307.699498
50%       7.036752   196.967627  20927.833607     7.130299   333.073546
75%       8.062066   216.667456  27332.762127     8.114887   359.950170
max      14.000000   323.124000  61227.196008    13.127000   481.030642

       Conductivity  Organic_carbon  Trihalomethanes    Turbidity   Potability
count   3276.000000     3276.000000      3114.000000  3276.000000  3276.000000
mean     426.205111       14.284970        66.396293     3.966786     0.390110
std       80.824064        3.308162        16.175008     0.780382     0.487849
min      181.483754        2.200000         0.738000     1.450000     0.000000
25%      365.734414       12.065801        55.844536     3.439711     0.000000
50%      421.884968       14.218338        66.622485     3.955028     0.000000
75%      481.792304       16.557652        77.337473     4.500320     1.000000
max      753.342620       28.300000       124.000000     6.739000     1.000000
4. Missing Values Count by Column
ph                 491
Hardness             0
Solids               0
Chloramines          0
Sulfate            781
Conductivity         0
Organic_carbon       0
Trihalomethanes    162
Turbidity            0
Potability           0
dtype: int64
```

Appendix 4, outcomes of cleaning measures)

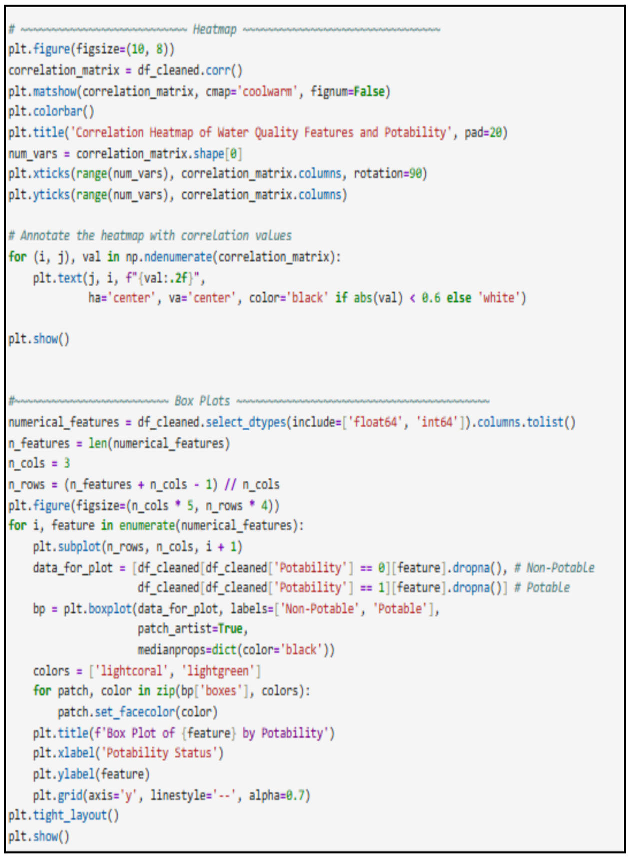


```
NaNs dropped, current rows: 2011
Duplicates dropped, current rows: 2011
Successfully converted 'Potability' to boolean.
Rounded 9 float64 columns to 3 decimal places: ['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity', 'Organic_carbon', 'Trihalomethanes', 'Turbidity']
       ph  Hardness     Solids  Chloramines  Sulfate  Conductivity  \
3   8.317   214.373  22018.417        8.059  356.886       363.267
4   9.092   181.102  17978.986        6.547  310.136       398.411
5   5.584   188.313  28748.688        7.545  326.678       280.468
6  10.224   248.072  28749.717        7.513  393.663       283.652
7   8.636   203.362  13672.092        4.563  303.310       474.608

