```
from sklearn.metrics import accuracy_score, classification_report

# Split the data

# Define the model

xgb = XGBClassifier()

# Expand the parameter grid

param_grid = {

'n_estimators': [10,30,50,80, 100,150, 200],  # Number of gradient boosted trees

'learning_rate': [0.01, 0.03,0.1, 0.2,0.5],  # Step size shrinkage used in update to prevent overfitting

'max_depth': [3, 6,7,8, 9]  # Maximum depth of a tree}

# Function to perform grid search

def perform_grid_search(X_train, y_train, param_name, param_range):

grid_search = GridSearchCV(estimator=xgb, param_grid={param_name: param_range},

cv=3, scoring='accuracy', verbose=2, n_jobs=-1)

grid_search.fit(X_train, y_train)

return grid_search.cv_results_['mean_test_score'], grid_search.cv_results_['std_test_score'], param_range

# Plot function

def plot_validation_curve(scores, stds, param_range, param_name, title):

plt.figure(figsize=(6, 2))

plt.plot(param_range, scores, label='CV Average Score', color='b')

plt.fill_between(param_range, scores - stds, scores + stds, color='blue', alpha=0.2)

plt.title(title)

plt.xlabel(param_name)

plt.ylabel('Accuracy')

plt.legend()

# Perform grid search and plot for each parameter

for param_name, param_range in param_grid.items():

scores, stds, param_range = perform_grid_search(X_train, y_train, param_name, param_range)

plot_validation_curve(scores, stds, param_range, param_name, f'Effect of {param_name} on Model Accuracy')
```

```
# Evaluate the final model (optional here as focus is on tuning)
predictions = grid_search_xgb.predict(X_test)
print("Accuracy on test set: {:.2f}%".format(accuracy_score(y_test, predictions) * 100))
print("Classification Report:\n", classification_report(y_test, predictions))
#XGBOOST
X = data.drop('Potability', axis=1)  # Adjust the column name as necessary
# Define the parameter grid
'n_estimators': [100],
'learning_rate': [0.03],
'max_depth': [50]}
# Set up GridSearchCV
grid_search_xgb = GridSearchCV(estimator=xgb, param_grid=param_grid, cv=5, scoring='accuracy', verbose=2, n_jobs=-1)
grid_search_xgb.fit(X_train, y_train)
# Best parameters and best score
print("Best parameters found: ", grid_search_xgb.best_params_)
print("Best cross-validation score: {:.2f}".format(grid_search_xgb.best_score_))
# Evaluate the model
accuracy = accuracy_score(y_test, predictions) * 100
rmse = np.sqrt(mean_squared_error(y_test, predictions))
mae = mean_absolute_error(y_test, predictions)
f1 = f1_score(y_test, predictions, average='weighted')
# Metrics output
print("Accuracy on test set: {:.2f}%".format(accuracy))
print("RMSE: {:.2f}".format(rmse))
print("MAE: {:.2f}".format(mae))
print("F1 Score: {:.2f}".format(f1))
# Confusion Matrix
sns.heatmap(confusion_matrix(y_test, predictions), annot=True, fmt="d", cmap="Blues")
```

```
train_sizes, train_scores, test_scores = learning_curve(xgb, X, y, cv=5, scoring='accuracy', n_jobs=-1, train_sizes=np.linspace(0.1, 1.0, 10))

plt.title('Learning Curve for XGBoost')

#Decision tree classifer

from sklearn.model_selection import GridSearchCV, train_test_split

# Load and split your dataset here (X_train, X_test, y_train, y_test)

# For this example, let's assume the dataset is already loaded and split

# Parameters to test in grid search

depths = np.arange(1, 21)  # Varying depths from 1 to 20

splits = np.arange(2, 102, 10)  # Varying min_samples_split from 2 to 100

leaves = np.arange(1, 101, 10)  # Varying min_samples_leaf from 1 to 100

dt = DecisionTreeClassifier()

grid_search = GridSearchCV(estimator=dt, param_grid={param_name: param_range},

cv=3, scoring='accuracy', n_jobs=-1, verbose=2)

# Perform grid search and plot for max_depth

scores, stds, param_range = perform_grid_search(X_train, y_train, 'max_depth', depths)

