4. IMPLEMENTATION CODE

```
import pandas as pd import

seaborn as sns import

numpy as np

import matplotlib. pyplot as plt data =

pd.read_csv("water_potability.csv") data

%pip install pandas numpy matplotlib seaborn

data.isnull().sum() data = data.dropna()

data.isnull().sum() data.head() data.info()

data.describe().T data.skew(axis=0,

skipna=True)

sns.pairplot(data = data, hue = 'Potability')

plt.show() data.corr()

plt.figure(figsize=(20,10))

sns.heatmap(data.corr(), cmap='YlGnBu', annot=True)

plt.show() sns.countplot(x = 'Potability', data = data)

plt.title("Distribution of Unsafe and Safe Water")

plt.show()

%pip install plotly nbformat

%pip install plotly --upgrade nbformat import plotly.express as px data = data figure =

px.histogram(data, x = "ph", color = "Potability", title= "Factors Affecting Water Quality: PH")

figure = px.histogram(data, x = "Hardness", color = "Potability", title= "Factors Affecting Water Quality:
Hardness") figure = px.histogram(data, x = "Solids", color = "Potability", title= "Factors Affecting Water
Quality: Solids")

figure = px.histogram(data, x = "Chloramines", color = "Potability", title= "Factors Affecting Water Quality:
Chloramines")
```

figure = px.histogram(data, x = "Sulfate", color = "Potability", title= "Factors Affecting Water Quality: Sulfate")

figure = px.histogram(data, x = "Conductivity", color = "Potability", title= "Factors Affecting Water Quality: Conductivity")

figure = px.histogram(data, x = "Organic_carbon", color = "Potability", title= "Factors Affecting Water Quality: Organic Carbon")

figure = px.histogram(data, x = "Trihalomethanes", color = "Potability", title= "Factors Affecting Water

Quality: Trihalomethanes") figure = px.histogram(data, x = "Turbidity", color = "Potability", title=

"Factors Affecting Water Quality: Turbidity") sns.pairplot(data=data) data.boxplot(figsize=(15,6))

plt.show() data.skew(axis=0, skipna=True)

x = data.iloc[:,:-1] #leaving last column y = data.iloc[:,-1] x y cols = 'Potability'

print((data[cols] == 3).all()) print(data['Potability'].value_counts())

print(data['Potability'].unique()) features = data.iloc[:, 1:].values labels = data['Potability'].values

%pip install scikit-learn from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.20, shuffle=True, random_state=0)

a=x_train

a

b=y_train

b x_test

y_test

%pip install xgboost import xgboost as xgb from xgboost import

XGBClassifier, XGBRegressor from xgboost import plot_importance from

sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor from

sklearn.svm import SVC svc = SVC(kernel = 'linear', random_state=0

,probability=True) svc.fit(a,b)