   Organic_carbon  Trihalomethanes  Turbidity  Potability
3          18.437          100.342      4.629       False
4          11.558           31.998      4.075       False
5           8.400           54.918      2.560       False
6          13.790           84.604      2.673       False
7          12.364           62.798      4.401       False
```

Appendix 5, code for exploratory data analysis)

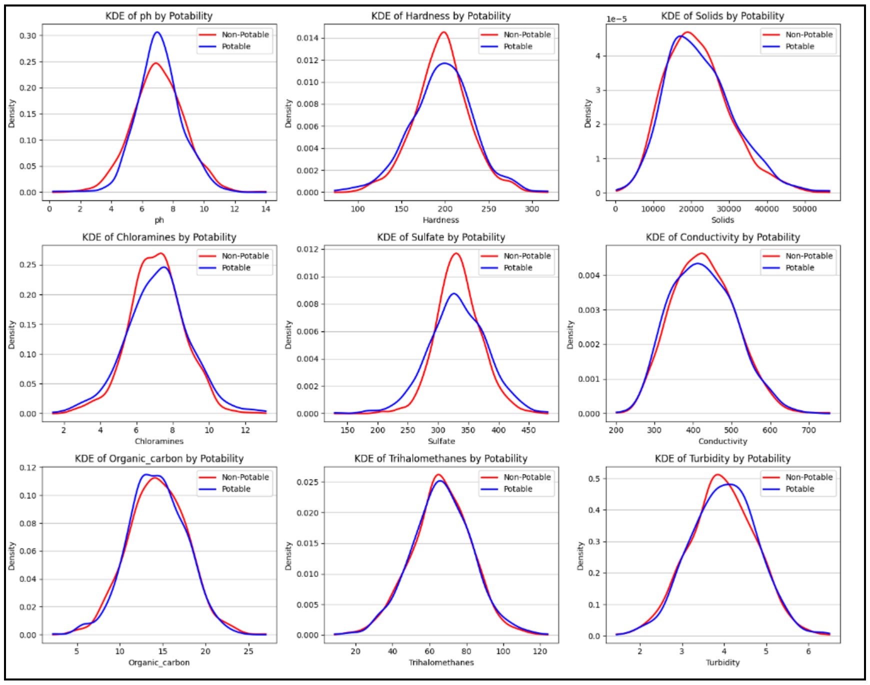


```python
# Initial visualizations
import matplotlib.pyplot as plt
from scipy.stats import gaussian_kde

#~~~~~~~~~~~~~~~~~~~~~~~~ KDE Plots ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
df_potable = df_cleaned[df_cleaned['Potability'] == True]
df_non_potable = df_cleaned[df_cleaned['Potability'] == False]
numerical_features = df_cleaned.select_dtypes(include=['float64', 'int64']).columns.tolist()

n_features = len(numerical_features)
n_cols = 3
n_rows = (n_features + n_cols - 1) // n_cols
plt.figure(figsize=(n_cols * 5, n_rows * 4))

for i, feature in enumerate(numerical_features):
    plt.subplot(n_rows, n_cols, i + 1)
    if len(df_non_potable[feature].dropna()) > 1:
        kde_non_potable = gaussian_kde(df_non_potable[feature].dropna())
        x_vals = np.linspace(df_cleaned[feature].min(), df_cleaned[feature].max(), 500)
        plt.plot(x_vals, kde_non_potable(x_vals), color='red', label='Non-Potable', linewidth=2)
    else:
        print(f"Warning: Not enough data in '{feature}' for Non-Potable to compute KDE.")

    if len(df_potable[feature].dropna()) > 1:
        kde_potable = gaussian_kde(df_potable[feature].dropna())
        plt.plot(x_vals, kde_potable(x_vals), color='blue', label='Potable', linewidth=2)
    else:
        print(f"Warning: Not enough data in '{feature}' for Potable to compute KDE.")

    plt.title(f'KDE of {feature} by Potability')
    plt.xlabel(feature)
    plt.ylabel('Density')
    plt.legend()
    plt.grid(axis='y', alpha=0.75) # Add a grid for readability

