Water Potability Analysis By Sameed

This document will guide you through the development process for creating an API that predicts water potability using a trained machine learning model. The API will take input data and return predictions on whether the water is potable or not.

1. Introduction

Water quality is essential for human health and overall well-being. This project aims to predict water potability, which is crucial for ensuring safe drinking water. Clean and potable water is fundamental for preventing diseases and sustaining healthy communities. Additionally, it plays a vital role in economic development and environmental sustainability.

2. Challenges Faced

The two major challenges faced were during Hyperparameter Tuning and API creation. There are several ways to optimize the hyperparameters, but I went with Grid Search because of its accuracy. However, I did not anticipate how long it would actually take, and in the end, my laptop was executing the code for 6 hours and 57 minutes.

I did not have previous experience with API creation, but I managed to get around Flask API with lots of research and help. There were many issues faced in this area, but with constant troubleshooting, I got around to it. The biggest issue I faced was that the results from this API were not accurate, after thorough research, I realized the reason behind this is that the model was trained on scaled data, but the API's input data was raw and not scaled, I fixed this issue by scaling the input data on the API as well.

3. Essential Libraries

To perform data analysis, develop a model, and create the API, we'll need several libraries:

```
#Essential Libraries for performing analysis
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import BoundaryNorm
from matplotlib.cm import get_cmap
import numpy as np
import seaborn as sns

#Machine Learing Libraries
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.inspection import permutation_importance

#Library to Save & Load Model
import joblib
```

```
from fastapi import FastAPI, Request, Form
from fastapi.templating import Jinja2Templates
from pydantic import BaseModel
import joblib
import numpy as np
import json
from sklearn.preprocessing import StandardScaler
```

4. Data Preprocessing

Handling Missing Values

Missing values were found in 'ph', 'Sulfate', and 'Trihalomethanes', and they were replaced with their means.

```
[89] # Replacing ph missing values with their mean
     mean_value_1 =df['ph'].mean()
     df['ph'].fillna(value=mean_value_1, inplace=True)
```

```
[90] # Replacing Trihalomethanes missing values with their mean
     mean_value_1 =df['Trihalomethanes'].mean()
     df['Trihalomethanes'].fillna(value=mean_value_1, inplace=True)
```

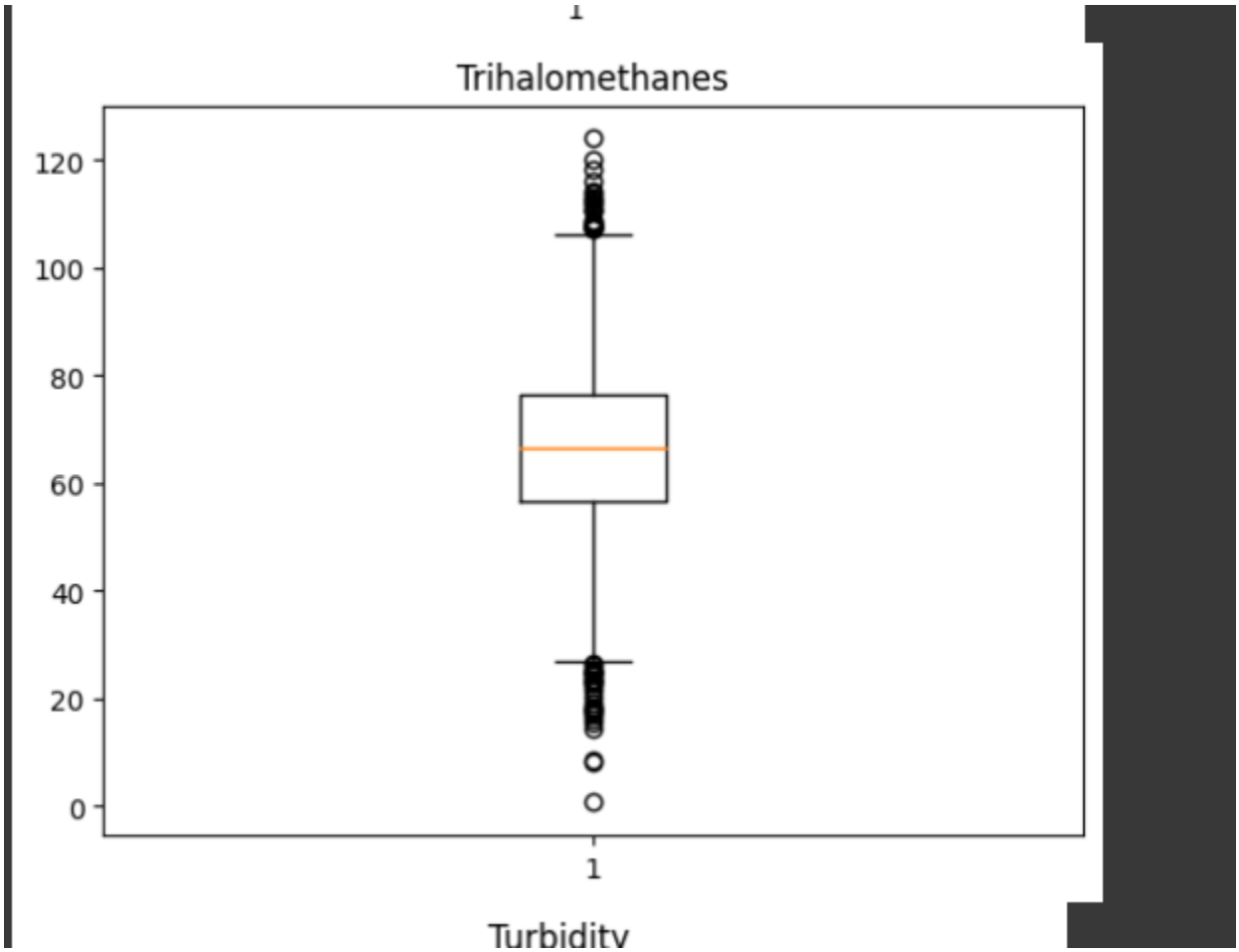
```
[91] # Replacing Sulfate missing values with their mean
     mean_value_2 =df['Sulfate'].mean()
     df['Sulfate'].fillna(value=mean_value_2, inplace=True)
```

