```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# 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))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

# <center> Water Quality </center>

<center> <img src="https://media.istockphoto.com/photos/analyzing-samples-picture-id182188515?k=6&m=182188515&s=612x612&w=0&h=Hcjly5YZGs4tFxPmD6Q-hbCKcoGFU-JIPT8qLaYDUOQ="> </center>

## About dataset

#### Content

The water_potability.csv file contains water quality metrics for 3276 different water bodies.

1. pH value:

PH is an important parameter in evaluating the acid–base balance of water. It is also the indicator of acidic or alkaline condition of water status.

WHO has recommended maximum permissible limit of pH from 6.5 to 8.5. The current investigation ranges were 6.52–6.83 which are in the range of WHO standards.

2. Hardness:

Hardness is mainly caused by calcium and magnesium salts. These salts are dissolved from geologic deposits through which water travels. The length of time water is in contact with hardness producing material helps determine how much hardness there is in raw water. Hardness was originally defined as the capacity of water to precipitate soap caused by Calcium and Magnesium.

3. Solids (Total dissolved solids - TDS):

Water has the ability to dissolve a wide range of inorganic and some organic minerals or salts such as potassium, calcium, sodium, bicarbonates, chlorides, magnesium, sulfates etc. These minerals produced un-wanted taste and diluted color in appearance of water. This is the important parameter for the use of water. The water with high TDS value indicates that water is highly mineralized. Desirable limit for TDS is 500 mg/l and maximum limit is 1000 mg/l which prescribed for drinking purpose.

4. Chloramines:

Chlorine and chloramine are the major disinfectants used in public water systems. Chloramines are most commonly formed when ammonia is added to chlorine to treat drinking water. Chlorine levels up to 4 milligrams per liter (mg/L or 4 parts per million (ppm)) are considered safe in drinking water.

5. Sulfate:

Sulfates are naturally occurring substances that are found in minerals, soil, and rocks. They are present in ambient air, groundwater, plants, and food. The principal commercial use of sulfate is in the chemical industry. Sulfate concentration in seawater is about 2,700 milligrams per liter (mg/L). It ranges from 3 to 30 mg/L in most freshwater supplies, although much higher concentrations (1000 mg/L) are found in some geographic locations.

6. Conductivity:

Pure water is not a good conductor of electric current rather’s a good insulator. Increase in ions concentration enhances the electrical conductivity of water. Generally, the amount of dissolved solids in water determines the electrical conductivity. Electrical conductivity (EC) actually measures the ionic process of a solution that enables it to transmit current. According to WHO standards, EC value should not exceeded 400 μS/cm.

7. Organic_carbon:

Total Organic Carbon (TOC) in source waters comes from decaying natural organic matter (NOM) as well as synthetic sources. TOC is a measure of the total amount of carbon in organic compounds in pure water. According to US EPA < 2 mg/L as TOC in treated / drinking water, and < 4 mg/Lit in source water which is use for treatment.

8. Trihalomethanes:

THMs are chemicals which may be found in water treated with chlorine. The concentration of THMs in drinking water varies according to the level of organic material in the water, the amount of chlorine required to treat the water, and the temperature of the water that is being treated. THM levels up to 80 ppm is considered safe in drinking water.

9. Turbidity:

The turbidity of water depends on the quantity of solid matter present in the suspended state. It is a measure of light emitting properties of water and the test is used to indicate the quality of waste discharge with respect to colloidal matter. The mean turbidity value obtained for Wondo Genet Campus (0.98 NTU) is lower than the WHO recommended value of 5.00 NTU.

10. Potability:

Indicates if water is safe for human consumption where 1 means Potable and 0 means Not potable.