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

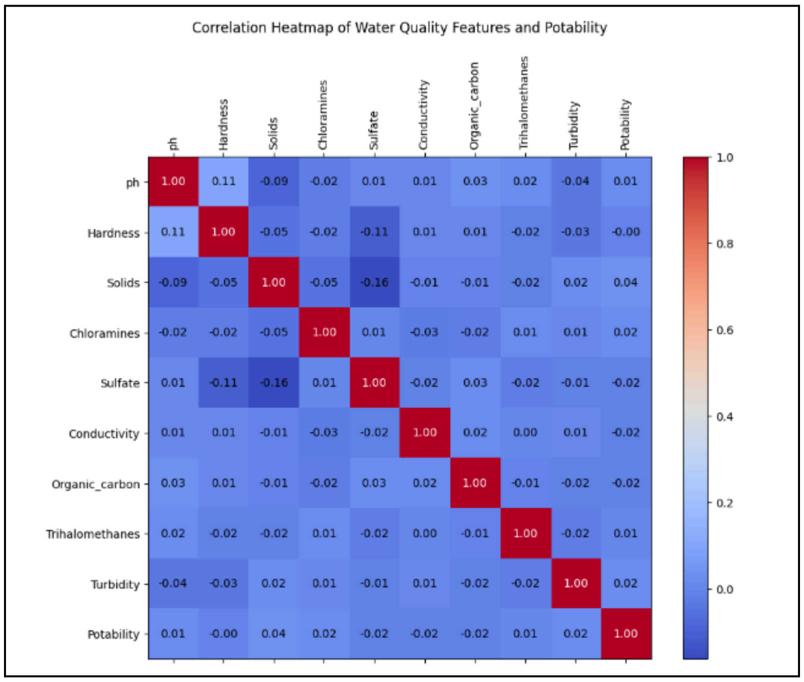


```python
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Heatmap ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
plt.figure(figsize=(10, 8))
correlation_matrix = df_cleaned.corr()
plt.matshow(correlation_matrix, cmap='coolwarm', fignum=False)
plt.colorbar()
plt.title('Correlation Heatmap of Water Quality Features and Potability', pad=20)
num_vars = correlation_matrix.shape[0]
plt.xticks(range(num_vars), correlation_matrix.columns, rotation=90)
plt.yticks(range(num_vars), correlation_matrix.columns)

# Annotate the heatmap with correlation values
for (i, j), val in np.ndenumerate(correlation_matrix):
    plt.text(j, i, f"{val:.2f}",