from sklearn.naive_bayes import GaussianNB nb

= GaussianNB()

```
nb.fit(x_train, y_train) from sklearn.tree import DecisionTreeClassifier
dectree = DecisionTreeClassifier(criterion = 'entropy', random_state=0)
dectree.fit(x_train, y_train)

from sklearn.ensemble import RandomForestClassifier ranfor =
RandomForestClassifier(n_estimators = 11, criterion = 'entropy',
random_state=0) ranfor.fit(x_train, y_train) y_pred_svc =
svc.predict(x_test) y_pred_nb = nb.predict(x_test) y_pred_dectree =
dectree.predict(x_test) y_pred_ranfor = ranfor.predict(x_test) from
sklearn.metrics import accuracy_score

from sklearn.metrics import
precision_score,recall_score,f1_score,roc_curve,roc_auc_score,classification_report model_scores
= {
'XGB': accuracy_score(y_test, y_pred_svc)*100,
'KNN': accuracy_score(y_test, y_pred_nb)*100,
'SVM': accuracy_score(y_test, y_pred_dectree)*100,
'NB': accuracy_score(y_test, y_pred_ranfor)*100,
'DT': accuracy_score(y_test, y_pred_ranfor)*100, 'RF':
accuracy_score(y_test, y_pred_ranfor)*100}
model_compare=pd.DataFrame(model_scores,index=['Accuracy']) model_compare
def create_report(model, X_test, y_test):
   y_pred = model.predict(X_test)    report =
classification_report(y_test, y_pred)    acc =
accuracy_score(y_test, y_pred)
print(f"Accuracy : {acc*100:.4f} %\n")
print("Classification report: \n")
print(report)
```

```
from sklearn.metrics import classification_report print('XGB:',
classification_report(y_test, y_pred_dectree),'\n'
    'KNN:', classification_report(y_test, y_pred_svc),'\n'
    'SVM:', classification_report(y_test, y_pred_nb),'\n',
    'NB:', classification_report(y_test, y_pred_dectree),'\n',
    'DT:', classification_report(y_test, y_pred_ranfor),'\n'      'RF:',
classification_report(y_test, y_pred_ranfor)) from sklearn.model_selection import
StratifiedKFold, cross_val_score from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import precision_score, recall_score, f1_score import numpy as
np dt_model = DecisionTreeClassifier() k = 20 kf = StratifiedKFold(n_splits=k) from
sklearn.model_selection import cross_val_score, KFold from skleam.ensemble
import RandomForestClassifier # Replace with your classifier classifier =
RandomForestClassifier()
kf = KFold(n_splits=5, shuffle=True, random_state=42) accuracy_scores =
cross_val_score(classifier, features, labels, scoring='accuracy', cv=kf) precision_scores =
cross_val_score(classifier, features, labels, scoring='precision', cv=kf) recall_scores =
cross_val_score(classifier, features, labels, scoring='recall', cv=kf) f1_scores =
cross_val_score(classifier, features, labels, scoring='f1', cv=kf) print("Cross-Validation
Accuracy Scores:", accuracy_scores) print("Cross-Validation Precision Scores:",
precision_scores) print("Cross-Validation Recall Scores:", recall_scores) print("Cross-
Validation F1 Scores:", f1_scores) print("Mean Accuracy:", accuracy_scores.mean())
print("Accuracy:", np.mean(accuracy_scores) * 100)
print("Precision:", np.nanmean(precision_scores) * 100)
print("Recall:", np.nanmean(recall_scores) * 100) print("F1
Score:", np.nanmean(f1_scores) * 100)
XGB_acc=np.mean(accuracy_scores) * 100
KNN_acc=np.mean(precision_scores) * 100
```

```
SVM_acc=np.mean(recall_scores) * 100 NB_acc=np.mean(f1_scores)
* 100
from sklearn.model_selection import cross_val_score, KFold num_folds = 10 kf =
KFold(n_splits=num_folds, shuffle=True,random_state=42) cv_scores =
cross_val_score(dt_model,data,labels, cv=kf, scoring='accuracy')
print(f"\n{num_folds}-Fold Cross-Validation Scores:") print(cv_scores)
average_cv_accuracy = np.mean(cv_scores) print(f"\nAverage Cross-Validation
Accuracy: {average_cv_accuracy*100:.2f}%") np.random.seed(42)
noisy_features = features + np.random.normal(0, 0.5, size=features.shape) classifier =
RandomForestClassifier() kf = KFold(n_splits=5, shuffle=True, random_state=42)
accuracy_scores_noisy = cross_val_score(classifier, noisy_features, labels, scoring='accuracy', cv=kf)
print("Cross-Validation Accuracy Scores with Noisy Features:", accuracy_scores_noisy) print("Mean
Accuracy with Noisy Features:", accuracy_scores_noisy.mean()) data = {
'Model':['XGB','RF','KNN','SVM','NB','DT'],
  'Accuracy': [98.0,93.1,89.8,90.1,81.1,87.93],
  'Precision': [94.9, 92.7,91.3,95.4,80.3,88.3],