Handling Outliers

All outliers were removed, because there weren't too many of them.

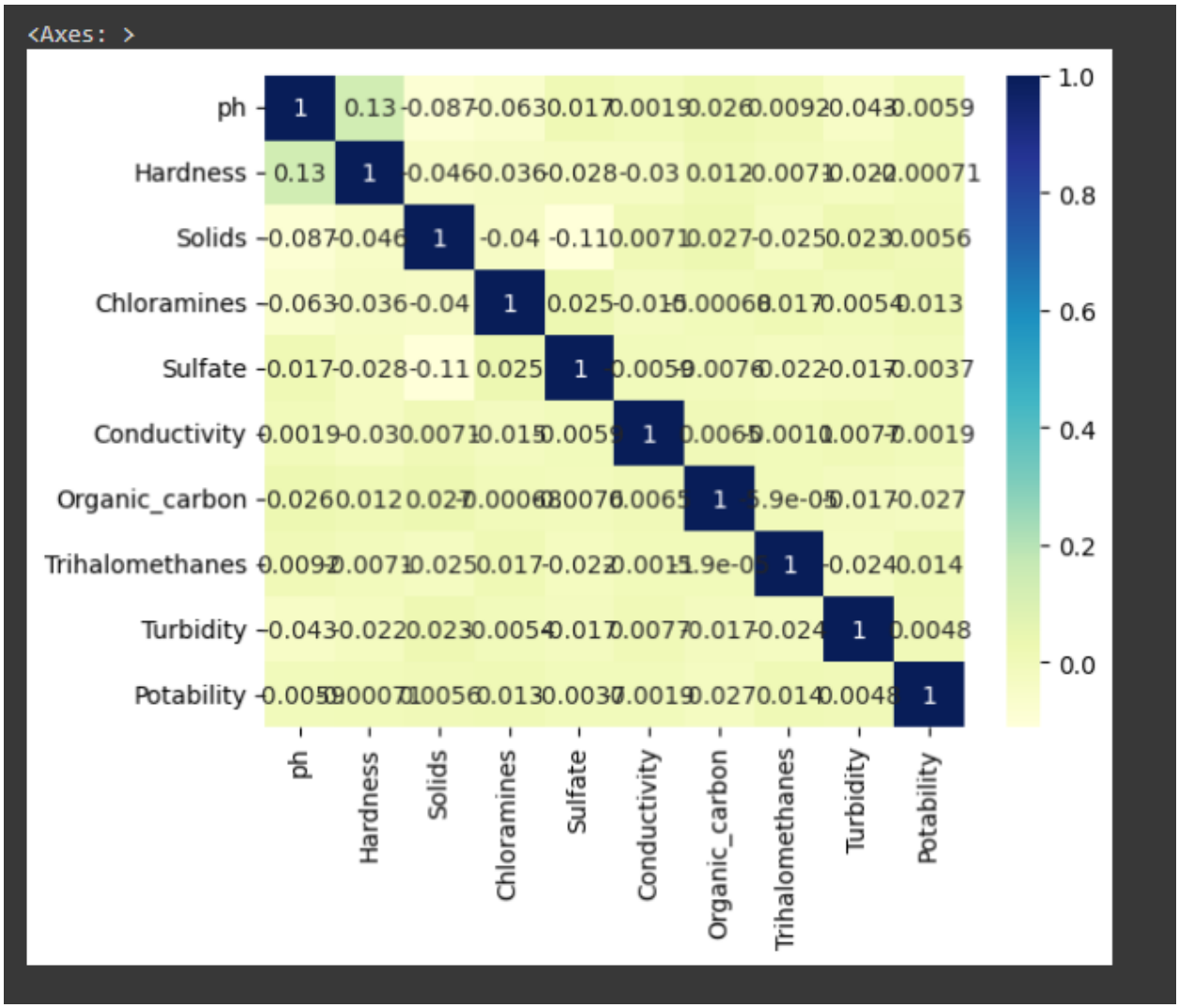
```
[96] #Dropping outliers
     Q1 = df.quantile(0.25)
     Q3 = df.quantile(0.75)
     IQR = Q3 - Q1
     df = df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
     df
```