             ha='center', va='center', color='black' if abs(val) < 0.6 else 'white')

plt.show()


#~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Box Plots ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
numerical_features = df_cleaned.select_dtypes(include=['float64', 'int64']).columns.tolist()
n_features = len(numerical_features)
n_cols = 3
n_rows = (n_features + n_cols - 1) // n_cols
plt.figure(figsize=(n_cols * 5, n_rows * 4))
for i, feature in enumerate(numerical_features):
    plt.subplot(n_rows, n_cols, i + 1)
    data_for_plot = [df_cleaned[df_cleaned['Potability'] == 0][feature].dropna(), # Non-Potable
                     df_cleaned[df_cleaned['Potability'] == 1][feature].dropna()] # Potable
    bp = plt.boxplot(data_for_plot, labels=['Non-Potable', 'Potable'],
                     patch_artist=True,
                     medianprops=dict(color='black'))
    colors = ['lightcoral', 'lightgreen']
    for patch, color in zip(bp['boxes'], colors):
        patch.set_facecolor(color)
    plt.title(f'Box Plot of {feature} by Potability')
    plt.xlabel('Potability Status')
    plt.ylabel(feature)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

Appendix 6, KDE plots by input variable)

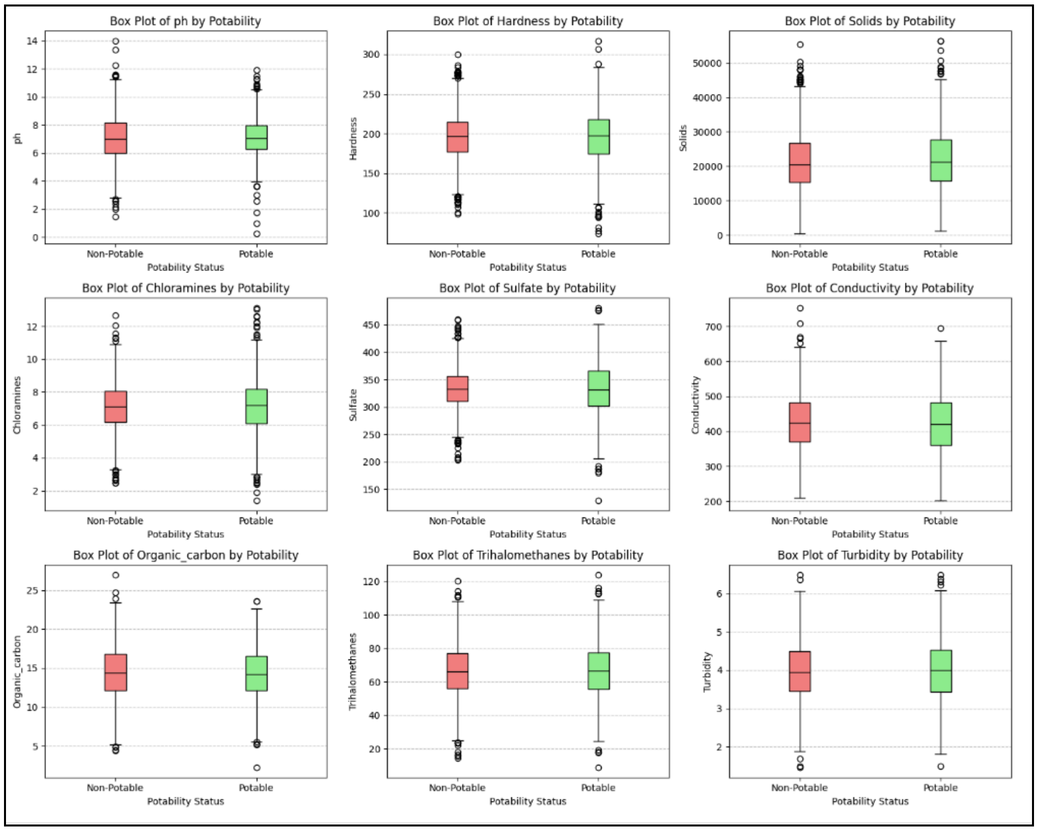


Appendix 7, heatmap for input and output variables)

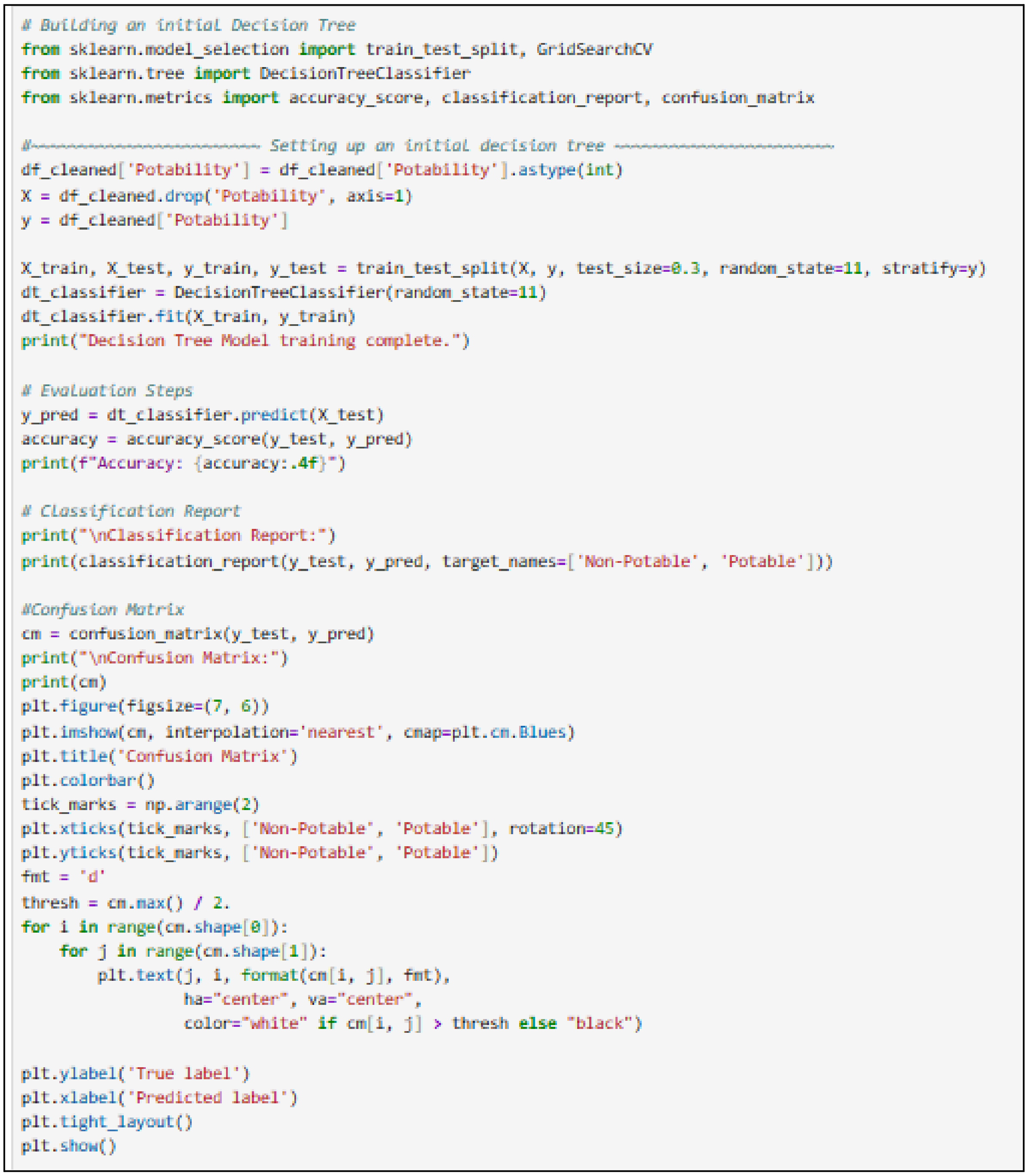


	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
ph	1.00	0.11	-0.09	-0.02	0.01	0.01	0.03	0.02	-0.04	0.01
Hardness	0.11	1.00	-0.05	-0.02	-0.11	0.01	0.01	-0.02	-0.03	-0.00
Solids	-0.09	-0.05	1.00	-0.05	-0.16	-0.01	-0.01	-0.02	0.02	0.04
