Project 2_PuppalaSucharitha

February 5, 2023

0.0.1 Term Project

0.0.2 Project - 2

0.0.3 Puppala Sucharitha

0.0.4 Data Science, Bellevue University

0.0.5 DSC680-T301 Applied Data Science (2233-1)

0.0.6 Prof. Catherine Williams

0.0.7 Date : 01/25/2023

[1]:
```
# Importing all the necessary libraries.
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy.stats import skew
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix, accuracy_score,␣
 ↪classification_report
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
# Loading the dataset named water_potability.csv
waterdf = pd.read_csv('water_potability.csv')
```

[3]:
```
# Getting the first 5 rows of the dataset.
waterdf.head()
```

[3]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	0
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180013	56.329076	4.500656	0
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.868637	66.420093	3.055934	0
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.341674	4.628771	0
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	0

[4]:
```
# Getting the shape of the dataset
waterdf.shape
```

[4]:
```
(3276, 10)
```

[5]:
```
# Getting the size of the dataset
waterdf.size
```

[5]:
```
32760
```

[6]:
```
# Information about the dataset variables.
waterdf.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
```

[7]:
```
# Checking for null values
waterdf.isna().sum()
```

[7]:
```
ph                491
Hardness            0
Solids              0
Chloramines         0
Sulfate           781
Conductivity        0
Organic_carbon      0
Trihalomethanes   162
Turbidity           0
Potability          0
dtype: int64
```

Observations:

- We can see many null values present in the dataset.
- From the above we can see that the features ph,Sulfate and Trihalomethanes are having more null values. Let us fill the null values with the meaan values.

[8]:
```
# filling the null values with the column mean values.
waterdf['ph']=waterdf['ph'].fillna(waterdf.groupby(['Potability'])['ph'].
 ↪transform('mean'))
waterdf['Sulfate']=waterdf['Sulfate'].fillna(waterdf.
 ↪groupby(['Potability'])['Sulfate'].transform('mean'))
waterdf['Trihalomethanes']=waterdf['Trihalomethanes'].fillna(waterdf.
 ↪groupby(['Potability'])['Trihalomethanes'].transform('mean'))
```

[9]:
```
# Checking for null values again
waterdf.isna().sum()
```

[9]:
```
ph                0
Hardness          0
Solids            0
Chloramines       0
Sulfate           0
Conductivity      0
Organic_carbon    0
Trihalomethanes   0
Turbidity         0
Potability        0
dtype: int64
```

Observations: From the above we can see that the null values are being removed.

[10]:
```
# Checking if any duplicated values present in the data set.
waterdf.duplicated()
```

[10]:
```
0       False
1       False
2       False
3       False
4       False
        …
3271    False
3272    False
3273    False
3274    False
3275    False
Length: 3276, dtype: bool
```

Observations: Check for duplicate values is done and there are no duplicate values in the dataset.

[11]:
```
# Getting the number of unique columns in the data set
waterdf.nunique()
```

[11]:
```
ph                2787
Hardness          3276
Solids            3276
Chloramines       3276
Sulfate           2497
Conductivity      3276
Organic_carbon    3276
Trihalomethanes   3116
Turbidity         3276
Potability           2
dtype: int64
```

[12]:
```
# Initially lets's use the describe() to get an idea on the dataset.
waterdf.describe()
```

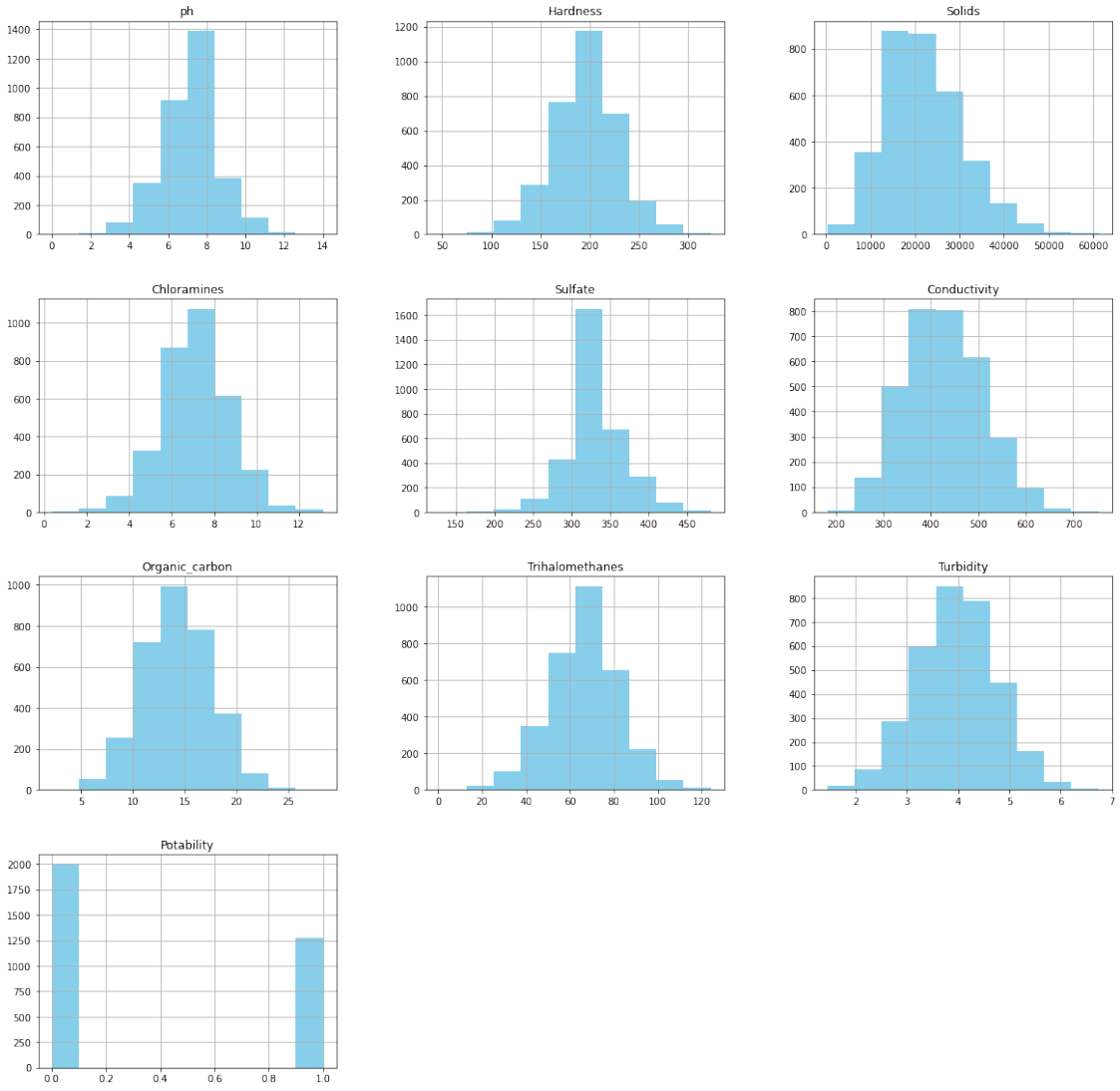
[12]:

	ph	Hardness	Solids	Chloramines	Sulfate
count	3276.000000	3276.000000	3276.000000	3276.000000	3276.000000
mean	7.080855	196.369496	22014.092526	7.122277	333.785123
std	1.469958	32.879761	8768.570828	1.583085	36.145701
min	0.000000	47.432000	320.942611	0.352000	129.000000
25%	6.277673	176.850538	15666.690297	6.127421	317.094638
50%	7.085378	196.967627	20927.833607	7.130299	334.564290
75%	7.870050	216.667456	27332.762127	8.114887	350.385756
max	14.000000	323.124000	61227.196008	13.127000	481.030642

	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
count	3276.000000	3276.000000	3276.000000	3276.000000	3276.000000
mean	426.205111	14.284970	66.395671	3.966786	0.390110
std	80.824064	3.308162	15.769901	0.780382	0.487849

min	181.483754	2.200000	0.738000	1.450000	0.000000
25%	365.734414	12.065801	56.647656	3.439711	0.000000
50%	421.884968	14.218338	66.303555	3.955028	0.000000
75%	481.792304	16.557652	76.666609	4.500320	1.000000
max	753.342620	28.300000	124.000000	6.739000	1.000000

