Safe and readily available water is important for public health, whether it is used for drinking, domestic use, food production or recreational purposes. Improved water supply and sanitation, and better management of water resources, can *boost* countries' economic growth and can contribute greatly to poverty reduction.

Contaminated water and poor sanitation are linked to transmission of diseases such as **cholera, diarrhoea, dysentery, hepatitis A, typhoid, and polio**. Absent, inadequate, or inappropriately managed water and sanitation services expose individuals to preventable health risks. This is particularly the case in health care facilities where both patients and staff are placed at additional risk of infection and disease when water, sanitation, and hygiene services are lacking. Globally, **15%** of patients develop an infection during a hospital stay, with the proportion much greater in low-income countries.

So, I took some inspiration from this to use this **Water Quality** dataset to understand what consitutes to safe, Potable water and apply machine learning to it to distinguish between Potable and Non-Potable water.

---

1. **ph**: pH of 1. water (0 to 14).
2. **Hardness**: Capacity of water to precipitate soap in mg/L.
3. **Solids**: Total dissolved solids in ppm.
4. **Chloramines**: Amount of Chloramines in ppm.
5. **Sulfate**: Amount of Sulfates dissolved in mg/L.
6. **Conductivity**: Electrical conductivity of water in µS/cm.
7. **Organic_carbon**: Amount of organic carbon in ppm.
8. **Trihalomethanes**: Amount of Trihalomethanes in µg/L.
9. **Turbidity**: Measure of light emiting property of water in NTU.
10. **Potability**: Indicates if water is safe for human consumption. Potable - 1 and Not potable - 0

```
# Basic Libraries
import numpy as np
import pandas as pd
from warnings import filterwarnings
from collections import Counter

# Visualizations Libraries
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import plotly
import plotly.offline as pyo
import plotly.express as px
import plotly.graph_objs as go
pyo.init_notebook_mode()
import plotly.figure_factory as ff
import missingno as msno

# Data Pre-processing Libraries
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import train_test_split

# Modelling Libraries
from sklearn.linear_model import
LogisticRegression,RidgeClassifier,SGDClassifier,PassiveAggressiveClas
sifier
from sklearn.linear_model import Perceptron
from sklearn.svm import SVC,LinearSVC,NuSVC
from sklearn.neighbors import KNeighborsClassifier,NearestCentroid
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import
RandomForestClassifier,AdaBoostClassifier,GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB,BernoulliNB
from sklearn.ensemble import VotingClassifier

# Evaluation & CV Libraries
from sklearn.metrics import precision_score,accuracy_score
from sklearn.model_selection import
RandomizedSearchCV,GridSearchCV,RepeatedStratifiedKFold
```

```
colors_blue = ["#132C33", "#264D58", '#17869E', '#51C4D3', '#B4DBE9']
colors_dark = ["#1F1F1F", "#313131", '#636363', '#AEAEAE', '#DADADA']
colors_green = ['#01411C','#4B6F44','#4F7942','#74C365','#D0F0C0']
sns.palplot(colors_blue)
sns.palplot(colors_green)
sns.palplot(colors_dark)
```

```
df=pd.read_csv('../input/water-potability/water_potability.csv')

df.info()

df.head()
```

```
d= pd.DataFrame(df['Potability'].value_counts())
fig = px.pie(d,values='Potability',names=['Not
Potable','Potable'],hole=0.4,opacity=0.6,
             color_discrete_sequence=[colors_green[3],colors_blue[3]],
              labels={'label':'Potability','Potability':'No. Of
Samples'})

fig.add_annotation(text='We can resample the data<br> to get a
balanced dataset',

x=1.2,y=0.9,showarrow=False,font_size=12,opacity=0.7,font_family='mono
space')
fig.add_annotation(text='Potability',

x=0.5,y=0.5,showarrow=False,font_size=14,opacity=0.7,font_family='mono
space')

fig.update_layout(
    font_family='monospace',
    title=dict(text='Q. How many samples of water are
Potable?',x=0.47,y=0.98,
               font=dict(color=colors_dark[2],size=20)),
    legend=dict(x=0.37,y=-0.05,orientation='h',traceorder='reversed'),
    hoverlabel=dict(bgcolor='white'))

fig.update_traces(textposition='outside', textinfo='percent+label')

fig.show()
```

Let's check out the distribution of the features.

