**Water Quality Analysis**

**Abstract:**

Water quality analysis is a critical component of environmental management and public health protection. This study presents a comprehensive assessment of water quality in the [Location] watershed, aiming to understand the spatial and temporal variations, identify potential sources of contamination, and evaluate the overall health of aquatic ecosystems.

To achieve this goal, water samples were collected at multiple sites across the watershed over a period of [duration]. A range of physical, chemical, and biological parameters were measured, including pH, turbidity, dissolved oxygen, temperature, nutrients (nitrogen and phosphorus), heavy metals, fecal coliforms, and various aquatic macroinvertebrate populations.

**DATA COLLECTION:**

Data have been taken from a trustworthy sources

DATALINK: https://www.kaggle.com/datasets/adityakadiwal/water-potability

**The results of this analysis reveal several key findings:**

1. Spatial Variability: Water quality parameters exhibited spatial variations across the watershed, indicating localized sources of pollution and ecological differences.
2. Seasonal Trends: Seasonal patterns in water quality were observed, with higher nutrient concentrations and turbidity during [season] and improved conditions during [season].
3. Pollution Sources: Identifiable pollution sources, such as agricultural runoff, industrial discharges, and urban development, were identified as contributing to water quality degradation in specific areas.
4. Ecological Health: The health of aquatic ecosystems was assessed using biological indicators, showing varying levels of impairment in different sections of the watershed.
5. Management Implications: The study’s findings have important implications for water resource management, highlighting the need for targeted pollution control measures and restoration efforts in identified problem areas.

This comprehensive water quality analysis provides essential data for local authorities, environmental agencies, and stakeholders to make informed decisions regarding watershed management and protection. Further research and collaborative efforts are recommended to address specific pollution sources and restore the overall health of the watershed.

**SOURCE CODE:**

Import numpy as np # linear algebra

Import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

Import seaborn as sns

Import matplotlib.pyplot as plt

From scipy.stats import shapiro

From sklearn.ensemble import IsolationForest

From sklearn.metrics import roc\_auc\_score, roc\_curve, accuracy\_score, f1\_score

From sklearn.inspection import permutation\_importance

From sklearn.linear\_model import LogisticRegression

From sklearn.model\_selection import train\_test\_split

From sklearn.svm import SVC, LinearSVC

From sklearn.pipeline import Pipeline, make\_pipeline

From sklearn.preprocessing import Normalizer, StandardScaler

From sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, StackingClassifier

From sklearn.naive\_bayes import GaussianNB

From sklearn.neighbors import KNeighborsClassifier

From sklearn.discriminant\_analysis import QuadraticDiscriminantAnalysis

From sklearn.metrics import confusion\_matrix

From sklearn.metrics import accuracy\_score, f1\_score

From imblearn.over\_sampling import SMOTE

From sklearn.model\_selection import cross\_val\_score, cross\_val\_predict

From sklearn.tree import DecisionTreeClassifier

From sklearn.model\_selection import GridSearchCV

From sklearn.ensemble import RandomForestClassifier

From sklearn.metrics import make\_scorer, f1\_score

# Input data files are available in the read-only “../input/” directory

# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

Import os

For dirname, \_, filenames in os.walk(‘/kaggle/input’):

For filename in filenames:

Print(os.path.join(dirname, filename))

Def show\_distributions(columns: list, data: pd.DataFrame, nrows: int = 1, ncols: int = 3):

# This function creates distribution subplots.

Fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(15, 5))

Axes = axes.ravel()

For index, column in enumerate(columns):

Sns.histplot(data[column], kde=True, ax=axes[index])

Axes[index].set\_title(column)

# Adjust layout

Plt.tight\_layout()

Plt.show()

Def show\_boxplots(columns: list, data: pd.DataFrame, nrows: int = 3, ncols: int = 3):

# This function creates box plot subplots.

Fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(10, 15))

Axes = axes.ravel()

For index, column in enumerate(columns):

Axes[index].boxplot(data[column])

Axes[index].set\_title(column)

Plt.tight\_layout()

Plt.show()

Def normality\_test(columns: list, data: pd.DataFrame):

# Conducts Shapiro-Wilk test.

For i in columns:

Results = shapiro(data[data[i].isna() == False][i])