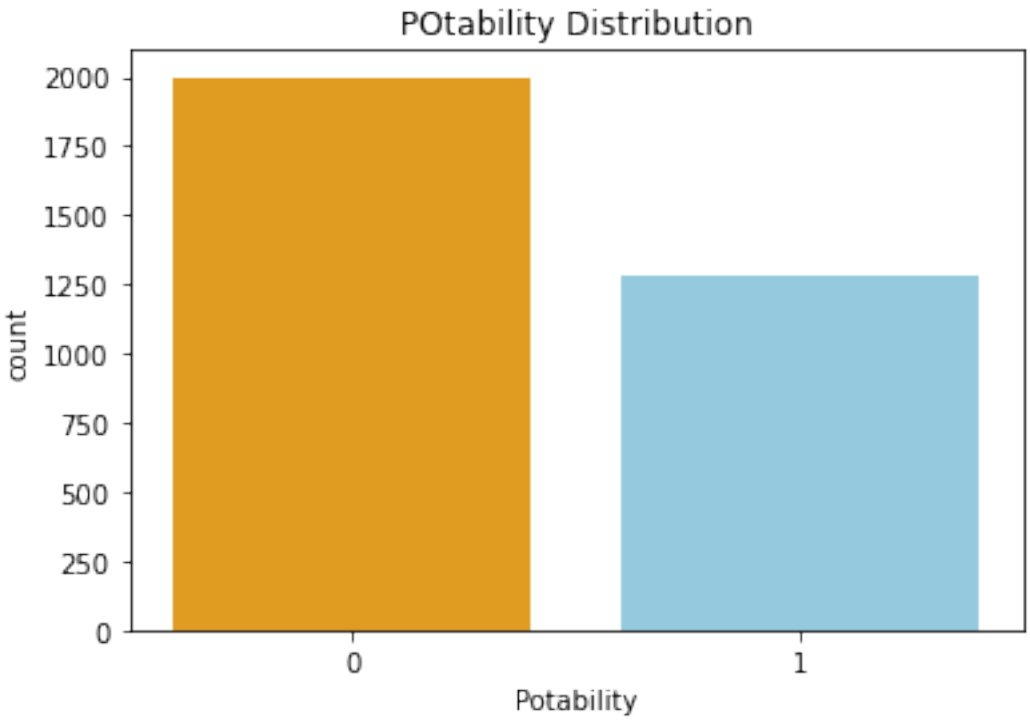
[13]:
```
# Univariate analysis.
# plot histograms for each numerical variable
waterdf.hist(figsize = (20, 20),color='skyblue')
plt.show()
```



Observations: From the above histogram plot of all the variables we see that few features are slightly skewed.

[14]:
```
#Let's cheeck for the distribution of the target variable Potability.
sns.countplot(data=waterdf,x='Potability' , palette=['orange','skyblue']);
#Title of the plot
plt.title('POtability Distribution')
```

[14]: Text(0.5, 1.0, 'POtability Distribution')

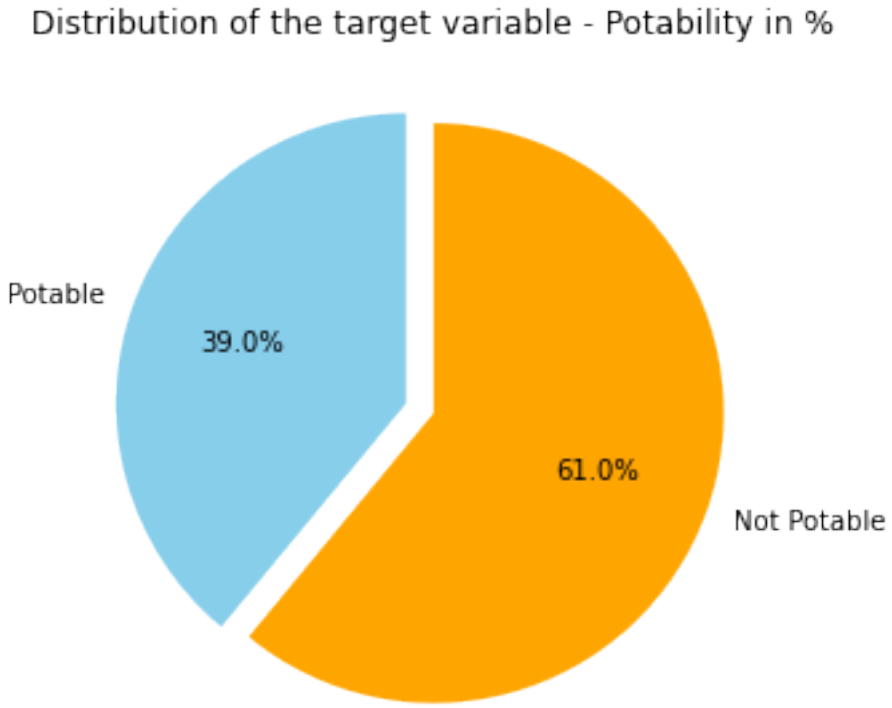


0.0.8 Observations:

From the above count plot we can see that the water that is potable i.e. safe for human consumption is less in count.

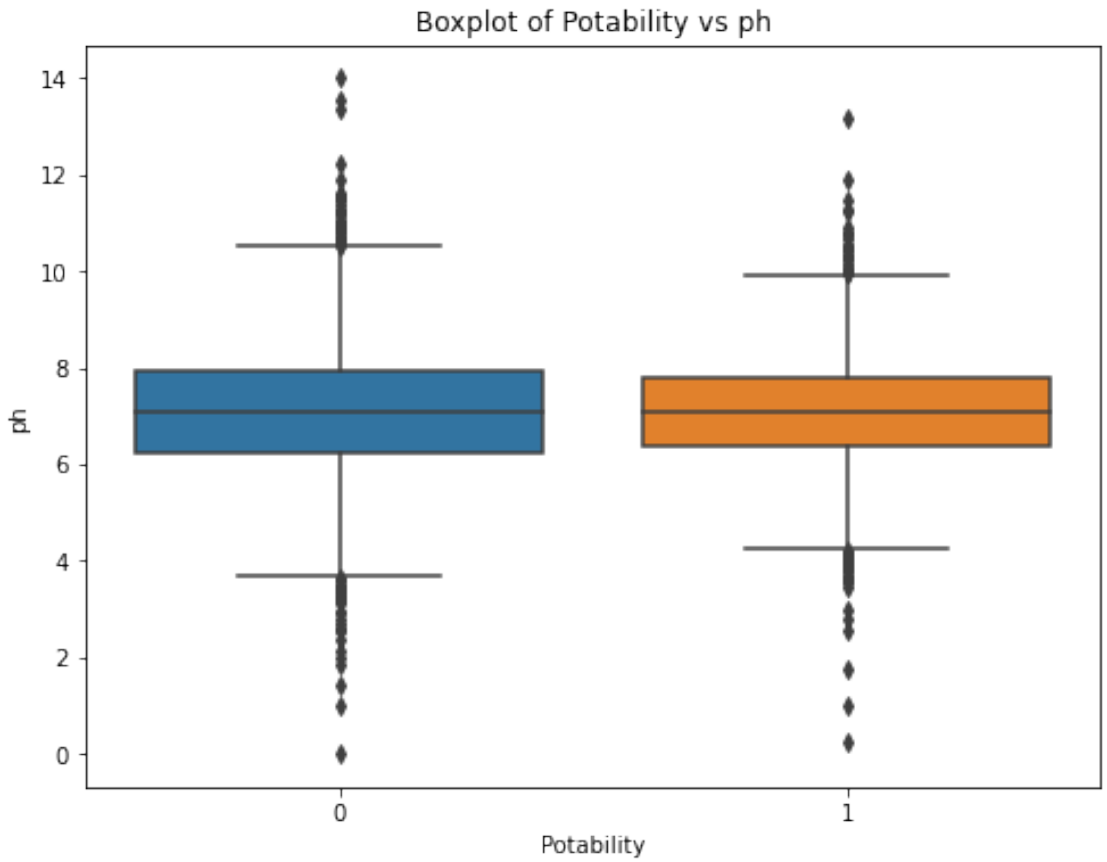
[15]:
```
# Let's plot a pie plot and count plot of the target variable 'Potability'
l = list(waterdf['Potability'].value_counts())
circle = [l[1] / sum(l) * 100,l[0] / sum(l) * 100]
# Setting the colors of the plots.
colors = ['skyblue','orange']

# Plotting the pie plot.
fig,ax = plt.subplots(nrows = 1,ncols = 1,figsize = (20,5))
plt.subplot(1,2,1)
plt.pie(circle,labels = ['Potable','Not Potable'],autopct='%1.1f%%',startangle
 ↪= 90,explode = (0.1,0),colors = colors)
```

```
plt.title('Distribution of the target variable - Potability in %'); # Title of␣
↪the plot.
```