Hardenss of water: The simple definition of water hardness is the amount of dissolved calcium and magnesium in the water. Hard water is high in dissolved minerals, largely calcium and magnesium. You may have felt the effects of hard water, literally, the last time you washed your hands. Depending on the hardness of your water, after using soap to wash you may have felt like there was a film of residue left on your hands. In hard water, soap reacts with the calcium (which is relatively high in hard water) to form "soap scum". When using hard water, more soap or detergent is needed to get things clean, be it your hands, hair, or your laundry.

```
fig =
px.histogram(df,x='Hardness',y=Counter(df['Hardness']),color='Potabili
ty',template='plotly_white',

marginal='box',opacity=0.7,nbins=100,color_discrete_sequence=[colors_g
reen[3],colors_blue[3]],
                  barmode='group',histfunc='count')

fig.add_vline(x=151, line_width=1,
line_color=colors_dark[1],line_dash='dot',opacity=0.7)
fig.add_vline(x=301, line_width=1,
```

```
line_color=colors_dark[1],line_dash='dot',opacity=0.7)
fig.add_vline(x=76, line_width=1,
line_color=colors_dark[1],line_dash='dot',opacity=0.7)

fig.add_annotation(text='<76 mg/L is<br> considered
soft',x=40,y=130,showarrow=False,font_size=9)
fig.add_annotation(text='Between 76 and 150<br> (mg/L)
is<br>moderately hard',x=113,y=130,showarrow=False,font_size=9)
fig.add_annotation(text='Between 151 and 300 (mg/L)<br> is considered
hard',x=250,y=130,showarrow=False,font_size=9)
fig.add_annotation(text='>300 mg/L is<br> considered very
hard',x=340,y=130,showarrow=False,font_size=9)

fig.update_layout(
    font_family='monospace',
    title=dict(text='Hardness Distribution',x=0.53,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Hardness (mg/L)',
    yaxis_title_text='Count',

legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,traceg
roupgap=5),
    bargap=0.3,
)
fig.show()
```

pH level: The pH of water is a measure of the acid–base equilibrium and, in most natural waters, is controlled by the carbon dioxide–bicarbonate–carbonate equilibrium system. An increased carbon dioxide concentration will therefore lower pH, whereas a decrease will cause it to rise. Temperature will also affect the equilibria and the pH. In pure water, a decrease in pH of about 0.45 occurs as the temperature is raised by 25 °C. In water with a buffering capacity imparted by bicarbonate, carbonate and hydroxyl ions, this temperature effect is modified (APHA, 1989). The pH of most drinking-water lies within the range 6.5–8.5. Natural waters can be of lower pH, as a result of, for example, acid rain or higher pH in limestone areas.

```
fig =
px.histogram(df,x='ph',y=Counter(df['ph']),color='Potability',template
='plotly_white',

marginal='box',opacity=0.7,nbins=100,color_discrete_sequence=[colors_g
reen[3],colors_blue[3]],
                  barmode='group',histfunc='count')

fig.add_vline(x=7, line_width=1,
line_color=colors_dark[1],line_dash='dot',opacity=0.7)

fig.add_annotation(text='<7 is
Acidic',x=4,y=70,showarrow=False,font_size=10)
fig.add_annotation(text='>7 is
```

```
Basic',x=10,y=70,showarrow=False,font_size=10)


fig.update_layout(
    font_family='monospace',
    title=dict(text='pH Level Distribution',x=0.5,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='pH Level',
    yaxis_title_text='Count',

legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,traceg
roupgap=5),
    bargap=0.3,
)
fig.show()
```

TDS: TDS means concentration of dissolved particles or solids in water. TDS comprises of inorganic salts such as calcium, magnesium, chlorides, sulfates, bicarbonates, etc, along with many more inorganic compounds that easily dissolve in water.

```
fig =
px.histogram(df,x='Solids',y=Counter(df['Solids']),color='Potability',
template='plotly_white',

marginal='box',opacity=0.7,nbins=100,color_discrete_sequence=[colors_g
reen[3],colors_blue[3]],
                  barmode='group',histfunc='count')

fig.update_layout(
    font_family='monospace',
    title=dict(text='Distribution Of Total Dissolved
Solids',x=0.5,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Dissolved Solids (ppm)',
    yaxis_title_text='Count',

legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,traceg
roupgap=5),
    bargap=0.3,
)
fig.show()
```

Chloramines: Chloramines (also known as secondary disinfection) are disinfectants used to treat drinking water and they:

- Are most commonly formed when ammonia is added to chlorine to treat drinking water.
- Provide longer-lasting disinfection as the water moves through pipes to consumers.