plot_validation_curve(scores, stds, param_range, 'max_depth', 'Effect of Max Depth on Model Accuracy')# Perform grid search and plot for min_samples_split

scores, stds, param_range = perform_grid_search(X_train, y_train, 'min_samples_split', splits)

plot_validation_curve(scores, stds, param_range, 'min_samples_split', 'Effect of Min Samples Split on Model Accuracy')

# Perform grid search and plot for min_samples_leaf

scores, stds, param_range = perform_grid_search(X_train, y_train, 'min_samples_leaf', leaves)

plot_validation_curve(scores, stds, param_range, 'min_samples_leaf', 'Effect of Min Samples Leaf on Model Accuracy')

#Decision tree

# Scaling features (optional for Decision Trees but included for consistency across models)

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

# Define the Decision Tree model and hyperparameter grid

'max_depth': [11],
```

```
'min_samples_split': [3],

'min_samples_leaf': [150]}

grid_search_dt = GridSearchCV(estimator=dt, param_grid=param_grid, cv=5, scoring='accuracy', verbose=2, n_jobs=-1)

grid_search_dt.fit(X_train_scaled, y_train)

print("Best parameters found: ", grid_search_dt.best_params_)

print("Best cross-validation score: {:.2f}".format(grid_search_dt.best_score_))

predictions = grid_search_dt.predict(X_test_scaled)

accuracy = accuracy_score(y_test, predictions)

variance = np.var(predictions)

metrics_data = {

'Metric': ['RMSE', 'MAE', 'F1 Score', 'Variance', 'CV Score', 'Accuracy'],

'Value': [rmse, mae, f1, variance, grid_search_dt.best_score_, accuracy]}

metrics_df = pd.DataFrame(metrics_data)

print(metrics_df)

train_sizes, train_scores, test_scores = learning_curve(dt, X_train_scaled, y_train, cv=5, scoring='accuracy', n_jobs=-1, train_sizes=np.linspace(0.1, 1.0, 10))

plt.title('Learning Curve for Decision Tree')

#SVM

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score

X = data.drop('Potability', axis=1)  # Replace 'Potability' with your target column name

# Scale the features

svm_model = SVC(kernel='rbf')

'C': [0.1, 1, 10, 100],  # Regularization parameter

'gamma': ['scale', 'auto', 0.01, 0.1, 1, 10]  # Kernel coefficient}

grid_search_svm = GridSearchCV(estimator=svm_model, param_grid=param_grid, cv=3, scoring='accuracy', verbose=2, n_jobs=-1)

grid_search_svm.fit(X_train_scaled, y_train)

# Extract information from the grid search

results = pd.DataFrame(grid_search_svm.cv_results_)
```

```
scores = np.array(results.mean_test_score).reshape(len(param_grid['C']), len(param_grid['gamma']))
# Plotting each 'C' value's effect for each 'gamma'
for ind, value in enumerate(param_grid['C']):
plt.plot(param_grid['gamma'], scores[ind], label='C={}'.format(value))
plt.xlabel('Gamma')
plt.ylabel('Mean Test Score')
plt.title('Validation Curve for C and Gamma')
plt.xscale('log')  # This scale is often more useful for the 'gamma' parameter
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Scaling features
'C': [0.1, 1, 10, 100],
'gamma': ['scale', 'auto', 0.01, 0.1, 1, 10]}
print("Best parameters found: ", grid_search_svm.best_params_)
print("Best cross-validation score: {:.2f}".format(grid_search_svm.best_score_))
predictions = grid_search_svm.predict(X_test_scaled)
cv_scores = cross_val_score(svm_model, X, y, cv=5)
# Print metrics
print(f"Train Accuracy: {grid_search_svm.best_score_:.2f}%")
print(f"Test Accuracy: {accuracy:.2f}%")
print(f"RMSE: {rmse}")
print(f"MAE: {mae}")
print(f"F1 Score: {f1}")
print(f"Variance: {variance}")
print(f"CV Score: {np.mean(cv_scores):.2f}")
# Plot learning curve
train_sizes, train_scores, test_scores = learning_curve(svm_model, X, y, cv=5, scoring='accuracy', n_jobs=-1, train_sizes=np.linspace(0.1, 1.0, 10))
plt.title('Learning Curve for SVM')
#neuralnetwork
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, mean_squared_error, mean_absolute_error