Outliers were spotted using boxplots, eg:



5. Checking Correlation

The correlation was checked using Heatmaps. The conclusion was that there is no linear relation between the features; hence, a non-linear model was used.



6. Data Modeling

Splitting and Transforming Data

Split the dataset into training and testing sets and scale the data:

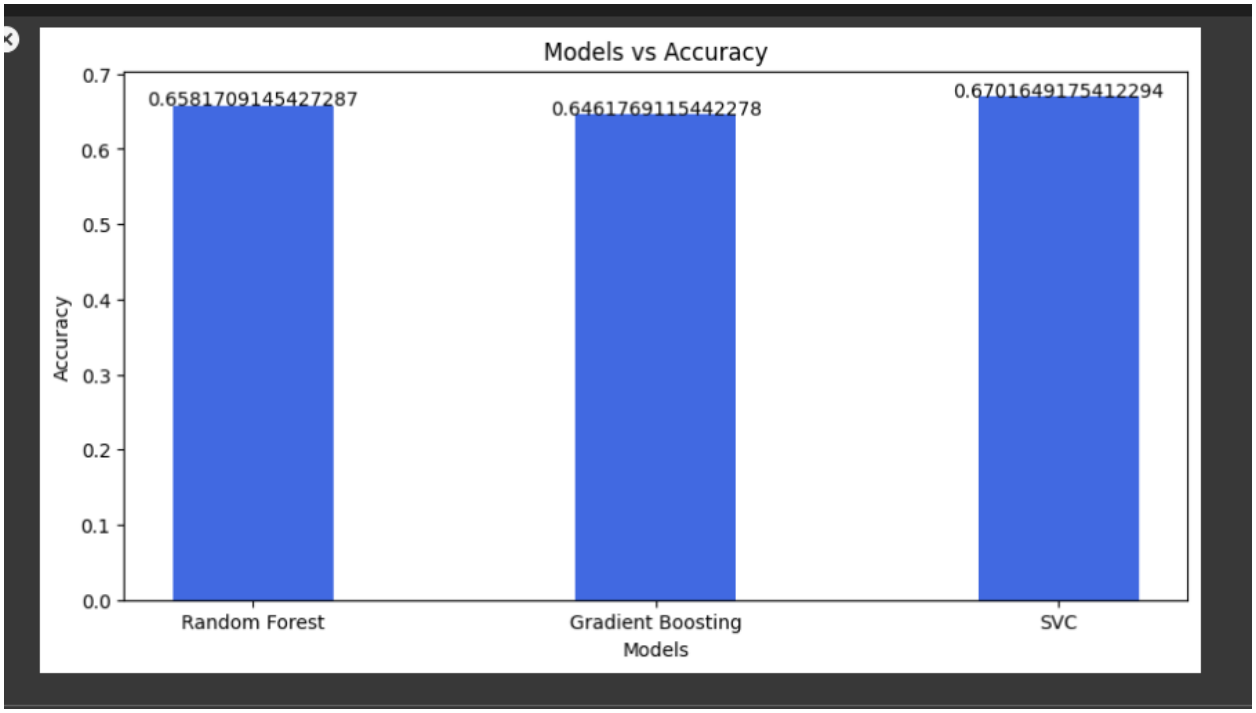
```
[100] #Splitting the dataset and scaling it to improve the performance
      x=df.drop(['Potability'], axis=1)
      y=df['Potability']
      X_train,X_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=42)
      sc = StandardScaler()
      X_train = sc.fit_transform(X_train)
      X_test = sc.transform(X_test)
```

Applying Models for Prediction

Train multiple machine learning models (Random Forest, Gradient Boosting, and SVC) and evaluate them using accuracy as the metric. I have used accuracy_score as the evaluation metric because it is generally considered a simple metric. It is not biased towards any class. I also calculated the F1 scores of each of these models and they yielded the same results.