Chloramines have been used by water utilities since the 1930s.

```
fig =
px.histogram(df,x='Chloramines',y=Counter(df['Chloramines']),color='Po
tability',template='plotly_white',

marginal='box',opacity=0.7,nbins=100,color_discrete_sequence=[colors_g
reen[3],colors_blue[3]],
                  barmode='group',histfunc='count')

fig.add_vline(x=4, line_width=1,
line_color=colors_dark[1],line_dash='dot',opacity=0.7)

fig.add_annotation(text='<4 ppm is considered<br> safe for
drinking',x=1.8,y=90,showarrow=False)

fig.update_layout(
    font_family='monospace',
    title=dict(text='Chloramines Distribution',x=0.53,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Chloramines (ppm)',
    yaxis_title_text='Count',

legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,traceg
roupgap=5),
    bargap=0.3,
)
fig.show()
```

Sulfate: Sulfate (SO4) can be found in almost all natural water. The origin of most sulfate compounds is the oxidation of sulfite ores, the presence of shales, or the industrial wastes. Sulfate is one of the major dissolved components of rain. High concentrations of sulfate in the water we drink can have a laxative effect when combined with calcium and magnesium, the two most common constituents of hardness.

```
fig =
px.histogram(df,x='Sulfate',y=Counter(df['Sulfate']),color='Potability
',template='plotly_white',

marginal='box',opacity=0.7,nbins=100,color_discrete_sequence=[colors_g
reen[3],colors_blue[3]],
                  barmode='group',histfunc='count')

fig.add_vline(x=250, line_width=1,
line_color=colors_dark[1],line_dash='dot',opacity=0.7)

fig.add_annotation(text='<250 mg/L is considered<br> safe for
drinking',x=175,y=90,showarrow=False)

fig.update_layout(
    font_family='monospace',
    title=dict(text='Sulfate Distribution',x=0.53,y=0.95,
```

```
                font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Sulfate (mg/L)',
    yaxis_title_text='Count',

legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,traceg
roupgap=5),
    bargap=0.3,
)
fig.show()
```

Conductivity: Conductivity is a measure of the ability of water to pass an electrical current. Because dissolved salts and other inorganic chemicals conduct electrical current, conductivity increases as salinity increases. Organic compounds like oil do not conduct electrical current very well and therefore have a low conductivity when in water. Conductivity is also affected by temperature: the warmer the water, the higher the conductivity.

```
fig =
px.histogram(df,x='Conductivity',y=Counter(df['Conductivity']),color='
Potability',template='plotly_white',

marginal='box',opacity=0.7,nbins=100,color_discrete_sequence=[colors_g
reen[3],colors_blue[3]],
                  barmode='group',histfunc='count')

fig.add_annotation(text='The Conductivity range <br> is safe for both
(200-800),<br> Potable and Non-Potable water',
                   x=600,y=90,showarrow=False)

fig.update_layout(
    font_family='monospace',
    title=dict(text='Conductivity Distribution',x=0.5,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Conductivity (µS/cm)',
    yaxis_title_text='Count',

legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,traceg
roupgap=5),
    bargap=0.3,
)
fig.show()
```

Organic Carbon: Organic contaminants (natural organic substances, insecticides, herbicides, and other agricultural chemicals) enter waterways in rainfall runoff. Domestic and industrial wastewaters also contribute organic contaminants in various amounts. As a result of accidental spills or leaks, industrial organic wastes may enter streams. Some of the contaminants may not be completely removed by treatment processes; therefore, they could become a problem for drinking water sources. It is important to know the organic content in a waterway.

```
fig =
px.histogram(df,x='Organic_carbon',y=Counter(df['Organic_carbon']),col
or='Potability',template='plotly_white',

marginal='box',opacity=0.7,nbins=100,color_discrete_sequence=[colors_g
reen[3],colors_blue[3]],
                  barmode='group',histfunc='count')

fig.add_vline(x=10, line_width=1,
line_color=colors_dark[1],line_dash='dot',opacity=0.7)

fig.add_annotation(text='Typical Organic Carbon<br> level is upto 10
ppm',x=5.3,y=110,showarrow=False)

fig.update_layout(
    font_family='monospace',
    title=dict(text='Organic Carbon Distribution',x=0.5,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Organic Carbon (ppm)',
    yaxis_title_text='Count',

legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,traceg
roupgap=5),
    bargap=0.3,
)
fig.show()
```