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras.optimizers import Adam

# Define the Neural Network model

model = Sequential([

Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)),

Dropout(0.2),

Dense(64, activation='relu'),

Dense(1, activation='sigmoid')])

# Compile the model

optimizer = Adam(learning_rate=0.003)

model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(X_train_scaled, y_train, validation_split=0.2, epochs=150, batch_size=10, verbose=1)

predictions = model.predict(X_test_scaled)

predictions = (predictions > 0.5).astype(int)

'Metric': ['RMSE', 'MAE', 'F1 Score', 'Variance', 'Accuracy'],

'Value': [rmse, mae, f1, variance, accuracy]}

cm = confusion_matrix(y_test, predictions)

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")

# Plot training & validation accuracy values

plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'])

plt.plot(history.history['val_accuracy'])

plt.title('Model accuracy')

plt.xlabel('Epoch')
```

```
plt.legend(['Train', 'Test'], loc='upper left')

# Plot training & validation loss values

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'])

plt.plot(history.history['val_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

#KNN

# Define the KNN model and hyperparameter grid

knn = KNeighborsClassifier()

'n_neighbors': range(1, 31, 2),

'weights': ['uniform', 'distance'],

'metric': ['euclidean', 'manhattan']}

grid_search_knn = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5, scoring='accuracy', verbose=2, n_jobs=-1)

grid_search_knn.fit(X_train_scaled, y_train)

print("Best parameters found: ", grid_search_knn.best_params_)

print("Best cross-validation score: {:.2f}".format(grid_search_knn.best_score_))

predictions = grid_search_knn.predict(X_test_scaled)

train_sizes, train_scores, test_scores = learning_curve(knn, X_train_scaled, y_train, cv=5, scoring='accuracy', n_jobs=-1, train_sizes=np.linspace(0.1, 1.0, 10))

plt.title('Learning Curve for KNN')

#random forest

# Scaling features (Optional for Random Forest but included for consistency)

# Define the Random Forest model and hyperparameter grid

rf = RandomForestClassifier()

'max_depth': [3,6,10],

'min_samples_split': [2],

'min_samples_leaf': [1, 4]}

grid_search_rf = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, scoring='accuracy', verbose=2, n_jobs=-1)
```

```
grid_search_rf.fit(X_train_scaled, y_train)

print("Best parameters found: ", grid_search_rf.best_params_)

print("Best cross-validation score: {:.2f}".format(grid_search_rf.best_score_))

predictions = grid_search_rf.predict(X_test_scaled)

'Value': [rmse, mae, f1, variance, grid_search_rf.best_score_, accuracy]

train_sizes, train_scores, test_scores = learning_curve(rf, X_train_scaled, y_train, cv=5, scoring='accuracy', n_jobs=-1, train_sizes=np.linspace(0.1, 1.0, 10))

plt.title('Learning Curve for Random Forest')

#feature importance

# Define feature names for later use

feature_names = X.columns.tolist()

Dense(1, activation='sigmoid')])

model.fit(X_train_scaled, y_train, epochs=50, batch_size=10, verbose=0)

# Function to calculate importance for each feature

def calculate_feature_importance(X, model, feature_names):

base_accuracy = accuracy_score(y_test, (model.predict(X_test_scaled) > 0.5).astype(int))

importances = []

for i in range(X.shape[1]):

X_new = X.copy()

np.random.shuffle(X_new[:, i])  # Shuffle individual feature

new_accuracy = accuracy_score(y_test, (model.predict(X_new) > 0.5).astype(int))

importance = base_accuracy - new_accuracy

importances.append(importance)

# Create a DataFrame for visualization

importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})

importance_df = importance_df.sort_values(by='Importance', ascending=False)