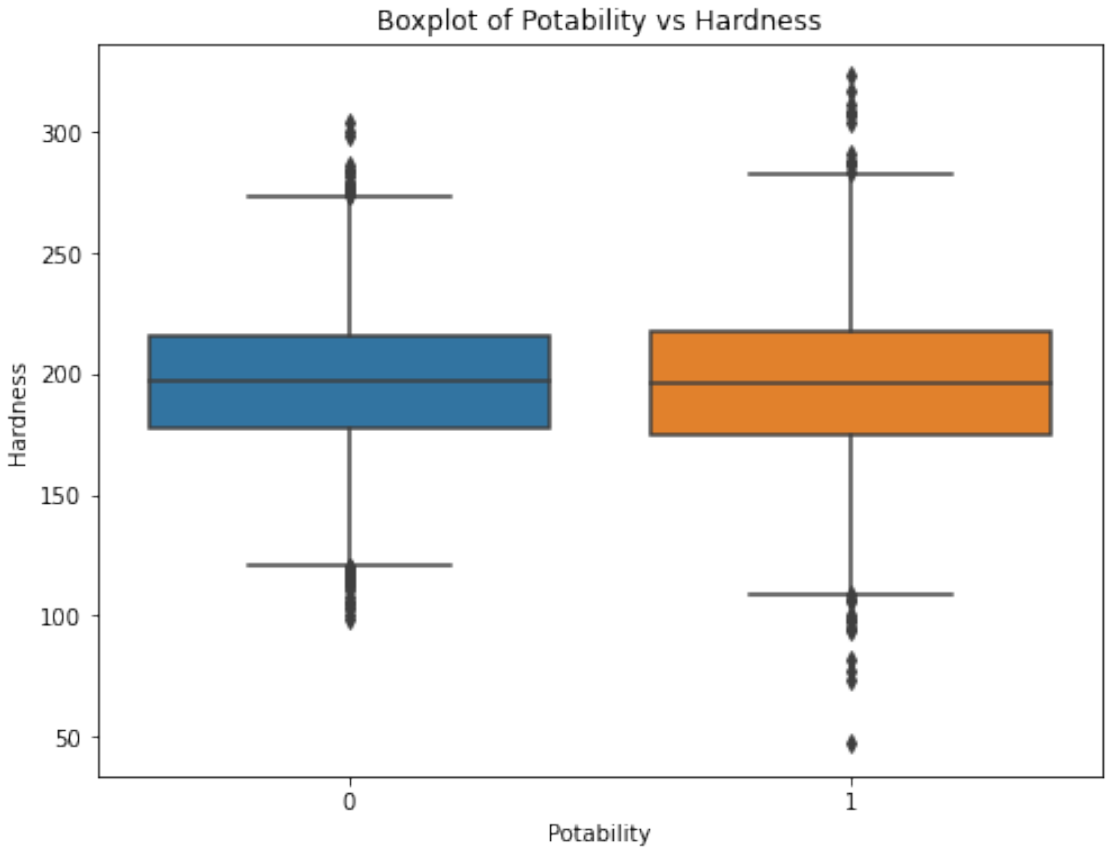


Observations: From the above plot we can say that about 39% of the water is potable i.e. safe for human consumption.

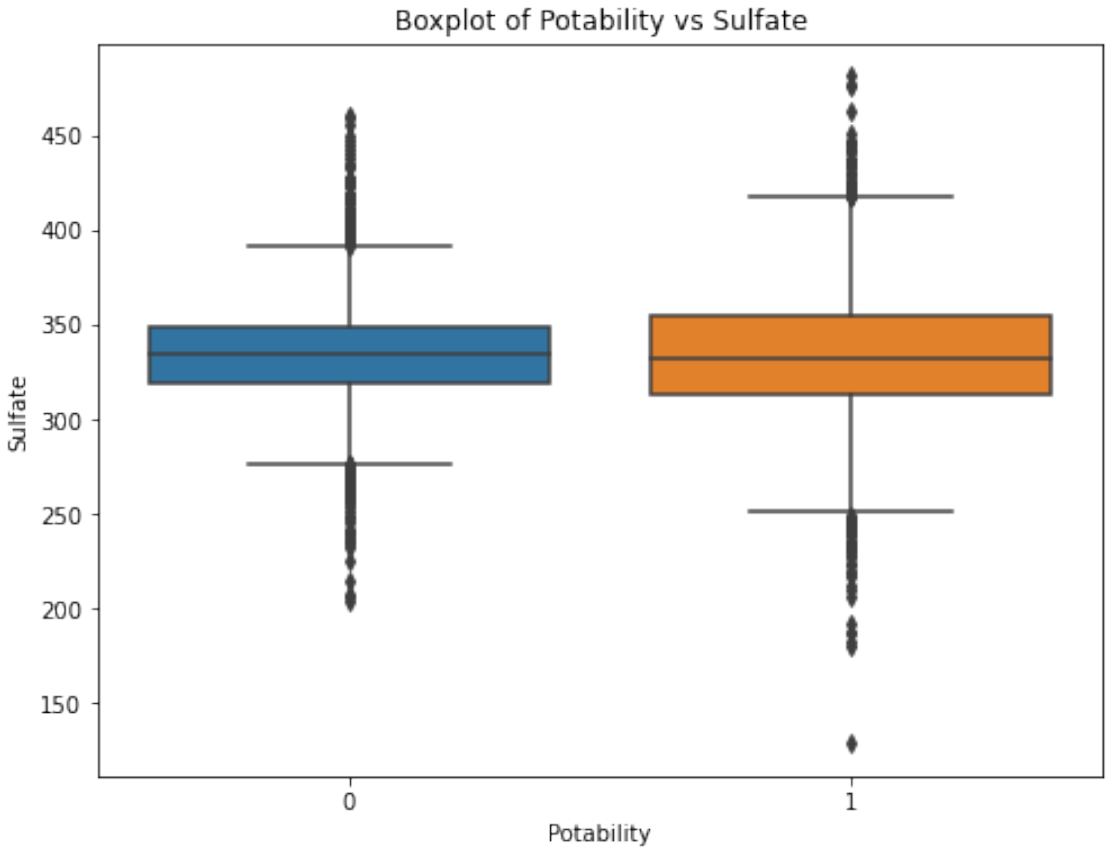
[16]:
```
# bivariate analysis
# seaborn bar plot gives the variable average
# defining the plot size
plt.figure(figsize=(8,6))
sns.boxplot(x="Potability",y="ph",data=waterdf)
plt.title("Boxplot of Potability vs ph")
plt.show()
```



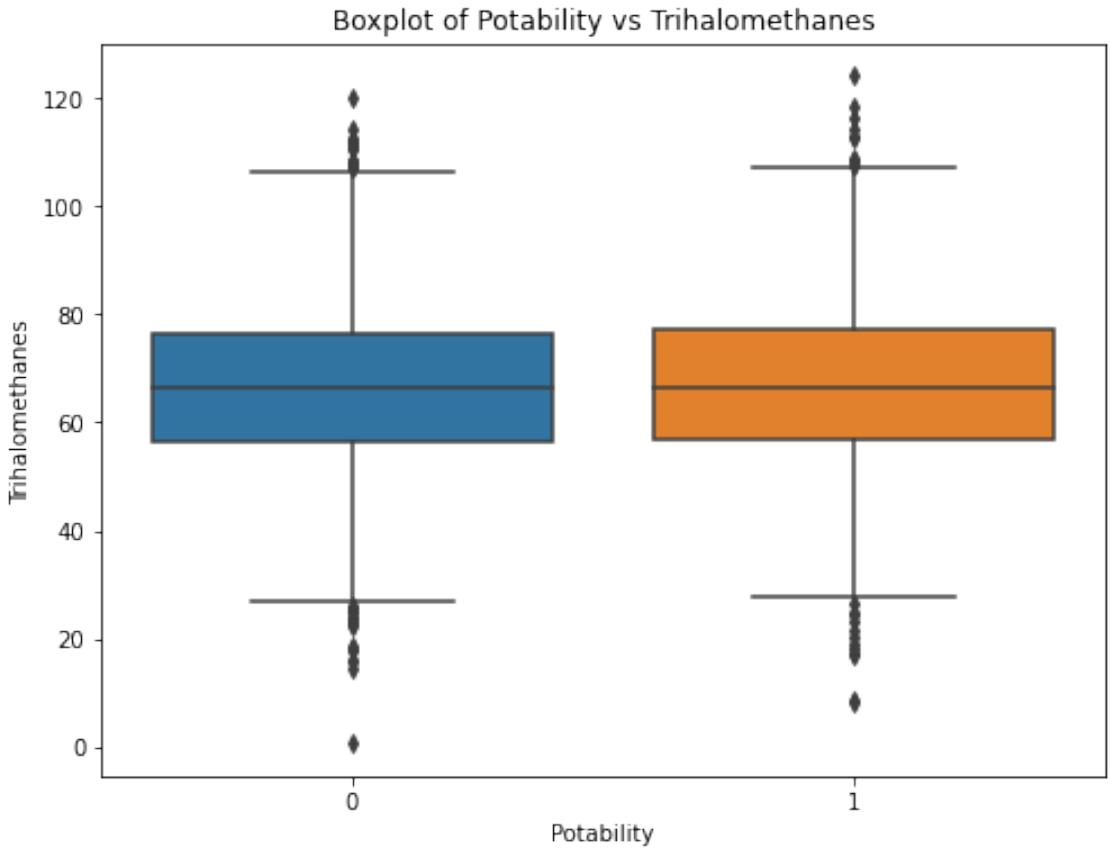
[17]:
```
plt.figure(figsize=(8,6))
sns.boxplot(x="Potability",y="Hardness",data=waterdf)
plt.title("Boxplot of Potability vs Hardness")
plt.show()
```