Trihalomethanes: Trihalomethanes (THMs) are the result of a reaction between the chlorine used for disinfecting tap water and natural organic matter in the water. At elevated levels, THMs have been associated with negative health effects such as cancer and adverse reproductive outcomes.

```
fig =
px.histogram(df,x='Trihalomethanes',y=Counter(df['Trihalomethanes']),c
olor='Potability',template='plotly_white',

marginal='box',opacity=0.7,nbins=100,color_discrete_sequence=[colors_g
reen[3],colors_blue[3]],
                  barmode='group',histfunc='count')

fig.add_vline(x=80, line_width=1,
line_color=colors_dark[1],line_dash='dot',opacity=0.7)

fig.add_annotation(text='Upper limit of Trihalomethanes<br> level is
80 µg/L',x=115,y=90,showarrow=False)

fig.update_layout(
    font_family='monospace',
    title=dict(text='Trihalomethanes Distribution',x=0.5,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
```

```
    xaxis_title_text='Trihalomethanes (µg/L)',
    yaxis_title_text='Count',

legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,traceg
roupgap=5),
    bargap=0.3,
)
fig.show()
```

Turbidity: Turbidity is the measure of relative clarity of a liquid. It is an optical characteristic of water and is a measurement of the amount of light that is scattered by material in the water when a light is shined through the water sample. The higher the intensity of scattered light, the higher the turbidity. Material that causes water to be turbid include clay, silt, very tiny inorganic and organic matter, algae, dissolved colored organic compounds, and plankton and other microscopic organisms.

```
fig =
px.histogram(df,x='Turbidity',y=Counter(df['Turbidity']),color='Potabi
lity',template='plotly_white',

marginal='box',opacity=0.7,nbins=100,color_discrete_sequence=[colors_g
reen[3],colors_blue[3]],
                  barmode='group',histfunc='count')

fig.add_vline(x=5, line_width=1,
line_color=colors_dark[1],line_dash='dot',opacity=0.7)

fig.add_annotation(text='<5 NTU Turbidity is<br> considered
safe',x=6,y=90,showarrow=False)

fig.update_layout(
    font_family='monospace',
    title=dict(text='Turbidity Distribution',x=0.5,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Turbidity (NTU)',
    yaxis_title_text='Count',

legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,traceg
roupgap=5),
    bargap=0.3,
)
fig.show()
```

Scatter Plot Matrix helps in finding out the correlation between all the features.

```
fig =
px.scatter_matrix(df,df.drop('Potability',axis=1),height=1250,width=12
50,template='plotly_white',opacity=0.7,
```

```
color_discrete_sequence=[colors_blue[3],colors_green[3]],color='Potabi
lity',

symbol='Potability',color_continuous_scale=[colors_green[3],colors_blu
e[3]])

fig.update_layout(font_family='monospace',font_size=10,
                  coloraxis_showscale=False,
                 legend=dict(x=0.02,y=1.07,bgcolor=colors_dark[4]),
                 title=dict(text='Scatter Plot Matrix b/w
Features',x=0.5,y=0.97,
                   font=dict(color=colors_dark[2],size=24)))
fig.show()
```

As we can see, there seems to be very less correlation between all the features.

```
cor=df.drop('Potability',axis=1).corr()
cor
```

Let's make a Heatmap to visualize the correlation.

```
fig =
px.imshow(cor,height=800,width=800,color_continuous_scale=colors_blue,
template='plotly_white')

fig.update_layout(font_family='monospace',
                 title=dict(text='Correlation Heatmap',x=0.5,y=0.93,
                                font=dict(color=colors_dark[2],size=24)),
                 coloraxis_colorbar=dict(len=0.85,x=1.1)
                  )

fig.show()
```

```
fig = msno.matrix(df,color=(0,0.5,0.5))
```

We can see that there are many missing values in the dataset, so I'll try to deal with it

```
df.isnull().sum()

df[df['Potability']==0].describe()

df[df['Potability']==1].describe()

df[df['Potability']==0][['ph','Sulfate','Trihalomethanes']].median()

df[df['Potability']==1][['ph','Sulfate','Trihalomethanes']].median()
```

We can see that the difference between the mean and median values of Potable and Non-Potable Water is minimal. So we use the overall median of the feature to impute the values

```
df['ph'].fillna(value=df['ph'].median(),inplace=True)
df['Sulfate'].fillna(value=df['Sulfate'].median(),inplace=True)
df['Trihalomethanes'].fillna(value=df['Trihalomethanes'].median(),inpl
ace=True)

df.isnull().sum()

X = df.drop('Potability',axis=1).values
y = df['Potability'].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=101)

scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Spot checking is a great method to find out the baseline models for our data. It's quite easy and takes really less time.