Chloramines	-0.02	-0.02	-0.05	1.00	0.01	-0.03	-0.02	0.01	0.01	0.02
Sulfate	0.01	-0.11	-0.16	0.01	1.00	-0.02	0.03	-0.02	-0.01	-0.02
Conductivity	0.01	0.01	-0.01	-0.03	-0.02	1.00	0.02	0.00	0.01	-0.02
Organic_carbon	0.03	0.01	-0.01	-0.02	0.03	0.02	1.00	-0.01	-0.02	-0.02
Trihalomethanes	0.02	-0.02	-0.02	0.01	-0.02	0.00	-0.01	1.00	-0.02	0.01
Turbidity	-0.04	-0.03	0.02	0.01	-0.01	0.01	-0.02	-0.02	1.00	0.02
Potability	0.01	-0.00	0.04	0.02	-0.02	-0.02	-0.02	0.01	0.02	1.00

Appendix 8, boxplots for each input variable)

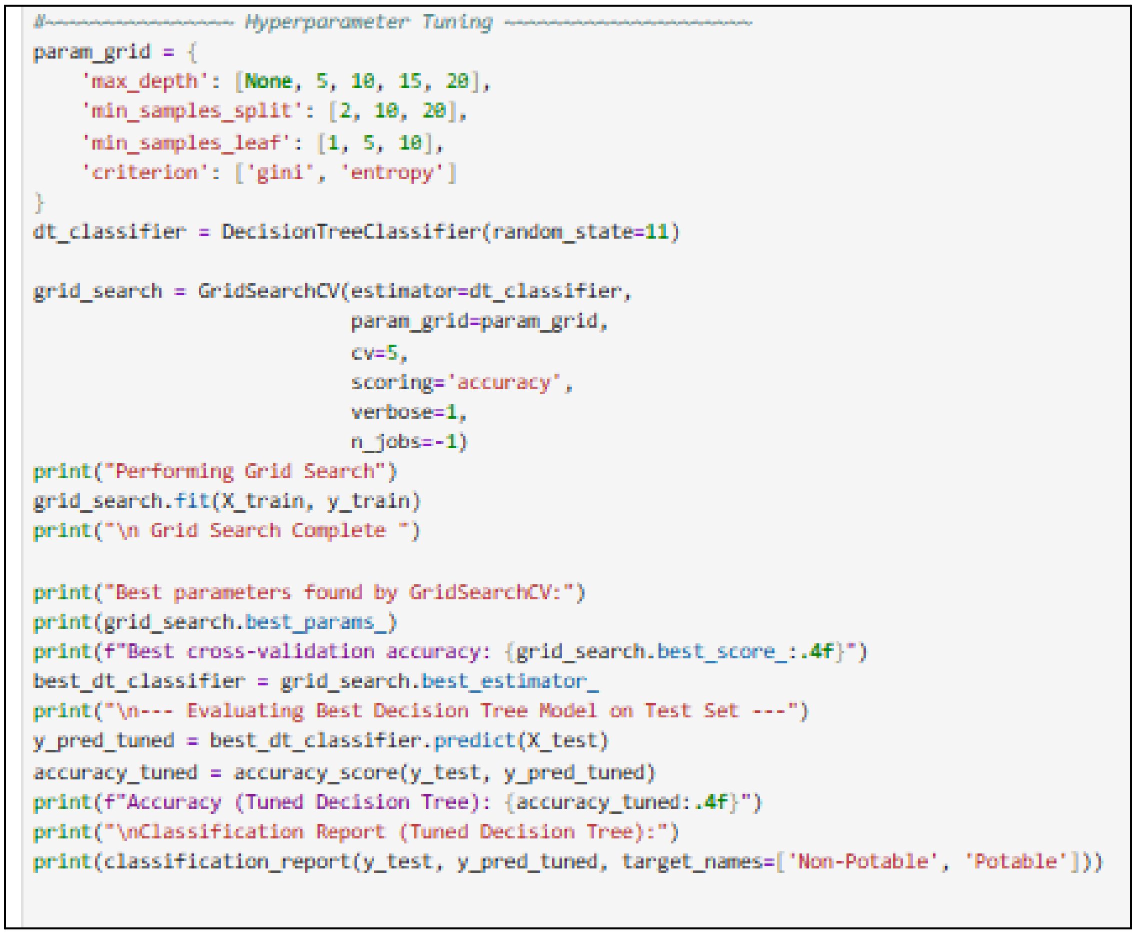


Appendix 9, code to set up and train the initial decision tree)



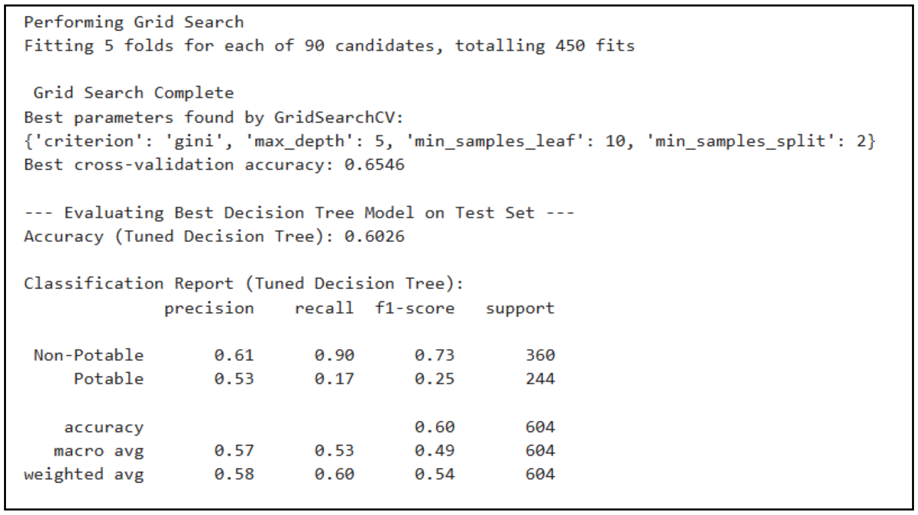
```python
# Building an initial Decision Tree
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