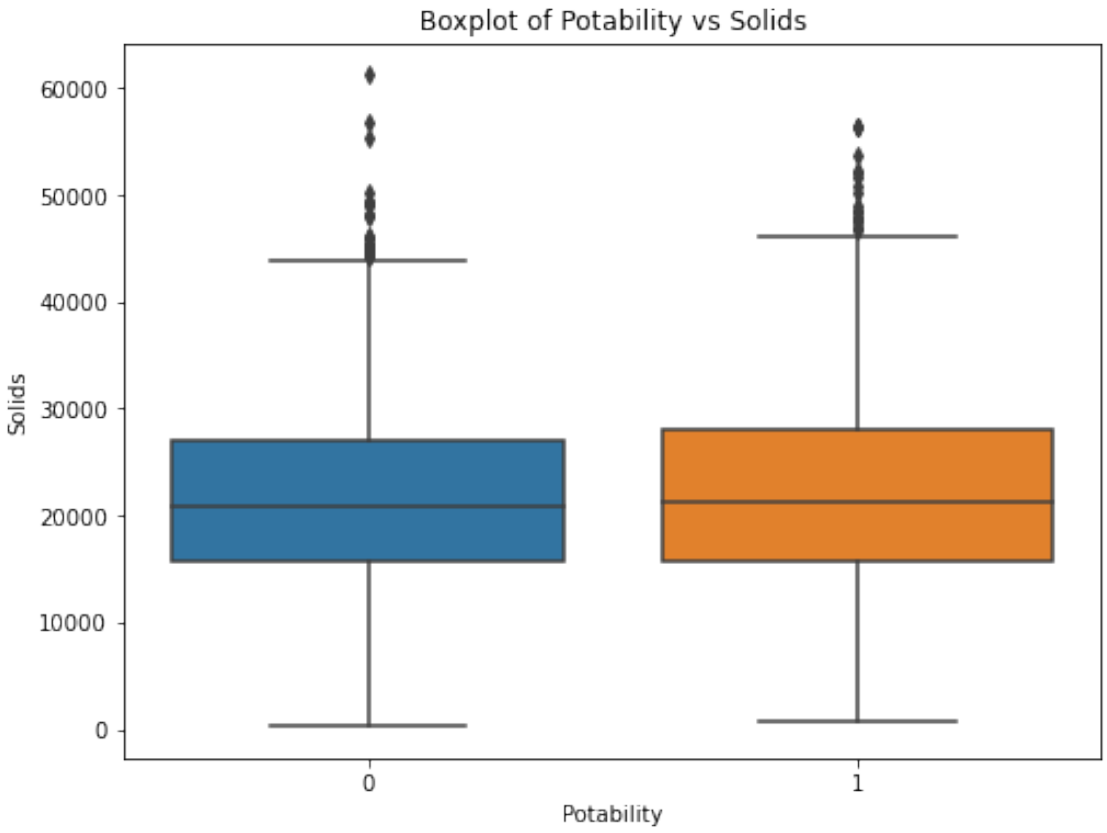
[18]:
```
plt.figure(figsize=(8,6))
sns.boxplot(x="Potability",y="Sulfate",data=waterdf)
plt.title("Boxplot of Potability vs Sulfate")
plt.show()
```



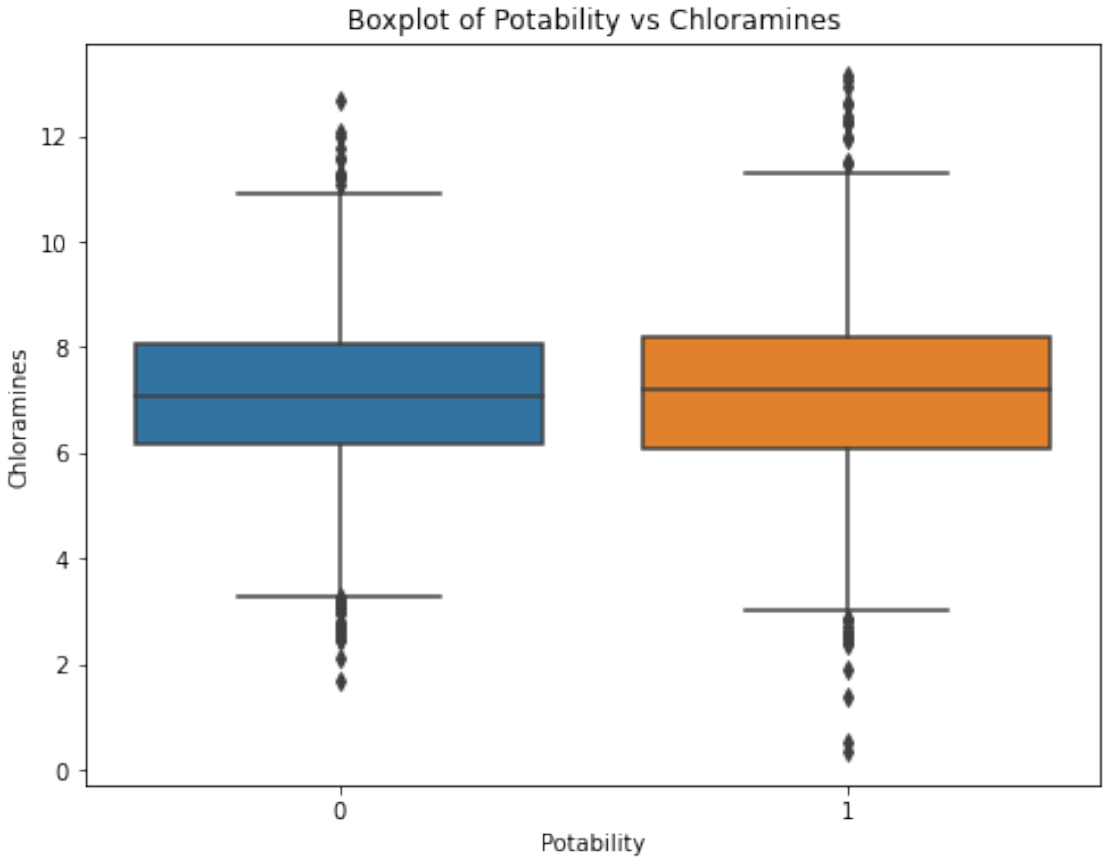
[19]:
```
plt.figure(figsize=(8,6))
sns.boxplot(x="Potability",y="Trihalomethanes",data=waterdf)
plt.title("Boxplot of Potability vs Trihalomethanes")
plt.show()
```



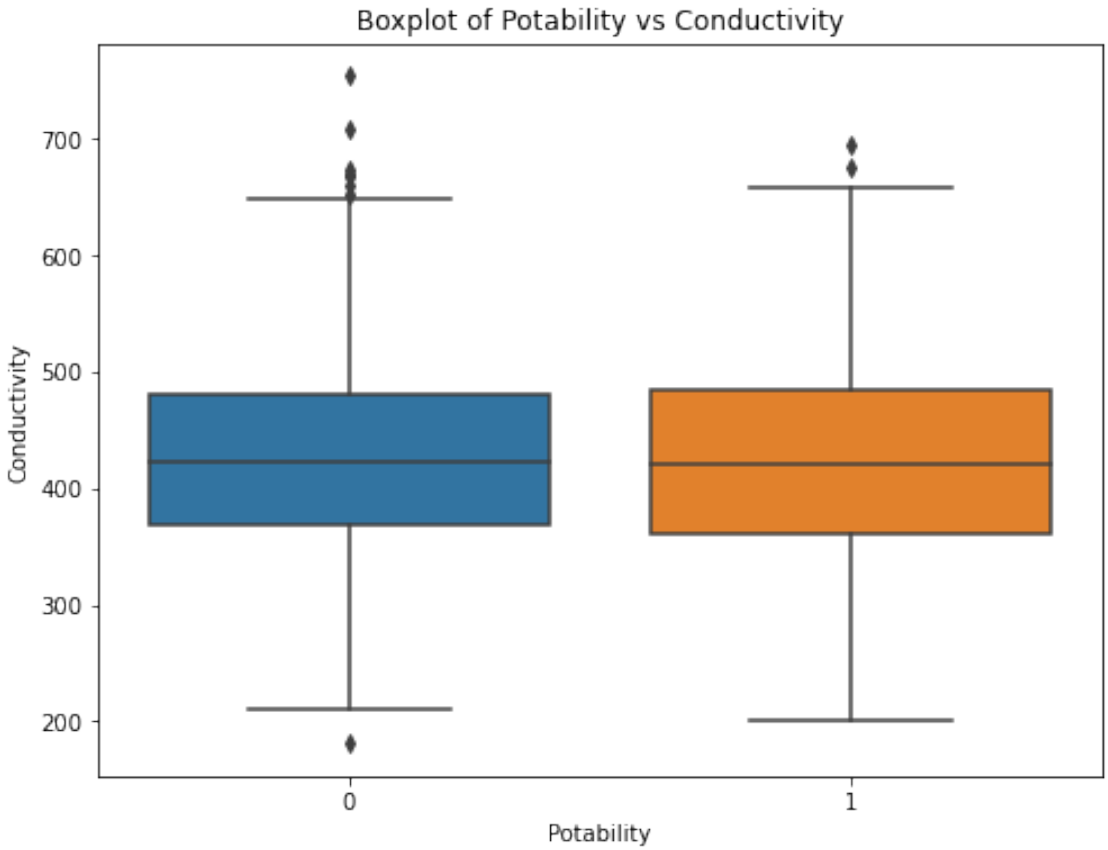
[20]:
```
plt.figure(figsize=(8,6))
sns.boxplot(x="Potability",y="Solids",data=waterdf)
plt.title("Boxplot of Potability vs Solids")
plt.show()
```