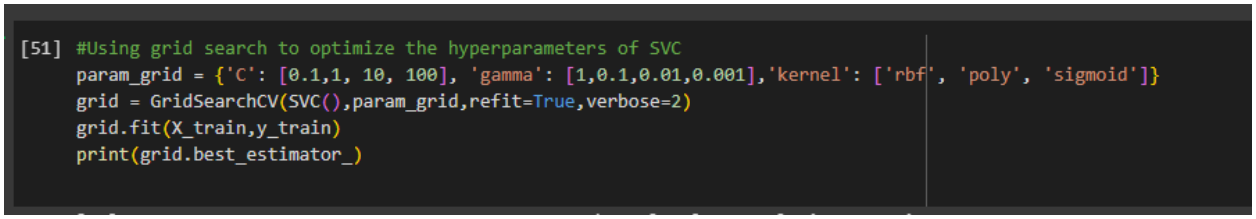
```
[101] #Trying three models for their accuracy
      models = [RandomForestClassifier(), GradientBoostingClassifier(), SVC()]
      accuracy= []
      for model in models:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        accuracy.append(accuracy_score(y_test, y_pred))

      def addlabels(x,y):
          for i in range(len(x)):
              plt.text(i, y[i], y[i], ha = 'center')

      models = ["Random Forest", "Gradient Boosting", "SVC"]
      fig = plt.figure(figsize = (10, 5))
      plt.bar(models, accuracy, color ='royalblue',
              width = 0.4)
      addlabels(models, accuracy)
      #Plotting their accuracy to visualize the best performing model
      plt.xlabel("Models")
      plt.ylabel("Accuracy")
      plt.title("Models vs Accuracy")
      plt.show()
```

The results showed SVM as the most accurate model.



7. Hyperparameter Tuning

Optimize the hyperparameters of SVC using grid search:



```
[51] #Using grid search to optimize the hyperparameters of SVC
     param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly', 'sigmoid']}
     grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=2)
     grid.fit(X_train,y_train)
     print(grid.best_estimator_)
```

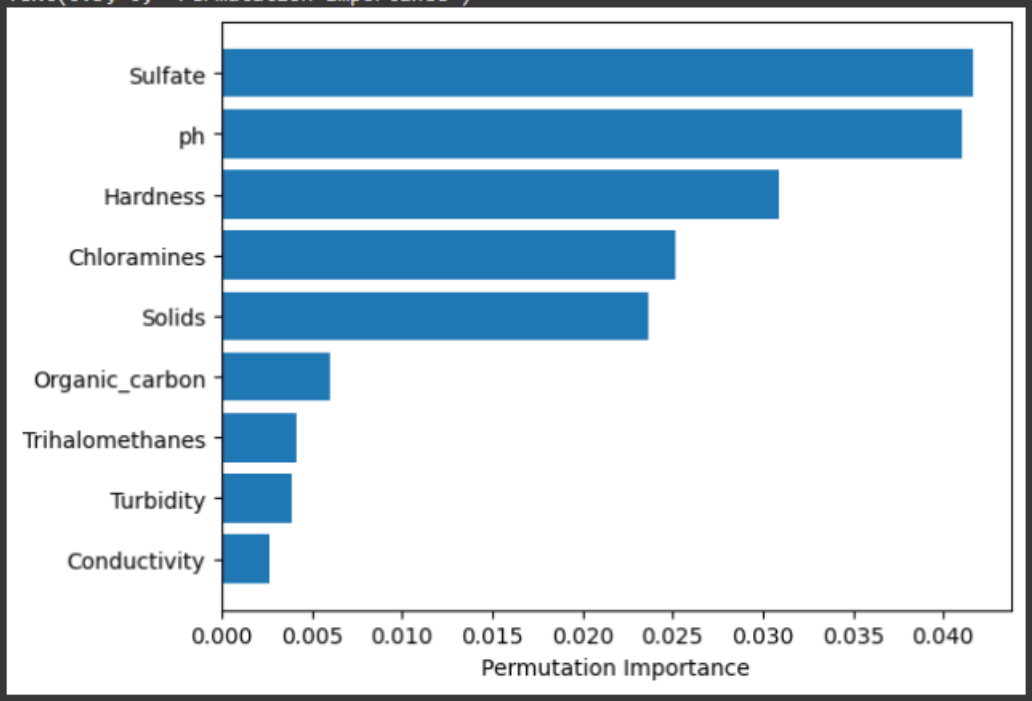
After 7 hours of execution, this was the result : SVC(C=1, gamma=0.1)