#-------------------------- Setting up an initial decision tree -------------------------
df_cleaned['Potability'] = df_cleaned['Potability'].astype(int)
X = df_cleaned.drop('Potability', axis=1)
y = df_cleaned['Potability']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=11, stratify=y)
dt_classifier = DecisionTreeClassifier(random_state=11)
dt_classifier.fit(X_train, y_train)
print("Decision Tree Model training complete.")

# Evaluation Steps
y_pred = dt_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['Non-Potable', 'Potable']))

#Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(cm)
plt.figure(figsize=(7, 6))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(2)
plt.xticks(tick_marks, ['Non-Potable', 'Potable'], rotation=45)
plt.yticks(tick_marks, ['Non-Potable', 'Potable'])
fmt = 'd'
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], fmt),
                 ha="center", va="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.tight_layout()
plt.show()
```

Appendix 10, code to adjust hyperparameters and tune decision tree)



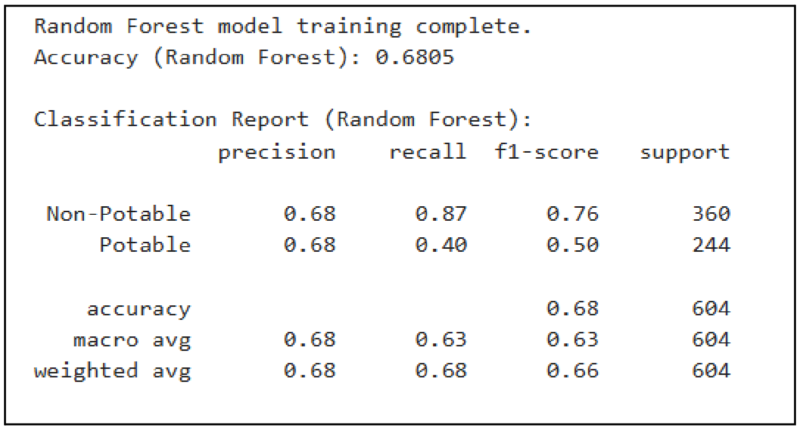
```
#------------------ Hyperparameter Tuning --------------------
param_grid = {
    'max_depth': [None, 5, 10, 15, 20],
    'min_samples_split': [2, 10, 20],
    'min_samples_leaf': [1, 5, 10],
    'criterion': ['gini', 'entropy']
}
dt_classifier = DecisionTreeClassifier(random_state=11)

grid_search = GridSearchCV(estimator=dt_classifier,
                           param_grid=param_grid,
                           cv=5,
                           scoring='accuracy',
                           verbose=1,
                           n_jobs=-1)
print("Performing Grid Search")
grid_search.fit(X_train, y_train)
print("\n Grid Search Complete ")