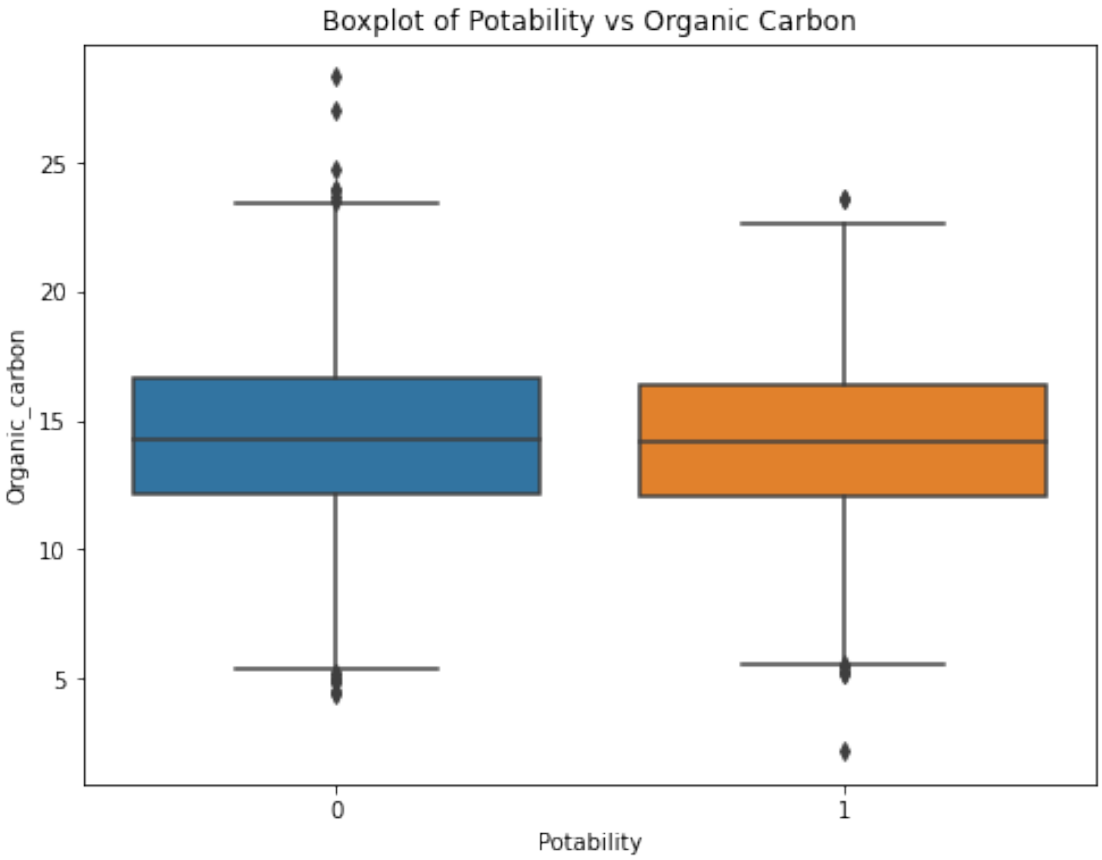
[21]:
```
plt.figure(figsize=(8,6))
sns.boxplot(x="Potability",y="Chloramines",data=waterdf)
plt.title("Boxplot of Potability vs Chloramines")
plt.show()
```



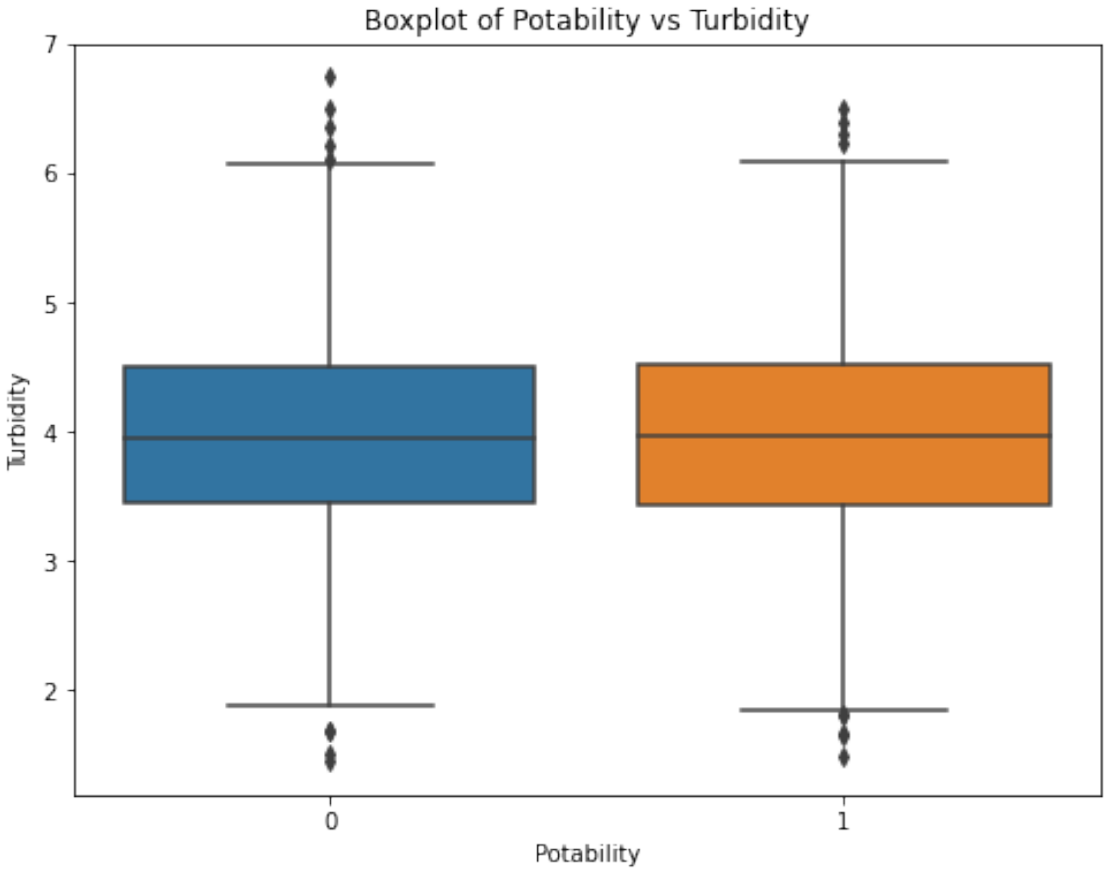
[22]:
```
plt.figure(figsize=(8,6))
sns.boxplot(x="Potability",y="Conductivity",data=waterdf)
plt.title("Boxplot of Potability vs Conductivity")
plt.show()
```



[23]:
```
plt.figure(figsize=(8,6))
sns.boxplot(x="Potability",y="Organic_carbon",data=waterdf)
plt.title("Boxplot of Potability vs Organic Carbon")
plt.show()
```



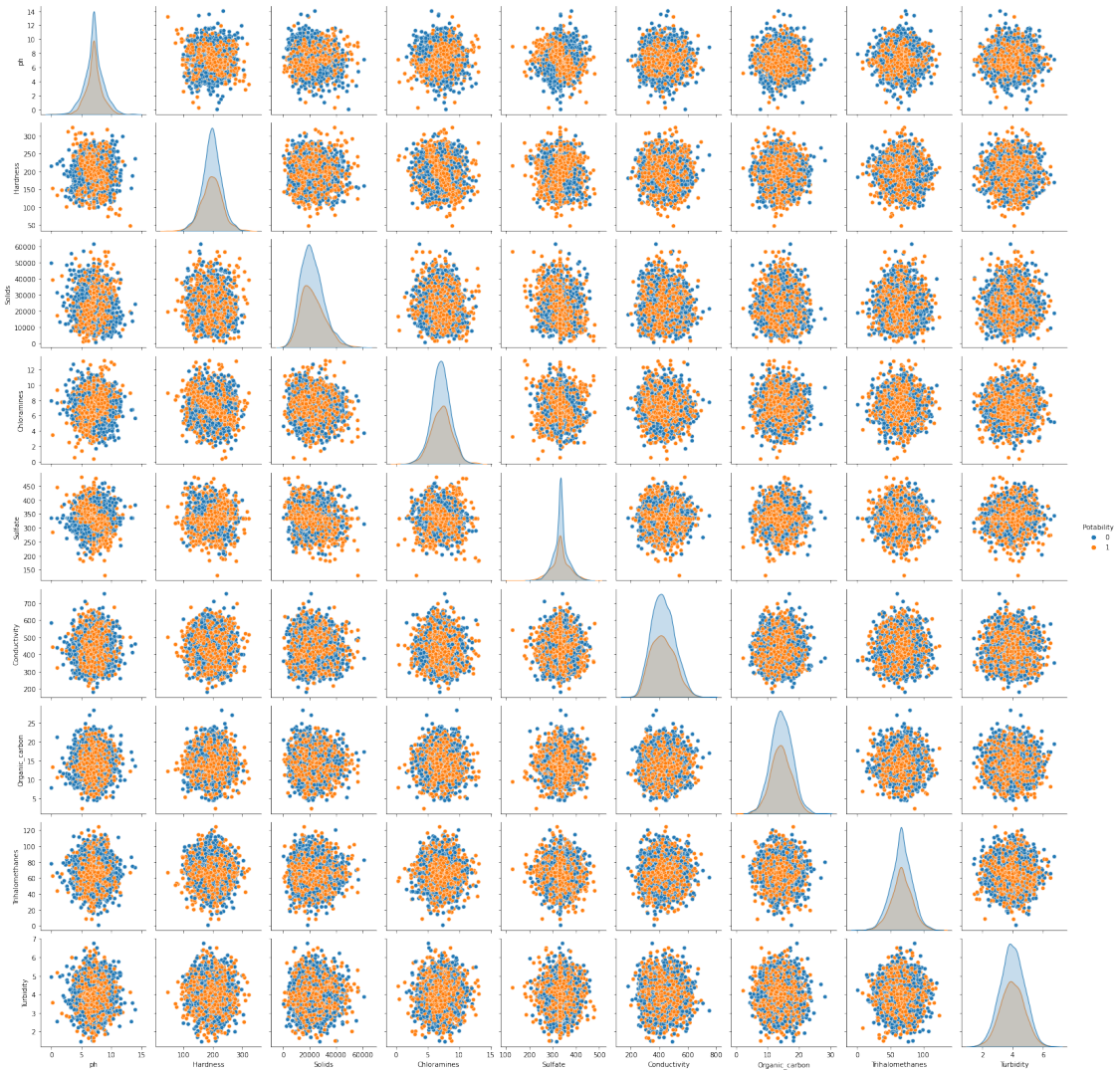
[24]:
```
plt.figure(figsize=(8,6))
sns.boxplot(x="Potability",y="Turbidity",data=waterdf)
plt.title("Boxplot of Potability vs Turbidity")
plt.show()
```