```
filterwarnings('ignore')
models =[("LR", LogisticRegression(max_iter=1000)),("SVC", SVC()),
('KNN',KNeighborsClassifier(n_neighbors=10)),
         ("DTC", DecisionTreeClassifier()),("GNB", GaussianNB()),
        ("SGDC", SGDClassifier()),("Perc", Perceptron()),
("NC",NearestCentroid()),
        ("Ridge", RidgeClassifier()),("NuSVC", NuSVC()),("BNB",
BernoulliNB()),
         ('RF',RandomForestClassifier()),('ADA',AdaBoostClassifier()),
        ('XGB',GradientBoostingClassifier()),
('PAC',PassiveAggressiveClassifier())]

results = []
names = []
finalResults = []

for name,model in models:
    model.fit(X_train, y_train)
    model_results = model.predict(X_test)
    score = precision_score(y_test, model_results,average='macro')
    results.append(score)
    names.append(name)
    finalResults.append((name,score))
```

```
finalResults.sort(key=lambda k:k[1],reverse=True)

finalResults
```

I chose the top 5 baseline models and performed Hyperparameter tuning to it. 2 models outshined other models and they were Random Forest and XGBoost so I choose them for my final model.

```
model_params = {
    'XGB':
    {
        'model':GradientBoostingClassifier(),
        'params':
        {
            'learning_rate':[0.0001,0.001,0.01,0.1],
            'n_estimators':[100,200,500,1000],
            'max_features':['sqrt','log2'],
            'max_depth':list(range(11))
        }
    },
    'Random Forest':
    {
        'model':RandomForestClassifier(),
        'params':
        {
            'n_estimators':[10,50,100,200],
            'max_features':['auto','sqrt','log2'],
            'max_depth':list(range(1,11))
        }
    }
}

cv = RepeatedStratifiedKFold(n_splits=5,n_repeats=2)
scores=[]
for model_name,params in model_params.items():
    rs =
RandomizedSearchCV(params['model'],params['params'],cv=cv,n_iter=20)
    rs.fit(X,y)
    scores.append([model_name,dict(rs.best_params_),rs.best_score_])
data=pd.DataFrame(scores,columns=['Model','Parameters','Score'])
data
```

So now we have the best parameters for our final ensembled model.

I've used the VotingClassfier to ensemble the models for better results!

```
param=data['Parameters']
model = VotingClassifier(estimators=[

('XGB',GradientBoostingClassifier(**param[0])),

('RF',RandomForestClassifier(**param[1])),
                                    ],voting='hard')

accuracy=[]
scaler = StandardScaler()
skf = RepeatedStratifiedKFold(n_splits=5,n_repeats=2)
skf.get_n_splits(X,y)

for train_index, test_index in skf.split(X,y):

    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    scaler.fit(X_train)
    X_train = scaler.transform(X_train)
    X_test = scaler.transform(X_test)

    model.fit(X_train,y_train)
    predictions=model.predict(X_test)
    score=accuracy_score(y_test,predictions)
    accuracy.append(score)

np.mean(accuracy)
```

I believe the features aren't proving much help to the model to distinguish between the 2 classes which is stopping the model to perform better.

---

1. The TDS levels seem to contain some descripency since its values are on an average 40 folds more than the upper limit for safe drinking water.
2. The data contains almost equal number of acidic and basic pH level water samples.
3. 92% of the data was considered Hard.
4. Only 2% of the water samples were safe in terms of Chloramines levels.
5. Only 1.8% of the water samples were safe in terms of Sulfate levels.
6. 90.6% of the water samples had higher Carbon levels than the typical Carbon levels in drinking water (10 ppm).
7. 76.6% of water samples were safe for drinking in terms of Trihalomethane levels in water.

8. 90.4% of the water samples were safe for drinking in terms of the Turbidity of water samples.
9. The correlation coefficients between the features were very low.
10. Random Forest and XGBoost worked the best to train the model.
11. The ensemble method of using the Voting Classfier on Stratified K-folded samples gave an accuracy of >64%