return importance_df

# Calculate feature importances

importance_df = calculate_feature_importance(X_test_scaled, model, feature_names)

# Plotting feature importances
```

```
plt.figure(figsize=(10, 6))

plt.bar(importance_df['Feature'], importance_df['Importance'], color='b')

plt.ylabel('Importance (Drop in Accuracy)')

plt.title('Feature Importance')

plt.show()

#tkinter application

from sklearn.preprocessing import StandardScaler, MinMaxScaler, PowerTransformer

from tkinter import ttk, messagebox

water = pd.read_csv("water_potability.csv")

labels = ['pH', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity', 'Organic Carbon', 'Trihalomethanes', 'Turbidity']

parameter_ranges = {'pH': (0, 14), 'Hardness': (0, 500), 'Solids': (0, 50000), 'Chloramines': (0, 10), 'Sulfate': (0, 600), 'Conductivity': (0, 1000), 'Organic Carbon': (0, 100), 'Trihalomethanes': (0, 200), 'Turbidity': (0, 10)}

# Function to check if the input value is within the specified range

def validate_input(text, param_range):

try: value = float(text)

return param_range[0] <= value <= param_range[1]

except ValueError:

return False

# Data Handling

X = water.drop("Potability",axis =1)

y = water.Potability

X_train , X_test , y_train , y_test = train_test_split(X , y, test_size=0.2, random_state=42)

# Define machine learning models

models = {

'Linear Regression': LinearRegression(),

'SVM': svm.SVC(C=1, gamma='scale'),

'KNN': KNeighborsClassifier(n_neighbors=3, p=1, metric='cosine'),

'AdaBoost': AdaBoostClassifier(),

'XGBoost': xgb.XGBClassifier(),
```

```
'Decision Tree': DecisionTreeClassifier(),
'Random Forest': RandomForestClassifier(n_estimators=100, criterion='entropy'),
'Neural Network': Sequential([
Dense(12, activation='relu', input_dim=X_train.shape[1]),
Dense(8, activation='relu'),
Dense(1, activation='sigmoid')])}
# Compile the Neural Network
models['Neural Network'].compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
# Train models
for name, model in models.items():
if name == 'Neural Network':
model.fit(X_train, y_train, epochs=10, verbose=0)  # Training neural network
else: model.fit(X_train, y_train)
def predict(inputs, model_name):
inputs = np.array(inputs).reshape(1, -1)
model = models[model_name]
if model_name == 'Neural Network':
probabilities = model.predict(inputs)[0]
predicted_class = (probabilities > 0.5).astype(int)
confidence = probabilities[0]
predicted_class = model.predict(inputs)[0]
probabilities = model.predict_proba(inputs)[0] if hasattr(model, 'predict_proba') else [1]
confidence = probabilities[int(predicted_class)]
return predicted_class, confidence
import tkinter as tk
def on_predict():
inputs = [float(entry.get()) for entry in entries]
model_name = model_var.get()
predicted_class, confidence = predict(inputs, model_name)
```

```
result_text = f"The water is {'potable' if predicted_class == 1 else 'not potable'}"

messagebox.showinfo("Prediction Result", result_text)

root = tk.Tk()

root.title("Water Potability Prediction")

labels = ['pH(1,12)', 'Hardness(120,270)', 'Solids(10,40000)', 'Chloramines(3,11)', 'Sulfate(270,400)', 'Conductivity(200,800)', 'Organic Carbon(5,25)', 'Trihalomethanes(30,110)', 'Turbidity(2,6)']

entries = []

for i, label in enumerate(labels):

tk.Label(root, text=label).grid(row=i, column=0)

entry = tk.Entry(root)

entry.grid(row=i, column=1)

entries.append(entry)

model_var = tk.StringVar(root)

model_var.set("SVM")  # default value

tk.Label(root, text="Select Model:").grid(row=10, column=0)

model_selector = ttk.Combobox(root, textvariable=model_var, values=list(models.keys()))

model_selector.grid(row=10, column=1)

predict_button = tk.Button(root, text="Predict", command=on_predict)

predict_button.grid(row=11, column=0, columnspan=2)

root.mainloop()
```