```
# It is always consider as a good practice to make a copy of original dataset.

main_df = pd.read_csv("/kaggle/input/water-potability/water_potability.csv")
df = main_df.copy()
# Getting top 5 row of the dataset

df.head()
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import plotly.express as px
import warnings
warnings.filterwarnings('ignore')
```

```
### Following are the list of algorithms that are used in this notebook.

|    Algorithm        |
| -------------------- |
| Logistic Regression  |
| Decision Tree|
| Random Forest|
| XGBoost|
| KNeighbours|
| SVM|
| AdaBoost|
print(df.shape)
print(df.columns)
df.describe()
df.info()
print(df.nunique())
print(df.isnull().sum())
df.dtypes
sns.heatmap(df.isnull())
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot= True, cmap='coolwarm')
# Unstacking the correlation matrix to see the values more clearly.
corr = df.corr()
c1 = corr.abs().unstack()
c1.sort_values(ascending = False)[12:24:2]
ax = sns.countplot(x = "Potability",data= df, saturation=0.8)
plt.xticks(ticks=[0, 1], labels = ["Not Potable", "Potable"])
plt.show()
x = df.Potability.value_counts()
labels = [0,1]
print(x)
```

```
sns.violinplot(x='Potability', y='ph', data=df, palette='rocket')
# Visualizing dataset and also checking for outliers


fig, ax = plt.subplots(ncols = 5, nrows = 2, figsize = (20, 10))
index = 0
ax = ax.flatten()


for col, value in df.items():
   sns.boxplot(y=col, data=df, ax=ax[index])
   index += 1
plt.tight_layout(pad = 0.5, w_pad=0.7, h_pad=5.0)
plt.rcParams['figure.figsize'] = [20,10]
df.hist()
plt.show()
sns.pairplot(df, hue="Potability")
plt.rcParams['figure.figsize'] = [7,5]
sns.distplot(df['Potability'])
df.hist(column='ph', by='Potability')
df.hist(column='Hardness', by='Potability')
# Individual box plot for each feature
def Box(df):
   plt.title("Box Plot")
   sns.boxplot(df)
   plt.show()
Box(df['ph'])
sns.histplot(x = "Hardness", data=df)
df.nunique()
skew_val = df.skew().sort_values(ascending=False)
skew_val
```

* Using pandas skew function to check the correlation between the values.

* Values between 0.5 to -0.5 will be considered as the normal distribution else will be skewed depending upon the skewness value.

```
fig = px.box(df, x="Potability", y="ph", color="Potability", width=800, height=400)

fig.show()

fig = px.box(df, x="Potability", y="Hardness", color="Potability", width=800, height=400)

fig.show()

fig = px.histogram (df, x = "Sulfate",  facet_row = "Potability",  template = 'plotly_dark')

fig.show ()

fig = px.histogram (df, x = "Trihalomethanes",  facet_row = "Potability",  template = 'plotly_dark')

fig.show ()

fig =  px.pie (df, names = "Potability", hole = 0.4, template = "plotly_dark")

fig.show ()

fig = px.scatter (df, x = "ph", y = "Sulfate", color = "Potability", template = "plotly_dark",
trendline="ols")

fig.show ()

fig = px.scatter (df, x = "Organic_carbon", y = "Hardness", color = "Potability", template =
"plotly_dark",  trendline="lowess")

fig.show ()

df.isnull().mean().plot.bar(figsize=(10,6))

plt.ylabel('Percentage of missing values')

plt.xlabel('Features')

plt.title('Missing Data in Percentages');

df['ph'] = df['ph'].fillna(df['ph'].mean())

df['Sulfate'] = df['Sulfate'].fillna(df['Sulfate'].mean())

df['Trihalomethanes'] = df['Trihalomethanes'].fillna(df['Trihalomethanes'].mean())

df.head()

sns.heatmap(df.isnull())

df.isnull().sum()

X = df.drop('Potability', axis=1)

y = df['Potability']

X.shape, y.shape

# import StandardScaler to perform scaling
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X = scaler.fit_transform(X)

X

# import train-test split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

## Using Logistic Regression

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

# Creating model object

model_lg = LogisticRegression(max_iter=120,random_state=0, n_jobs=20)

# Training Model

model_lg.fit(X_train, y_train)

# Making Prediction

pred_lg = model_lg.predict(X_test)

# Calculating Accuracy Score

lg = accuracy_score(y_test, pred_lg)

print(lg)

print(classification_report(y_test,pred_lg))

# confusion Maxtrix

cm1 = confusion_matrix(y_test, pred_lg)

sns.heatmap(cm1/np.sum(cm1), annot = True, fmt=  '0.2%', cmap = 'Reds')

## Using Decision Tree Classifier

from sklearn.tree import DecisionTreeClassifier

# Creating model object

model_dt = DecisionTreeClassifier( max_depth=4, random_state=42)

# Training Model

model_dt.fit(X_train,y_train)