print("Best parameters found by GridSearchCV:")
print(grid_search.best_params_)
print(f"Best cross-validation accuracy: {grid_search.best_score_:.4f}")
best_dt_classifier = grid_search.best_estimator_
print("\n--- Evaluating Best Decision Tree Model on Test Set ---")
y_pred_tuned = best_dt_classifier.predict(X_test)
accuracy_tuned = accuracy_score(y_test, y_pred_tuned)
print(f"Accuracy (Tuned Decision Tree): {accuracy_tuned:.4f}")
print("\nClassification Report (Tuned Decision Tree):")
print(classification_report(y_test, y_pred_tuned, target_names=['Non-Potable', 'Potable']))
```

Appendix 11, output of GridSearch function)



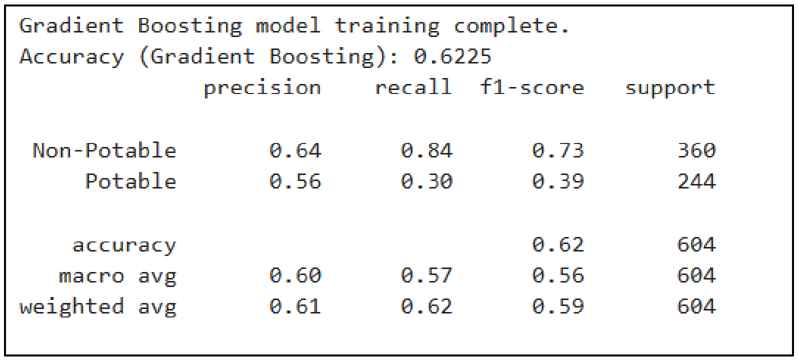
```
Performing Grid Search
Fitting 5 folds for each of 90 candidates, totalling 450 fits

 Grid Search Complete
Best parameters found by GridSearchCV:
{'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 10, 'min_samples_split': 2}
Best cross-validation accuracy: 0.6546

--- Evaluating Best Decision Tree Model on Test Set ---
Accuracy (Tuned Decision Tree): 0.6026

Classification Report (Tuned Decision Tree):
              precision    recall  f1-score   support

 Non-Potable       0.61      0.90      0.73       360
     Potable       0.53      0.17      0.25       244

    accuracy                           0.60       604
   macro avg       0.57      0.53      0.49       604
weighted avg       0.58      0.60      0.54       604
```

Appendix 12, accuracy and classification report for Random Forest)



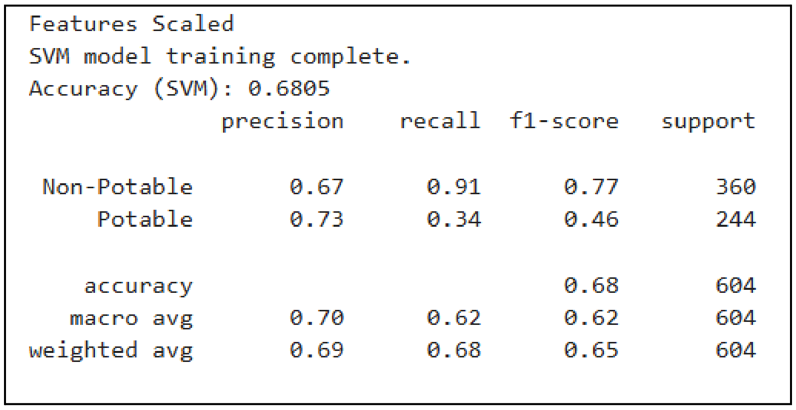
```
Random Forest model training complete.
Accuracy (Random Forest): 0.6805

Classification Report (Random Forest):
              precision    recall  f1-score   support

 Non-Potable       0.68      0.87      0.76       360
     Potable       0.68      0.40      0.50       244

    accuracy                           0.68       604
   macro avg       0.68      0.63      0.63       604
weighted avg       0.68      0.68      0.66       604
```

Appendix 13, accuracy and classification report for Boosting Model)



```
Gradient Boosting model training complete.
Accuracy (Gradient Boosting): 0.6225
              precision    recall  f1-score   support

 Non-Potable       0.64      0.84      0.73       360
     Potable       0.56      0.30      0.39       244

    accuracy                           0.62       604
   macro avg       0.60      0.57      0.56       604
weighted avg       0.61      0.62      0.59       604
```

Appendix 14, accuracy and classification report for SVM)



```
Features Scaled
SVM model training complete.
Accuracy (SVM): 0.6805
              precision    recall  f1-score   support

 Non-Potable       0.67      0.91      0.77       360
     Potable       0.73      0.34      0.46       244

    accuracy                           0.68       604
   macro avg       0.70      0.62      0.62       604
weighted avg       0.69      0.68      0.65       604
```