0.0.9 Observations:

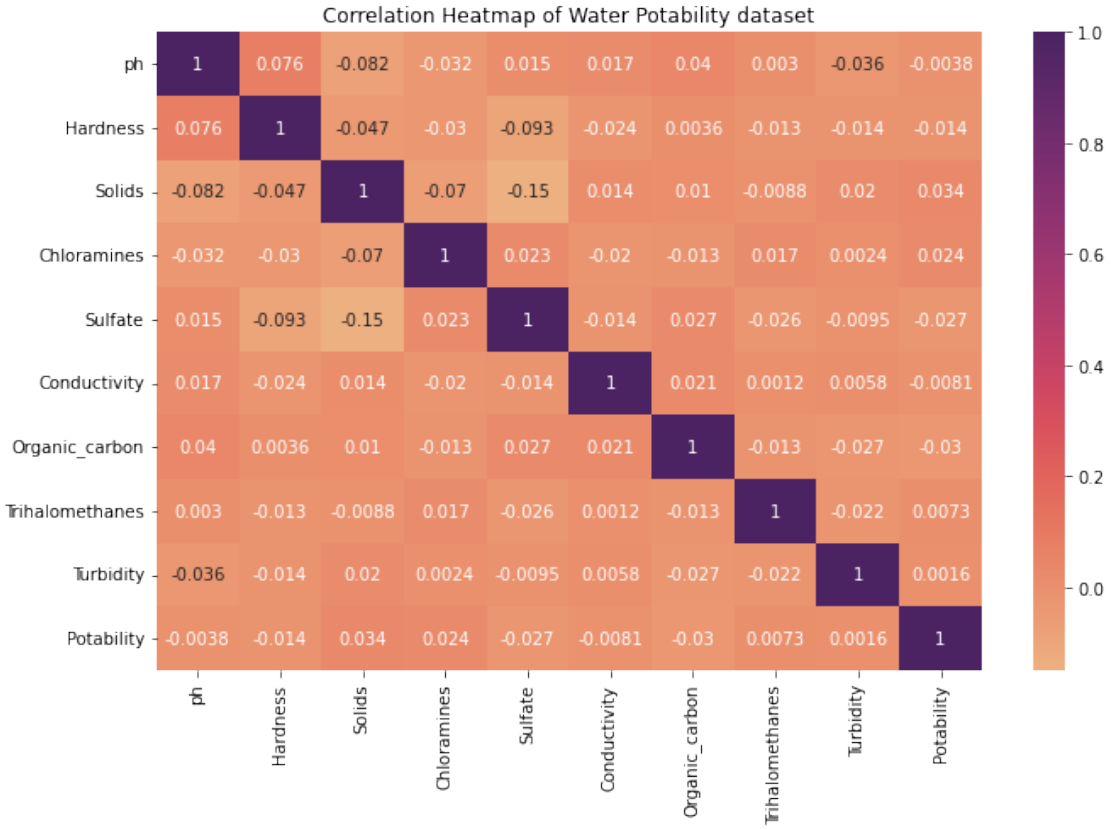
From the above box plots for the distribution of the features of the dataset we see that the presence of all the features is more in the water that is not safe for drinking, i.e. Potability = 0.

[25]:
```
# Let's use pairplot() function from seaborn to understand the relationship␣
↪between all features.

sns.pairplot(waterdf,hue="Potability");
```



[26]:
```
# Let's plot a correlation heatmap of the dataset.

# Calculate the correlation coefficient with corr().
corr_number = waterdf.corr()

# Create the heatmap for the correlation coefficients calculated above.
fig, ax = plt.subplots(1, 1, figsize=(10,7), tight_layout = True)
sns.heatmap(corr_number, annot = True, cmap = 'flare')

# Title of the plot
plt.title('Correlation Heatmap of Water Potability dataset')
```

[26]: Text(0.5, 1.0, 'Correlation Heatmap of Water Potability dataset')

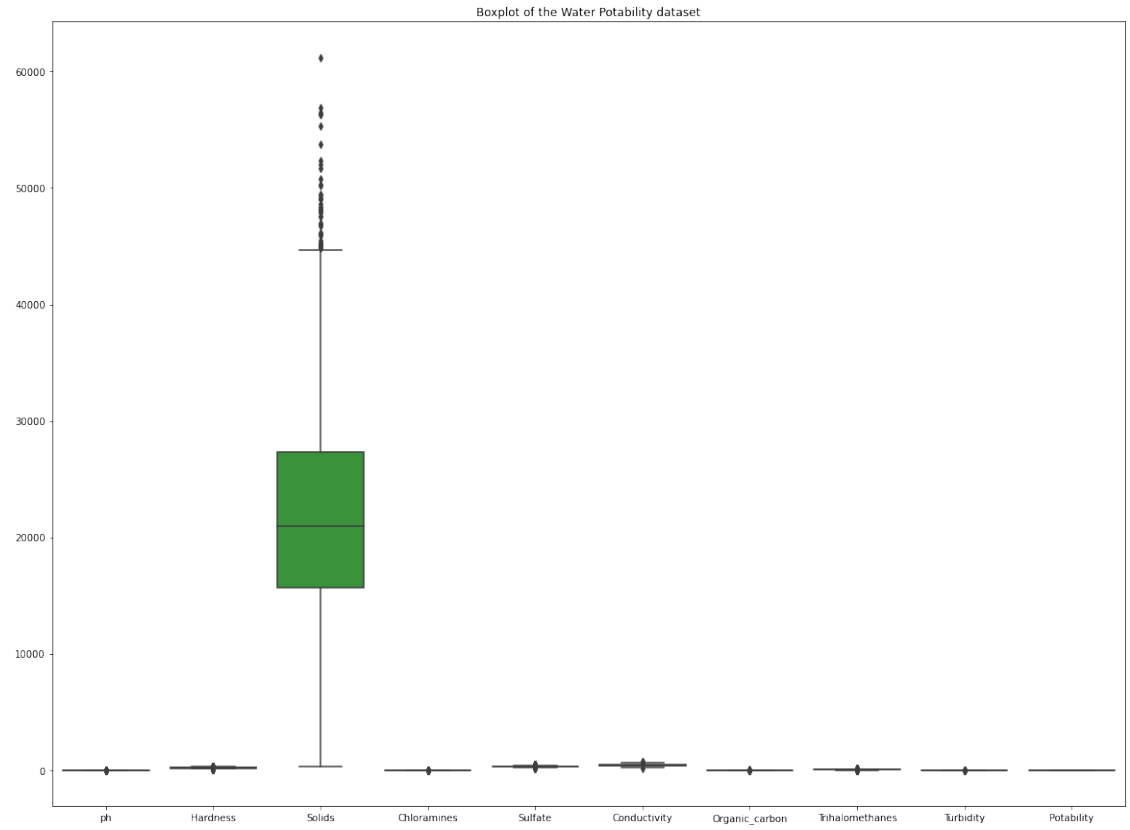


Observations:

- From the above correlation heat map we can see that there is no feature that is strongly correlated to the target variable "Potability".
- But from the above plot we can say that 'Hardness' has good correlation with the target variable 'Potability' but with very low correlation coefficient value.

[27]:
```
# let's plot a boxplot for the diabetes dataset.
plt.figure(figsize=(20,15))
sns.boxplot(data = waterdf)
# Plot title
plt.title('Boxplot of the Water Potability dataset')
```

[27]:
```
Text(0.5, 1.0, 'Boxplot of the Water Potability dataset')
```



Observations: From the above boxplot we can see that there are few outliers present in the 'Solids' feature.