Appendices B

WATER POTABILITY ASSESSMENT USING MACHINE LEARNING TECHNIQUES

DIGITAL TECHNOLOGY FOR AGRICULTURE AND NATURAL RESOURCES

702706968

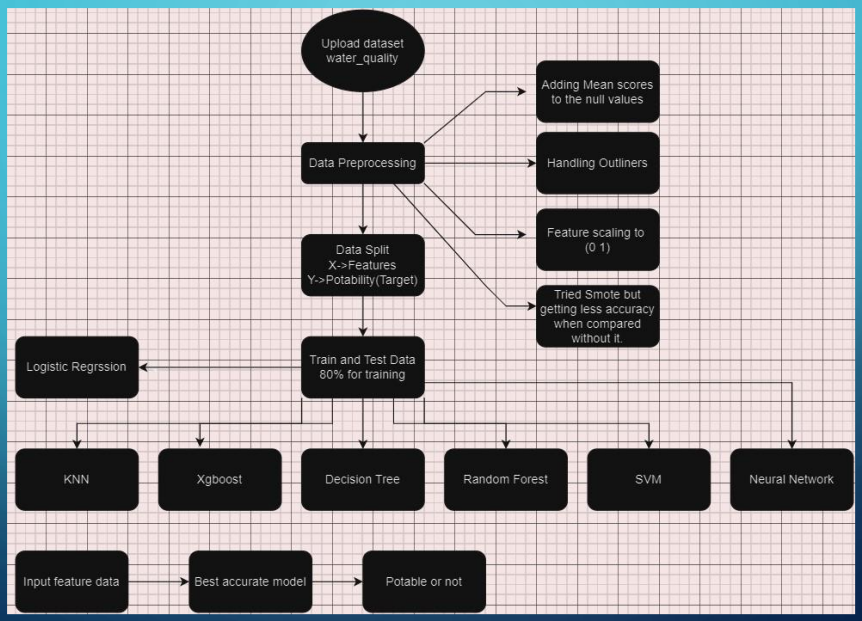
PRATHYUSH KVSS

UNIVERSITY OF DELAWARE

- **Problem Statement**:

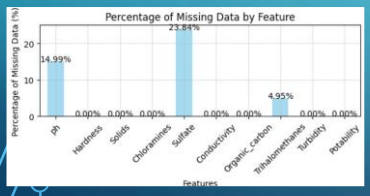
Access to safe and potable water is a necessity for human health and well-being. However, many regions face contaminated or quality-compromised water sources, posing public health risks. There is a need for an automated, scalable approach leveraging sensor data and machine learning techniques to classify water quality and determine potability.

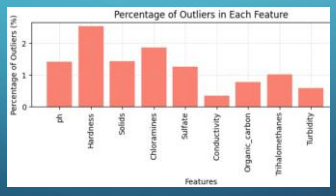
OVER ALL WORKFLOW

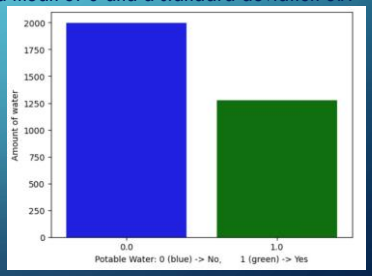


DATA PREPROCESSING

- **Analyze Data:** Features like 'ph' , 'Sulphate' and 'Organic_carbon' show higher counts of missing data, while all other shows no missing entries.
- **Handling missing values:** Fill missing values in specific columns with the mean of their nonnull entries.
- **Handling Outliners:** Apply Winsorization to the water dataset by capping outliers below the 5th percentile and above the 85th percentile as threshold values.
- **Featured Scaling:** Feature scaling is crucial for this dataset to standardize variables like Solids, sulphates which vary widely in their ranges and units. I rescaled features with standardization, so that they have a mean of 0 and a standard deviation of1.
- Have got score 39% of data showing potable water, data has output imbalance.







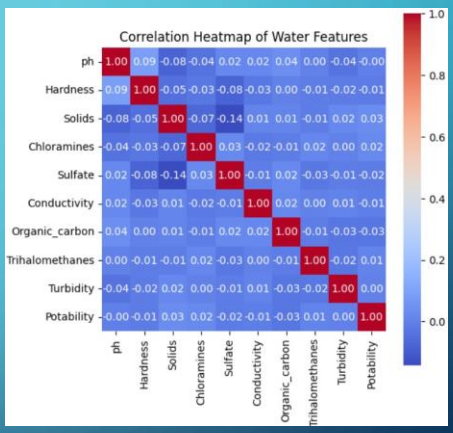
- **Correlation:**

Most of the cells show values close to zero, indicating little to no linear correlation between the features.

For example, the correlation between pH and Hardness is 0.09, suggesting a very weak positive linear relationship, Sulphates and Solids have a correlation coefficient of -0.14, indicating a negligible negative relationship.

Co-relation matrix says that there is no linear relationship between the features that can explain the target variable.

So, Linear model may not work on this problem. we need to try with probability based models.



MACHINE LEARNING MODELS

Considering the following models for the best accuracy of potability.