8. Feature Importance

The following features have the most importance:

```
[103] #Figuring out the feature importance
      perm_importance = permutation_importance(model, X_test, y_test)

      feature_names = df.columns
      features = np.array(feature_names)

      sorted_idx = perm_importance.importances_mean.argsort()
      plt.barh(features[sorted_idx], perm_importance.importances_mean[sorted_idx])
      plt.xlabel("Permutation Importance")
```

```
Text(0.5, 0, 'Permutation Importance')
```



9. Model Save

Finally, the model was saved as:

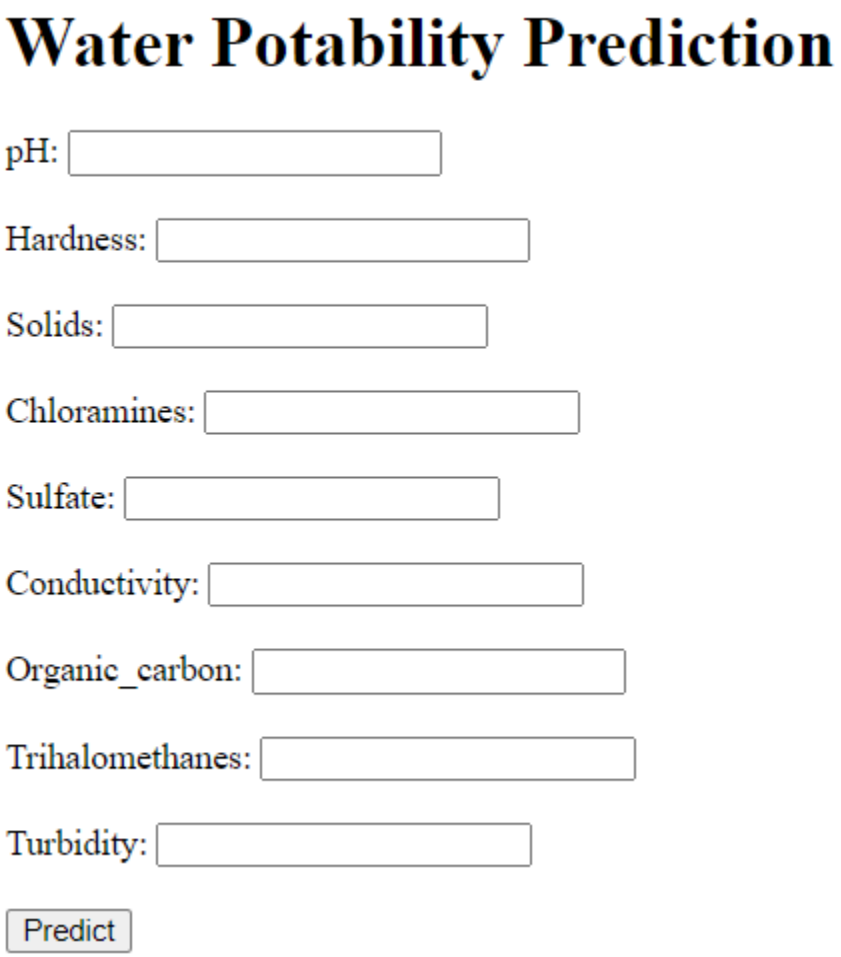
```
[105] #Exporting the trained model as a pickle file for API integration
      joblib.dump(model, 'model.pkl')

      ['model.pkl']
```

10. API Creation

FastAPI was used for API creation,

```python
# Create an endpoint to receive POST requests with input data
@app.post("/predict")
async def predict(request: Request, json_data: str = Form(...)):
    features = json.loads(json_data)
    # Call the predict_potability function to make predictions
    prediction = predict_potability(WaterPotabilityRequest(**features))

    # Render the HTML template with the prediction result
    return templates.TemplateResponse("index.html", {"request": request, "prediction": prediction})

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

11. Prediction



An HTML form was also created to go along with the API.