[28]:
```
#Let's define a function for detecting outliers using IQR method.

def outliers_IQR(df):

   q1 = df.quantile(0.25)

   q3 = df.quantile(0.75)

   IQR = q3-q1

   outlier = df[((df<(q1-1.5*IQR)) | (df>(q3+1.5*IQR)))]

   return outlier
```

[29]:
```
# Now let's check for the outliers in features : Glucose.
outlier = outliers_IQR(waterdf['Solids'])

print('Total number of outliers in feature Solids : '+ str(len(outlier)))
```

```
print('Max. outlier value : '+ str(outlier.max()))

print('Min. outlier value : '+ str(outlier.min()))
```

```
Total number of outliers in feature Solids : 47
Max. outlier value : 61227.19600771213
Min. outlier value : 44868.45836802399
```

Observations:

- We can see that the outliers are 47 present in the Solids feature.
- There are no zeros present when observed the outliers, let us proceed with the analysis without changing the values of the outliers present.

0.0.10 Modelling

[30]:
```
# Let's define the features and target variables X and y respectively.

X = waterdf.drop('Potability', axis = 1)
y = waterdf['Potability']
```

[31]:
```
# Let's split the dataset into 80% train and 20% test datasets using␣
 ↪train_test_split().
# Test size is 0.2
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2,␣
 ↪random_state = 42)
```

[32]:
```
# Let's get the size of the test and train datasets.

print("X_train shape : {} rows and {} columns.".format(X_train.shape[0],X_train.
 ↪shape[1]))
print("y_train shape : {} rows.".format(y_train.shape[0]))
print("X_test shape : {} rows and {} columns.".format(X_test.shape[0],X_test.
 ↪shape[1]))
print("y_test shape : {} rows.".format(y_test.shape[0]))
```

```
X_train shape : 2620 rows and 9 columns.
y_train shape : 2620 rows.
X_test shape : 656 rows and 9 columns.
y_test shape : 656 rows.
```

[33]:
```
# Using the standard scaler on X_train and X_test datasets.
# Fit the transform.

standscaler = StandardScaler()
X_train_std = standscaler.fit_transform(X_train)
X_test_std = standscaler.transform(X_test)
```

0.0.11 KNN Classifier

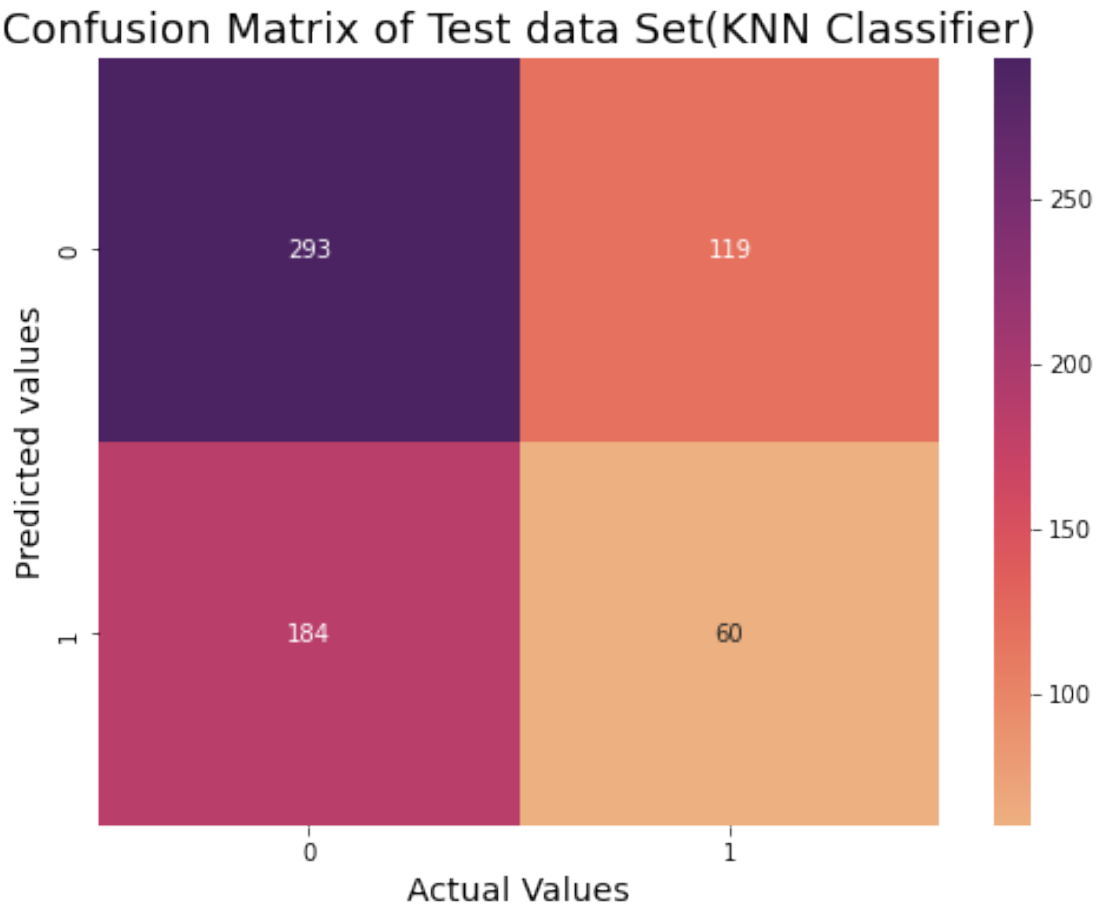
[34]:
```
# Create KNN classifier
knnclass = KNeighborsClassifier()

# Fit the model to train datasets.
knnclass = knnclass.fit(X_train, y_train )

# Create prediction of the model using the test data.
knnclass_pred = knnclass.predict(X_test)

# Create prediction of the model using the train data.
knnclass_pred_train = knnclass.predict(X_train)
```

[35]:
```
# Let's create a Confusion Matrix for the test set predictions.
knnconmatrix = confusion_matrix(y_test, knnclass_pred)

# Print the Confusion matrix.
print('Confusion Matrix of Test set predictions(KNN Classifier) : \n',␣
 ↪knnconmatrix)
```

```
Confusion Matrix of Test set predictions(KNN Classifier) :
 [[293 119]
 [184  60]]
```

[36]:
```
# Plot the confusion matrix.
# Define the size of the plot
plt.figure(figsize=(8,6))
#  Confusion matrix heat map.
sns.heatmap(knnconmatrix, annot=True,cmap = 'flare', fmt='d')
# Plot Title
plt.title('Confusion Matrix of Test data Set(KNN Classifier)', fontsize  = 18)
# x- Label
plt.xlabel('Actual Values', fontsize = 14)
# y-label
plt.ylabel('Predicted values', fontsize = 14)
```

[36]:
```
Text(51.0, 0.5, 'Predicted values')
```



[37]:
```
# Let's get the accuracy score, Precision, Recall and F1 Score of the KNN␣
 ↪Classifier.
# Getting the accuracy score of the test dataset.
knnclass_accuracy = metrics.accuracy_score(y_test, knnclass_pred)

# Getting the accuracy score of the train dataset
knnclass_accu_train = metrics.accuracy_score(y_train, knnclass_pred_train)

# Getting the precision score.
knnclass_precision = round(precision_score(y_test, knnclass_pred),3)

# Getting the Recall Score.
knnclass_recall = round(recall_score(y_test, knnclass_pred),3)

# Getting the F1 Score.
knnclass_f1score = round(f1_score(y_test, knnclass_pred),3)
```

```
# Printing the accuracy of the model.
print('The accuracy score of the KNN Classifier on test dataset: {} '.
 ↪format(knnclass_accuracy))
print('The accuracy score of the KNN Classifier on train dataset : {} '.
 ↪format(knnclass_accu_train))
print('Precision Score of the test set for the KNN Classifier : {}'.
 ↪format(knnclass_precision))
print('Recall Score of the test set for the KNN Classifier : {}'.
 ↪format(knnclass_recall))
print('F1 Score of the test set for the KNN Classifier : {}'.
 ↪format(knnclass_f1score))
```

```
The accuracy score of the KNN Classifier on test dataset: 0.538109756097561
The accuracy score of the KNN Classifier on train dataset : 0.7145038167938931
Precision Score of the test set for the KNN Classifier : 0.335
Recall Score of the test set for the KNN Classifier : 0.246
F1 Score of the test set for the KNN Classifier : 0.284
```

0.0.12 Random Forest Classifier

[38]:
```
# Let's create Random Forest Model.
rdreg = RandomForestClassifier()

# Fit the Random Forest model to training datasets.
rdreg.fit(X_train, y_train)

# Creating the predictions on the test data for model validation.
rdreg_pred = rdreg.predict(X_test)

# Creating the predictions on the trianed data for model validation.
rdreg_pred_train = rdreg.predict(X_train)
```