Linear Model

- Logistic Regression

Non Linear Model

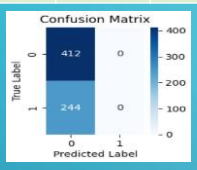
- Xgboost
- Random Forest
- KNN
- Decision Tree
- SVM
- Neural network

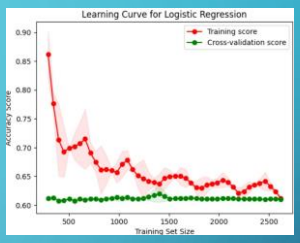
MODEL TRAINING AND VALIDATION

- X = water.drop(['Potability'],axis=1)
- Y = water['Potability']
- Split data train 80% and test 20%
- Use different algorithms with their respective parameters and tune them for better accuracy.

- Logistic Regression

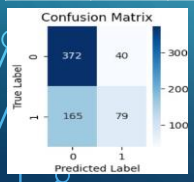
Model	Size	Rmse	Mae	F1 Score	Variance	CV Score	Accuracy
Logist	80:20	0.609	0.372	0.48	0.1	0.61	0.62

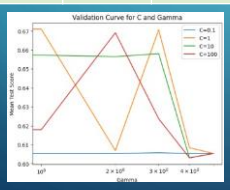


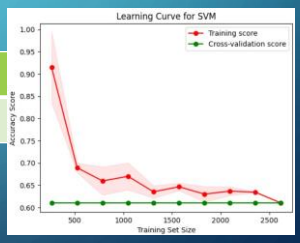


- SVM (Tuned parameters: C =1,gamma ='scale')

Model	Size	Rmse	Mae	F1 Score	Variance	CV Score	Accuracy
SVM	80:20	0.559	0.3125	0.6542	0.14	0.61	0.67

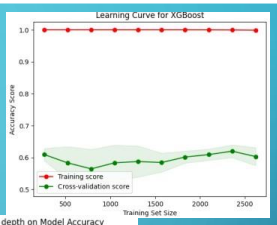


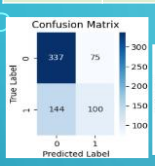


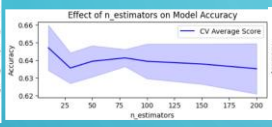


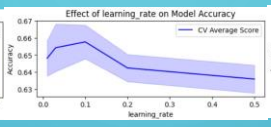
- Xgboost (tuning no.of estimators, max_depth,learning rate)

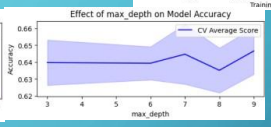
Model	Size	Rmse	Mae	F1 Score	Variance	CV Score	Accuracy
Xgboost	80:20	0.58	0.63	0.65	0.11	0.65	0.66





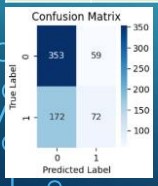


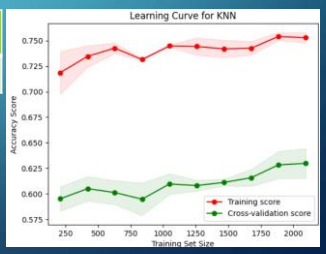




- KNN (metric: manhattan,n_neighbors: 15,weights: distance)

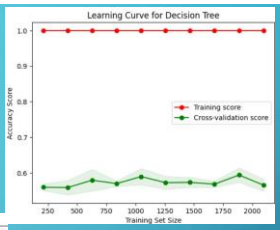
Model	Size	Rmse	Mae	F1 Score	Variance	CV Score	Accuracy
KNN	80:20	0.59	0.35	0.61	0.15	0.63	0.64

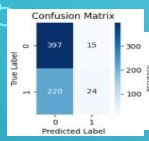


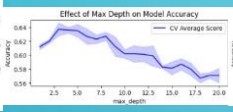


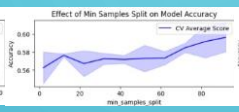
- Decision Tree (max_depth,min_samples_leaf,min_split_samples)

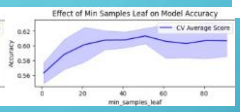
Model	Size	Rmse	Mae	F1 Score	Variance	CV Score	Accuracy
DecisionT	80:20	0.59	0.35	0.54	0.05	0.62	0.64





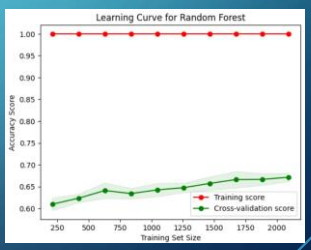


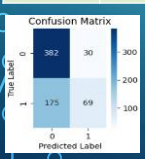




- Random Forest (estimators, max_depth,min_leaf_split)