# Making Prediction

pred_dt = model_dt.predict(X_test)
```

```
# Calculating Accuracy Score

dt = accuracy_score(y_test, pred_dt)

print(dt)

print(classification_report(y_test,pred_dt))

# confusion Maxtrix

cm2 = confusion_matrix(y_test, pred_dt)

sns.heatmap(cm2/np.sum(cm2), annot = True, fmt=  '0.2%', cmap = 'Reds')

## Using Random Forest

from sklearn.ensemble import RandomForestClassifier

# Creating model object

model_rf = RandomForestClassifier(n_estimators=300,min_samples_leaf=0.16, random_state=42)

# Training Model

model_rf.fit(X_train, y_train)

# Making Prediction

pred_rf = model_rf.predict(X_test)

# Calculating Accuracy Score

rf = accuracy_score(y_test, pred_rf)

print(rf)

print(classification_report(y_test,pred_rf))

# confusion Maxtrix

cm3 = confusion_matrix(y_test, pred_rf)

sns.heatmap(cm3/np.sum(cm3), annot = True, fmt=  '0.2%', cmap = 'Reds')

## Using XGBoost Classifier

from xgboost import XGBClassifier

# Creating model object

model_xgb = XGBClassifier(max_depth= 8, n_estimators= 125, random_state= 0,  learning_rate=
0.03, n_jobs=5)

# Training Model

model_xgb.fit(X_train, y_train)

# Making Prediction

pred_xgb = model_xgb.predict(X_test)
```

```
# Calculating Accuracy Score

xgb = accuracy_score(y_test, pred_xgb)

print(xgb)

print(classification_report(y_test,pred_xgb))

# confusion Maxtrix

cm4 = confusion_matrix(y_test, pred_xgb)

sns.heatmap(cm4/np.sum(cm4), annot = True, fmt=  '0.2%', cmap = 'Reds')

## Using KNeighbours

from sklearn.neighbors import KNeighborsClassifier

# Creating model object

model_kn = KNeighborsClassifier(n_neighbors=9, leaf_size=20)

# Training Model

model_kn.fit(X_train, y_train)

# Making Prediction

pred_kn = model_kn.predict(X_test)

# Calculating Accuracy Score

kn = accuracy_score(y_test, pred_kn)

print(kn)

print(classification_report(y_test,pred_kn))

# confusion Maxtrix

cm5 = confusion_matrix(y_test, pred_kn)

sns.heatmap(cm5/np.sum(cm5), annot = True, fmt=  '0.2%', cmap = 'Reds')

## Using SVM

from sklearn.svm import SVC, LinearSVC

model_svm = SVC(kernel='rbf', random_state = 42)

model_svm.fit(X_train, y_train)

# Making Prediction

pred_svm = model_svm.predict(X_test)

# Calculating Accuracy Score

sv = accuracy_score(y_test, pred_svm)

print(sv)
print(classification_report(y_test,pred_kn))

# confusion Maxtrix

cm6 = confusion_matrix(y_test, pred_svm)

sns.heatmap(cm6/np.sum(cm6), annot = True, fmt=  '0.2%', cmap = 'Reds')

## Using AdaBoost Classifier

from sklearn.ensemble import AdaBoostClassifier

model_ada = AdaBoostClassifier(learning_rate= 0.002,n_estimators= 205,random_state=42)

model_ada.fit(X_train, y_train)

# Making Prediction

pred_ada = model_ada.predict(X_test)

# Calculating Accuracy Score

ada = accuracy_score(y_test, pred_ada)

print(ada)

print(classification_report(y_test,pred_ada))

# confusion Maxtrix

cm7 = confusion_matrix(y_test, pred_ada)

sns.heatmap(cm7/np.sum(cm7), annot = True, fmt=  '0.2%', cmap = 'Reds')

models = pd.DataFrame({

    'Model':['Logistic Regression', 'Decision Tree', 'Random Forest', 'XGBoost', 'KNeighbours', 'SVM', 
'AdaBoost'],

    'Accuracy_score' :[lg, dt, rf, xgb, kn, sv, ada]

})

models

sns.barplot(x='Accuracy_score', y='Model', data=models)


models.sort_values(by='Accuracy_score', ascending=False)
```

#### Conclusion :- Here SVM classifier has achieved highest accuracy.