```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
plt.style.use('dark_background')
import seaborn as sns
color = sns.color_palette()
import plotly.express          as ex
import plotly.graph_objs       as go
import plotly.offline          as pyo
import scipy.stats             as stats
import pymc3                   as pm
import theano.tensor           as tt
from matplotlib.colors import ListedColormap
from scipy.stats import norm, boxcox
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score
from collections import Counter
from scipy import stats
from tqdm import tqdm_notebook

from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error,
confusion_matrix, r2_score, accuracy_score
from sklearn.model_selection import (GridSearchCV, KFold, train_test_split,
cross_val_score)

from imblearn.over_sampling import SMOTE
from collections import Counter

from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn import svm
from xgboost.sklearn import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
```

Importing The Dataset

```
path = "water_potability.csv"
df = pd.read_csv(path)
```

Initial Analysis

```
df.shape
```

(3276, 10)

```
df.info()
```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):

#	Column	Non-Null Count	Dtype
0	ph	2785 non-null	float64
1	Hardness	3276 non-null	float64
2	Solids	3276 non-null	float64
3	Chloramines	3276 non-null	float64
4	Sulfate	2495 non-null	float64
5	Conductivity	3276 non-null	float64
6	Organic_carbon	3276 non-null	float64
7	Trihalomethanes	3114 non-null	float64
8	Turbidity	3276 non-null	float64
9	Potability	3276 non-null	int64

dtypes: float64(9), int64(1)
memory usage: 256.1 KB

```
df.nunique()
```

ph 2785
Hardness 3276
Solids 3276
Chloramines 3276
Sulfate 2495
Conductivity 3276
Organic_carbon 3276
Trihalomethanes 3114
Turbidity 3276
Potability 2
dtype: int

Statistical Analysis

```
df.describe().T.style.background_gradient(subset=['mean','std','50%','count'],
cmap='PuBu')
```

```
#Portability is 1 - means good for Human
df[df['Potability']==1].describe().T.style.background_gradient(subset=['mean',
'std','50%','count'], cmap='PuBu')
```

```
# Portability is 0 - means not good for Human
df[df['Potability']==0].describe().T.style.background_gradient(subset=['mean',
'std','50%','count'], cmap='RdBu')
```

Check for missing values

```
plt.title('Missing Values Per Feature')
nans = df.isna().sum().sort_values(ascending=False).to_frame()
sns.heatmap(nans,annot=True,fmt='d',cmap='vlag')
```

```
##################################### Imputing 'ph' value
#####################################

phMean_0 = df[df['Potability'] == 0]['ph'].mean(skipna=True)
df.loc[(df['Potability'] == 0) & (df['ph'].isna()), 'ph'] = phMean_0
phMean_1 = df[df['Potability'] == 1]['ph'].mean(skipna=True)
df.loc[(df['Potability'] == 1) & (df['ph'].isna()), 'ph'] = phMean_1

##################################### Imputing 'Sulfate' value
#####################################

SulfateMean_0 = df[df['Potability'] == 0]['Sulfate'].mean(skipna=True)
df.loc[(df['Potability'] == 0) & (df['Sulfate'].isna()), 'Sulfate'] =
SulfateMean_0
SulfateMean_1 = df[df['Potability'] == 1]['Sulfate'].mean(skipna=True)
df.loc[(df['Potability'] == 1) & (df['Sulfate'].isna()), 'Sulfate'] =
SulfateMean_1

################################## Imputing 'Trihalomethanes' value
#####################################

TrihalomethanesMean_0 = df[df['Potability'] ==
0]['Trihalomethanes'].mean(skipna=True)
```

```
df.loc[(df['Potability'] == 0) & (df['Trihalomethanes'].isna()),
'Trihalomethanes'] = TrihalomethanesMean_0
TrihalomethanesMean_1 = df[df['Potability'] ==
1]['Trihalomethanes'].mean(skipna=True)
df.loc[(df['Potability'] == 1) & (df['Trihalomethanes'].isna()),
'Trihalomethanes'] = TrihalomethanesMean_1
```

```
print('Checking to see any more missing data \n')
df.isna().sum()
```

Checking to see any more missing data

ph 0
Hardness 0
Solids 0
Chloramines 0
Sulfate 0
Conductivity 0
Organic_carbon 0
Trihalomethanes 0
Turbidity 0
Potability 0
dtype: int64