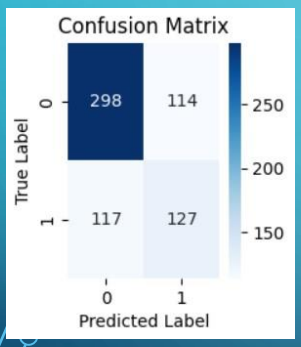
Model	Size	Rmse	Mae	F1 Score	Variance	CV Score	Accuracy
Rand F	80:20	0.559	0.312	0.64	0.12	0.67	0.68

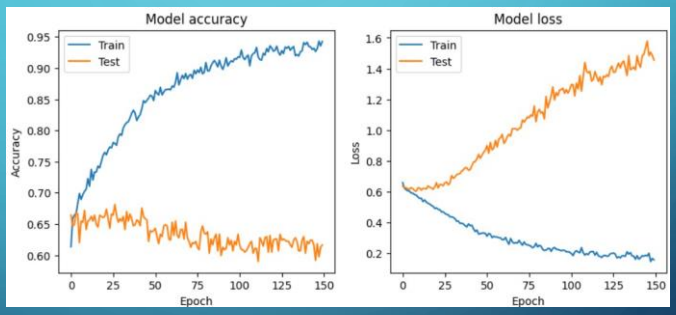




- Neural Network

Model	Size	Rmse	Mae	F1 Score	Variance	CV Score	Accuracy
Neural network	80:20	0.59	0.35	0.64	0.23	0.66	0.68





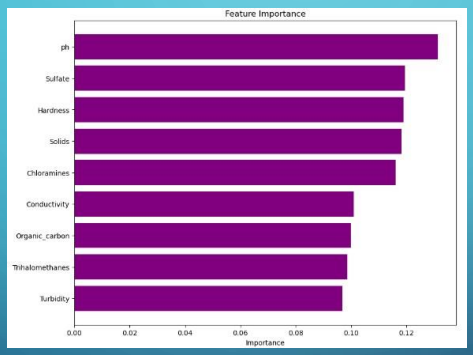
COMPARE MODELS

Model	Size	Rmse	Mae	F1 Score(P&R)	Variance	CV Score	Accuracy
Logistic Regression	80:20	0.609	0.372	0.48	0.01	0.61	0.62
SVM	80:20	0.559	0.3125	0.6542	0.14	0.61	0.67
Xgboost	80:20	0.58	0.63	0.65	0.11	0.65	0.66
KNN	80:20	0.59	0.35	0.61	0.15	0.63	0.64
Decision Tree	80:20	0.59	0.35	0.54	0.05	0.62	0.64
Random Forest	80:20	0.56	0.312	0.64	0.12	0.67	0.68
Neural network	80:20	0.59	0.35	0.64	0.23	0.66	0.67

FEATURE IMPORTANCE

Feature importance is done with the best model.(Random Forest)

Results:

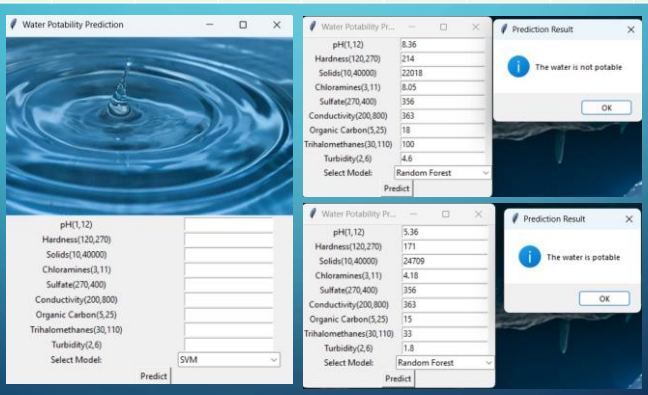


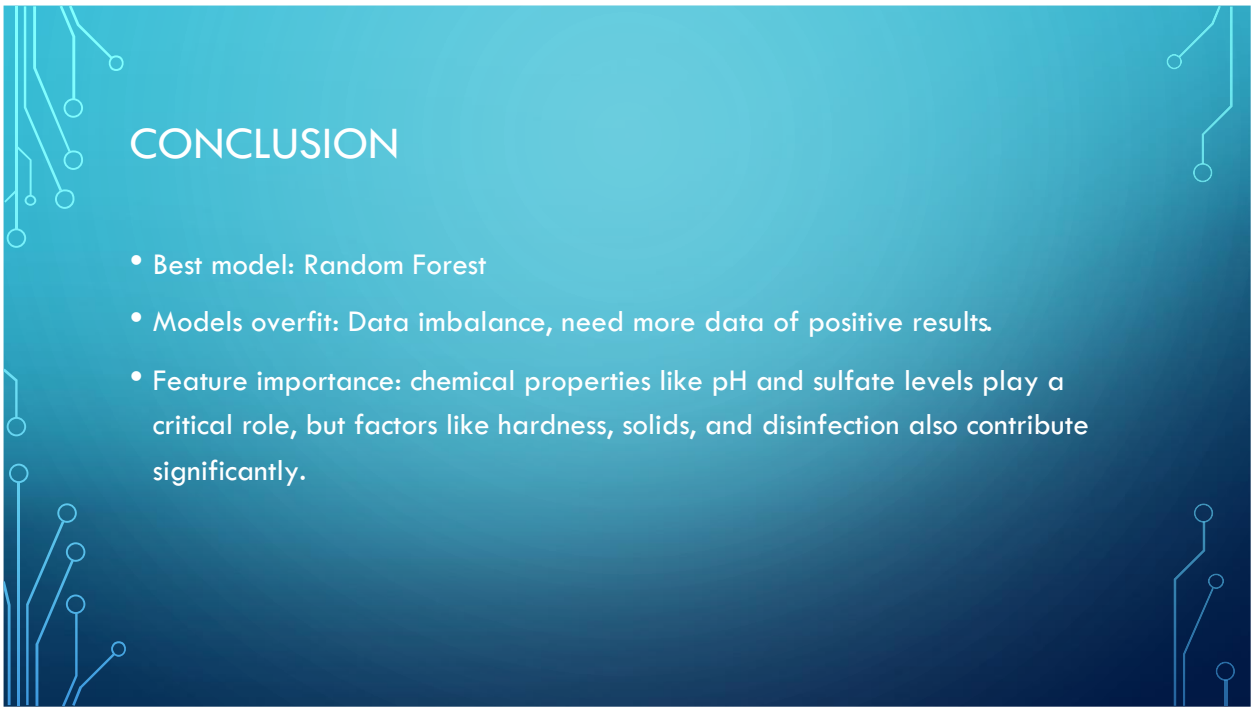
Found out that PH is more effecting the output parameter Potability.

APPLICATION

- Tkinter

ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
5.361211666	171.315543	24709.72886	4.184890692	366.8149789	490.658747	15.12267225	33.1196126	1.81252894	1
8.316765884	214.3733941	22018.41744	8.059332377	356.8861356	363.2665162	18.4365245	100.3416744	4.628770537	0





Appendics C

References

- Dataset source from Kaggle

(https://www.kaggle.com/datasets/adityakadiwal/water-potability/data)

- Exploring the Application of Machine Learning Algorithms to Water Quality Analysis.(https://ieeexplore.ieee.org/document/9900636)

- Predicting and analyzing water quality using Machine Learning: A comprehensive model Publisher: IEEE Authors: Yafra Khan; Chai Soo See (https://ieeexplore.ieee.org/abstract/document/7494106)

- Efficient Drinking Water Quality Analysis using Machine Learning Model with Hyper-Parameter Tuning Publisher: IEEE Authors: M Uma Maheswari; R. Sudharsanan; M Arthy; Anne Jenefer; L Oormila; V Samuthira Pandi (https://ieeexplore.ieee.org/document/10142799)

- An Artificial Neural Network Model for Water Quality and Water Consumption Prediction Author: Furqan Rustam Abid Ishaq (https://www.researchgate.net/publication/364701220_An_Artificial_Neural_Network_Model_for_Water_Quality_and_Water_Consumption_Prediction)

- J. Brzezinski, "Logistic Regression Modeling for Context-Based Classification", 2012 23rd International Workshop on Database and Expert Systems Applications, pp. 755, 1999. (Logistic regression modeling for context-based classification | IEEE Conference Publication | IEEE Xplore)

- P. Cunningham and S. J. Delany, "k-Nearest Neighbor Classifiers - A Tutorial", ACM Computing Surveys, vol. 54, no. 6, pp. 1-25, Jul. 2021. (k-Nearest Neighbour Classifiers - A Tutorial | ACM Computing Surveys)