Print(i, results.statistic)

Def random\_sample\_imputation(df):

# Random Sample Imputatuin

Cols\_with\_missing\_values = df.columns[df.isna().any()].tolist()

For var in cols\_with\_missing\_values:

Random\_sample\_df = df[var].dropna().sample(df[var].isnull().sum(),

Random\_state=0)

Random\_sample\_df.index = df[

Df[var].isnull()].index

Df.loc[df[var].isnull(), var] = random\_sample\_df

Return df

Def visualize\_isolation\_forest(columns: list, data: pd.DataFrame, contamination: list,nrows: int = 2, ncols: int = 3,):

# Visualize resuls from different contamination values

Sns.set\_style(“darkgrid”)

Fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(15, 5))

Axes = axes.ravel()

For index, i in enumerate(contamination):

Model = IsolationForest(contamination=i) # Adjust the contamination parameter

Model.fit(data.to\_numpy())

Sns.scatterplot(data = data[model.predict(data.to\_numpy()) == -1], x = columns[0], y = columns[1], color=’blue’, label=’Outlier’, ax = axes[index])

Sns.scatterplot(data = data[model.predict(data.to\_numpy()) == 1], x = columns[0], y = columns[1], color=’red’, label=’Normal’, ax = axes[index])

Axes[index].legend()

Axes[index].set\_title(f”Contamination Value: {round(i,3)}”)

Plt.show()

Def remove\_outliers(data: pd.DataFrame, contamination: int):

# Apply Isolation Forest

Model = IsolationForest(contamination = contamination)

Model.fit(data.to\_numpy())

Return data[model.predict(data.to\_numpy()) == 1]

Def roc\_plot(models: list, X\_test: pd.DataFrame, y\_test: pd.DataFrame):

# Create ROC plot for multiple models.]

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

Ns\_probs = [0 for \_ in range(len(X\_test))]

Ns\_auc = roc\_auc\_score(y\_test, ns\_probs)

Ns\_fpr, ns\_tpr, \_ = roc\_curve(y\_test, ns\_probs)

Plt.plot(ns\_fpr, ns\_tpr, linestyle=’—‘, label=’Baseline’)

For model in models:

Lr\_probs = model.predict\_proba(X\_test)

Lr\_probs = lr\_probs[:, 1]

Lr\_auc = roc\_auc\_score(y\_test, lr\_probs)

Lr\_fpr, lr\_tpr, \_ = roc\_curve(y\_test, lr\_probs)

Plt.plot(lr\_fpr, lr\_tpr, marker=’.’, label = model[“classifier”])

Plt.xlabel(‘False Positive Rate’)

Plt.ylabel(‘True Positive Rate’)

Plt.legend()

Plt.show()

Def forest\_feature\_importance(model,X\_test):

# Visualize Random Forest Feature Importance

Feature\_names = list(X\_test.columns)

Importances = model.feature\_importances\_

Std = np.std([tree.feature\_importances\_ for tree in model.estimators\_], axis=0)

Forest\_importances = pd.Series(importances, index=feature\_names)

Fig, ax = plt.subplots()

Forest\_importances.plot.bar(yerr=std, ax=ax)

Ax.set\_title(“Feature importances using MDI”)

Ax.set\_ylabel(“Mean decrease in impurity”)

Fig.tight\_layout()

Def forest\_permutation\_importance(model, X\_test, y\_test, n\_repeats:int = 10):

# Visualize Random Forest Permutation Importance

Feature\_names = list(X\_test.columns)

Importances = model.feature\_importances\_

Importances = model.feature\_importances\_

Std = np.std([tree.feature\_importances\_ for tree in model.estimators\_], axis=0)

Result = permutation\_importance(

Model, X\_test.to\_numpy(), y\_test, n\_repeats=n\_repeats, random\_state=42, n\_jobs=2

)

Forest\_importances = pd.Series(importances, index = feature\_names)

Fig, ax = plt.subplots()

Forest\_importances.plot.bar(yerr=result.importances\_std, ax=ax)

Ax.set\_title(“Feature importances using permutation on full model”)

Ax.set\_ylabel(“Mean accuracy decrease”)

Fig.tight\_layout()

Plt.show()