Exploratory Data Analysis

```
Corrmat = df.corr()
plt.subplots(figsize=(7,7))
sns.heatmap(Corrmat, cmap="YlGnBu", square = True, annot=True, fmt='.2f')
plt.show()
```

```
fig =  ex.pie (df, names = "Potability", hole = 0.4, template = "plotly_dark")
fig.show ()
```

```
sns.violinplot(x='Potability', y='ph', data=df, palette='rocket')
```

```
print('Boxplot and density distribution of different features by
Potability\n')

fig, ax = plt.subplots(ncols=2, nrows=9, figsize=(14, 28))
```

```
features = list(df.columns.drop('Potability'))
i=0
for cols in features:
    sns.kdeplot(df[cols], fill=True, alpha=0.4, hue = df.Potability,
                palette=('indianred', 'steelblue'), multiple='stack',
ax=ax[i,0])

    sns.boxplot(data= df, y=cols, x='Potability', ax=ax[i, 1],
                palette=('indianred', 'steelblue'))
    ax[i,0].set_xlabel(' ')
    ax[i,1].set_xlabel(' ')
    ax[i,1].set_ylabel(' ')
    ax[i,1].xaxis.set_tick_params(labelsize=14)
    ax[i,0].tick_params(left=False, labelleft=False)
    ax[i,0].set_ylabel(cols, fontsize=16)
    i=i+1

plt.show()
```

SMOTE

```
###################### Preparing the Data for Modelling ######################

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

############################## Train-Test split ##############################
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25)

########################### Synthetic OverSampling ###########################
print('Balancing the data by SMOTE - Oversampling of Minority level\n')
smt = SMOTE()
counter = Counter(y_train)
print('Before SMOTE', counter)
X_train, y_train = smt.fit_resample(X_train, y_train)
counter = Counter(y_train)
print('\nAfter SMOTE', counter)

################################## Scaling ###################################
ssc = StandardScaler()

X_train = ssc.fit_transform(X_train)
X_test = ssc.transform(X_test)

modelAccuracy = list()
```

Modelling and Prediction

```
model = [LogisticRegression(), DecisionTreeClassifier(), GaussianNB(),
RandomForestClassifier(),
        svm.LinearSVC(), XGBClassifier()]
trainAccuracy = list()
testAccuracy = list()
kfold = KFold(n_splits=10, random_state=7, shuffle=True)

for mdl in model:
    trainResult = cross_val_score(mdl, X_train, y_train, scoring='accuracy',
cv=kfold)
    trainAccuracy.append(trainResult.mean())
    mdl.fit(X_train, y_train)
    y_pred = mdl.predict(X_test)
    testResult = metrics.accuracy_score(y_test, y_pred)
    testAccuracy.append(testResult)
```

```
print('Random Forest Classifier\n')
Rfc = RandomForestClassifier()
Rfc.fit(X_train, y_train)

y_Rfc = Rfc.predict(X_test)
print(metrics.classification_report(y_test, y_Rfc))
print(modelAccuracy.append(metrics.accuracy_score(y_test, y_Rfc)))

sns.heatmap(confusion_matrix(y_test, y_Rfc), annot=True, fmt='d')
plt.show()
```

```
Random Forest Classifier

              precision    recall  f1-score   support

           0       0.83      0.79      0.81       506
           1       0.69      0.74      0.72       313

    accuracy                           0.77       819
   macro avg       0.76      0.77      0.76       819
weighted avg       0.78      0.77      0.78       819

None
```

```
################################### XGB Classifier() ######################
print('XGB Classifier\n')
xgb = XGBClassifier()
xgb.fit(X_train, y_train)
```

```
y_xgb = xgb.predict(X_test)
print(metrics.classification_report(y_test, y_xgb))
print(modelAccuracy.append(metrics.accuracy_score(y_test, y_xgb)))

sns.heatmap(confusion_matrix(y_test, y_xgb), annot=True, fmt='d')
plt.show()
```

```
The use of label encoder in XGBClassifier is deprecated and will be
removed in a future release. To remove this warning, do the
following: 1) Pass option use_label_encoder=False when constructing
XGBClassifier object; and 2) Encode your labels (y) as integers
starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

              precision    recall  f1-score   support

           0       0.85      0.80      0.83       506
           1       0.71      0.78      0.74       313

    accuracy                           0.79       819
   macro avg       0.78      0.79      0.79       819
weighted avg       0.80      0.79      0.80       819

None
```

Conclusion

The Solid levels seem to contain some descripency since its values are on an average 40 folds more than the upper limit for safe drinking water.(Desirable limit for TDS is 500 mg/l and maximum limit is 1000 mg/l which prescribed for drinking purpose.)

The data contains almost equal number of acidic and basic pH level water samples.

The correlation coefficients between the features were very low.

Random Forest and XGBoost worked the best to train the model, both gives us f1 score (Balanced with precision & recall) as around 76%.