  'F1_Score': [92.9, 91.8, 92.4, 88.8,81.21,86.2],
  'Recall': [93,96.5,92.8,92.5,93.0,84.23]
}
dtfm= pd.DataFrame(data)
print(dtfm) plt.figure(figsize=(12, 8)) plt.subplot(2, 2, 1)
sns.barplot(x=dtfm['Model'], y=dtfm['Accuracy'], color='blue')
plt.title('Model Accuracy') plt.subplot(2, 2, 2)
sns.barplot(x=dtfm['Model'], y=dtfm['Precision'], color='lightgreen')
plt.title('Model Precision') plt.subplot(2, 2, 3)
sns.barplot(x=dtfm['Model'], y=dtfm['Recall'], color='purple')
plt.title('Model Recall') plt.subplot(2, 2, 4)
```

```
sns.barplot(x=dtfm['Model'], y=dtfm['F1_Score'], color='skyblue')

plt.title('Model F1 Score') plt.tight_layout() plt.show() importpandas

as pd import matplotlib.pyplot as plt import numpy as np data_before

= {

  'Model': ['XGBOOST', 'RANDOM FOREST','K-NEAREST NEIGHBOUR','SUPPORT VECTOR MACHINE','NAIVE BAYES','DECISION TREE'],

  'Accuracy': [98.0,93.1,89.8,90.1,81.1,87.93],

  'Precision': [94.9, 92.7,91.3,95.4,80.3,88.3],

  'F1_Score': [92.9, 91.8, 92.4, 88.8,81.21,86.2],

  'Recall': [93,96.5,92.8,92.5,93.0,84.23]

} data_after =

{

  'Model': ['XGBOOST', 'RANDOM FOREST','K-NEAREST NEIGHBOUR','SUPPORT VECTOR MACHINE','NAIVE BAYES','DECISION TREE',],

  'Accuracy': XGB_acc,

  'Precision': KNN_acc,

  'F1_Score': SVM_acc,

  'Recall': NB_acc

}

dtfm_before = pd.DataFrame(data_before)

dtfm_after = pd.DataFrame(data_after)

dtfm_before.set_index('Model', inplace=True)

dtfm_after.set_index('Model', inplace=True)

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(30, 20)) def plot_bargraph(ax, metric, title): before_values = dtfm_before[metric] after_values = dtfm_after[metric] models = dtfm_before.index index = np.arange(len(models)) bar_width = 0.35 ax.bar(index, before_values, width=bar_width, label='Before', color='skyblue') ax.bar(index +
```

```
bar_width, after_values, width=bar_width, label='After', color='lightcoral')     for i in
range(len(models)):
      ax.text(index[i] - 0.05, before_values[i] + 1, f'{before_values[i]:.1f}', fontsize=20)     ax.text(index[i] +
bar_width - 0.05, after_values[i] + 1, f'{after_values[i]:.1f}', fontsize=20)        ax.set_xlabel('Model')
ax.set_ylabel(metric)
      ax.set_title(title)
      ax.set_xticks(index + bar_width / 2)
ax.set_xticklabels(models)
ax.legend()
plot_bargraph(axes[0, 0], 'Accuracy', 'Accuracy Comparison')
plot_bargraph(axes[0, 1], 'Precision', 'Precision Comparison')
plot_bargraph(axes[1, 0], 'F1_Score', 'F1 Score Comparison')
plot_bargraph(axes[1, 1], 'Recall', 'Recall Comparison')
plt.tight_layout() plt.show()
model_compare.T.plot(kind='bar') data
%pip install pickle import pickle
with open("water.pkl","wb") as file:
pickle.dump(dectree,file) from flask import
Flask, render_template, request import pickle
import numpy as np app = Flask(_name_)
model = pickle.load(open("water.pkl", "rb"))
@app.route('/') def
result():
   return render_template("front.html")
@app.route('/prediction',   methods=['POST',   'GET'])   def   prediction():if
request.method == 'POST':         features = [float(x) for x in request.form.values()] #
Convert to float instead of int          final = [np.array(features)]          prediction =
```

```
model.predict_proba(final)                output = '{0:.{1}f}'.format(prediction[0][1], 2)

output_float = float(output)

 if output_float > 0.5:

          return render_template("front.html", pred="Water is safe to drink. Potability:
{}".format(output_float))

       else:

          return render_template("front.html", pred="Water is unsafe. Do not drink. Potability:
{}".format(output_float))

   else:

       return render_template("front.html", pred="Invalid request method.") if

__name__== '_main_':

   app.run(debug=True,host="0.0.0.0",port=7000)

# Use an official Python runtime as a parent image

FROM python:3.12-slim

# Set the working directory in the container

WORKDIR /app

# Copy the current directory contents into the container at /app

COPY . /app

# Install any needed packages specified in requirements.txt

RUN pip install --no-cache-dir -r requirements.txt

# Make port 80 available to the world outside this container

EXPOSE 7000

# Define environment variable

ENV NAME World

# Run app.py when the container launches

CMD ["python", "app.py",7000]
```