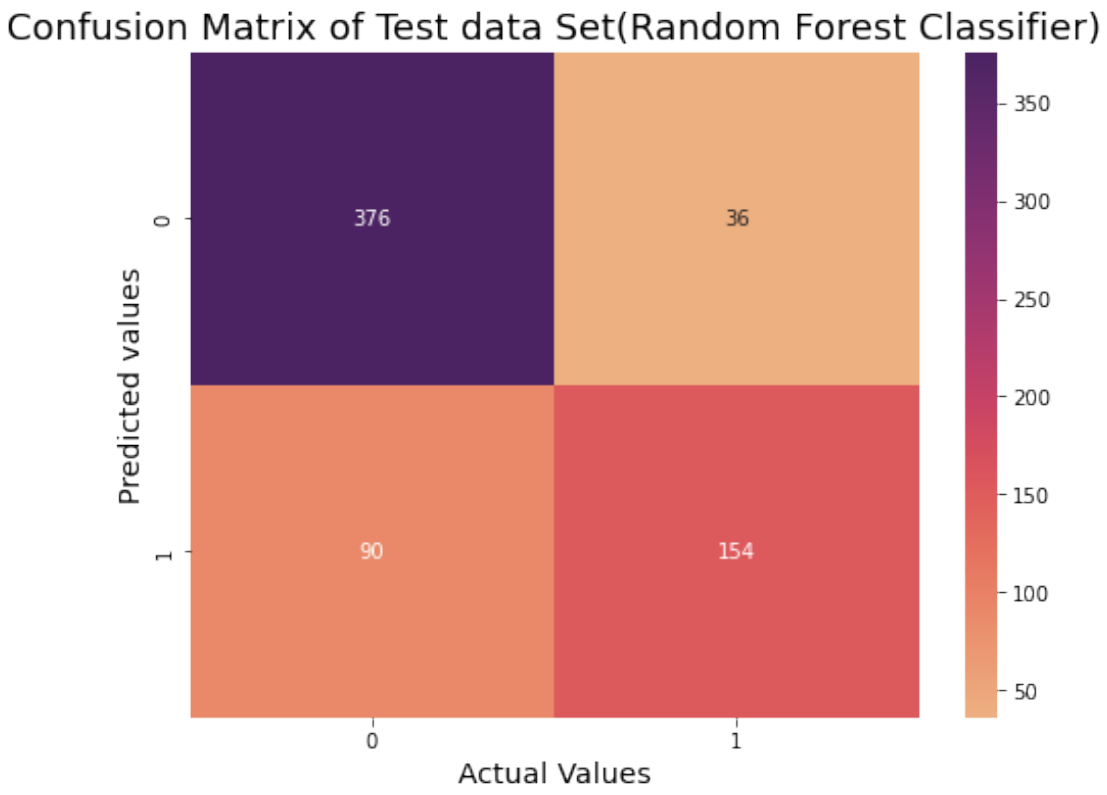
[39]:
```
# Let's create a Confusion Matrix for the test set predictions.
rdconmatrix = confusion_matrix(y_test, rdreg_pred)

# Print the Confusion matrix.
print('Confusion Matrix of Test set predictions(Support Vector Machine␣
 ↪Classifier): \n', rdconmatrix)
```

```
Confusion Matrix of Test set predictions(Support Vector Machine Classifier):
 [[376  36]
 [ 90 154]]
```

[40]:
```
# Plot the confusion matrix.
# Define the size of the plot
plt.figure(figsize=(8,6))
#  Confusion matrix heat map.
sns.heatmap(rdconmatrix, annot=True,cmap = 'flare', fmt='d')
```

```
# Plot Title
plt.title('Confusion Matrix of Test data Set(Random Forest Classifier)',␣
 ↪fontsize  = 18)
# x- Label
plt.xlabel('Actual Values', fontsize = 14)
# y-label
plt.ylabel('Predicted values', fontsize = 14)
```

[40]: Text(51.0, 0.5, 'Predicted values')



[41]:
```
# Let's get the accuracy score, Precision, Recall and F1 Score of the Support␣
 ↪Vector Machine Classifier.
# Getting the accuracy score of the test dataset.
rdreg_accuracy = metrics.accuracy_score(y_test, rdreg_pred)

# Getting the accuracy score of the train dataset
rdreg_accu_train = metrics.accuracy_score(y_train, rdreg_pred_train)

# Getting the precision score.
rdreg_precision = round(precision_score(y_test, rdreg_pred),3)
```

```
# Getting the Recall Score.
rdreg_recall = round(recall_score(y_test, rdreg_pred),3)

# Getting the F1 Score.
rdreg_f1score = round(f1_score(y_test, rdreg_pred),3)


# Printing the accuracy of the model.
print('The accuracy score of the Random Forest Classifier on test dataset: {} '.
 ↪format(rdreg_accuracy))
print('The accuracy score of the Random Forest Classifier on train dataset : {}␣
 ↪'.format(rdreg_accu_train))
print('Precision Score of the test set for the  Random Forest Classifier : {}'.
 ↪format(rdreg_precision))
print('Recall Score of the test set for the Random Forest Classifier : {}'.
 ↪format(rdreg_recall))
print('F1 Score of the test set for the Random Forest Classifier : {}'.
 ↪format(rdreg_f1score))
```

```
The accuracy score of the Random Forest Classifier on test dataset:
0.8079268292682927
The accuracy score of the Random Forest Classifier on train dataset : 1.0
Precision Score of the test set for the  Random Forest Classifier : 0.811
Recall Score of the test set for the Random Forest Classifier : 0.631
F1 Score of the test set for the Random Forest Classifier : 0.71
```

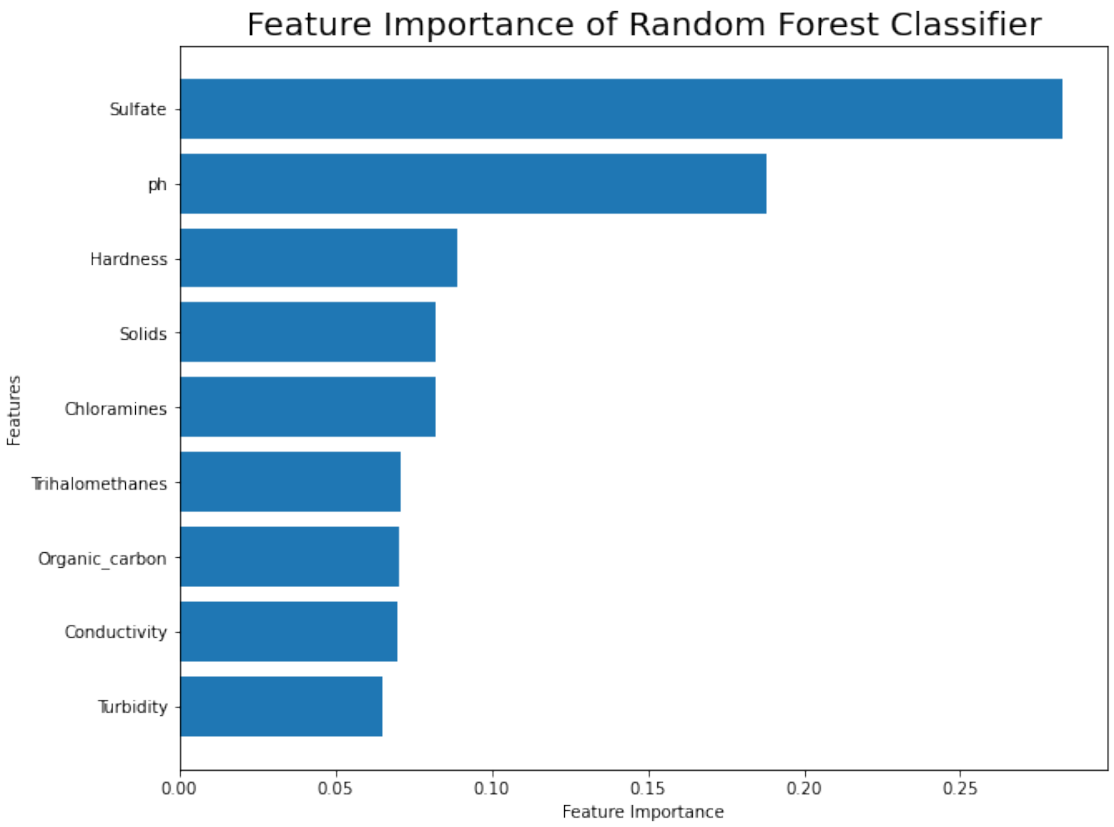
[42]:
```
# The importance of a feature is basically: how much this feature is used in␣
 ↪each tree of the forest.
importantfeatures = rdreg.feature_importances_
importantfeatures
```

[42]:
```
array([0.18781212, 0.08895214, 0.08228198, 0.08199801, 0.28297704,
       0.06966496, 0.07022538, 0.07091603, 0.06517234])
```

[43]:
```
# initialize the idicies
indices = np.argsort(importantfeatures)
# Defining the size of the plot
fig, ax = plt.subplots(figsize = (10,8))
# Plotting the graph
ax.barh(range(len(importantfeatures)), importantfeatures[indices])
# Y-ticks
ax.set_yticks(range(len(importantfeatures)))
_ = ax.set_yticklabels(np.array(X_train.columns)[indices])
# X-label
plt.xlabel("Feature Importance", fontsize = 10)
# Y - label
plt.ylabel("Features", fontsize = 10)
```

```
# Title of the plot
plt.title("Feature Importance of Random Forest Classifier", fontsize = 20)
```

[43]: Text(0.5, 1.0, 'Feature Importance of Random Forest Classifier')



0.0.13 XGB Classifier

[44]:
```
#from xgboost import XGBClassifier
from xgboost import XGBClassifier
```

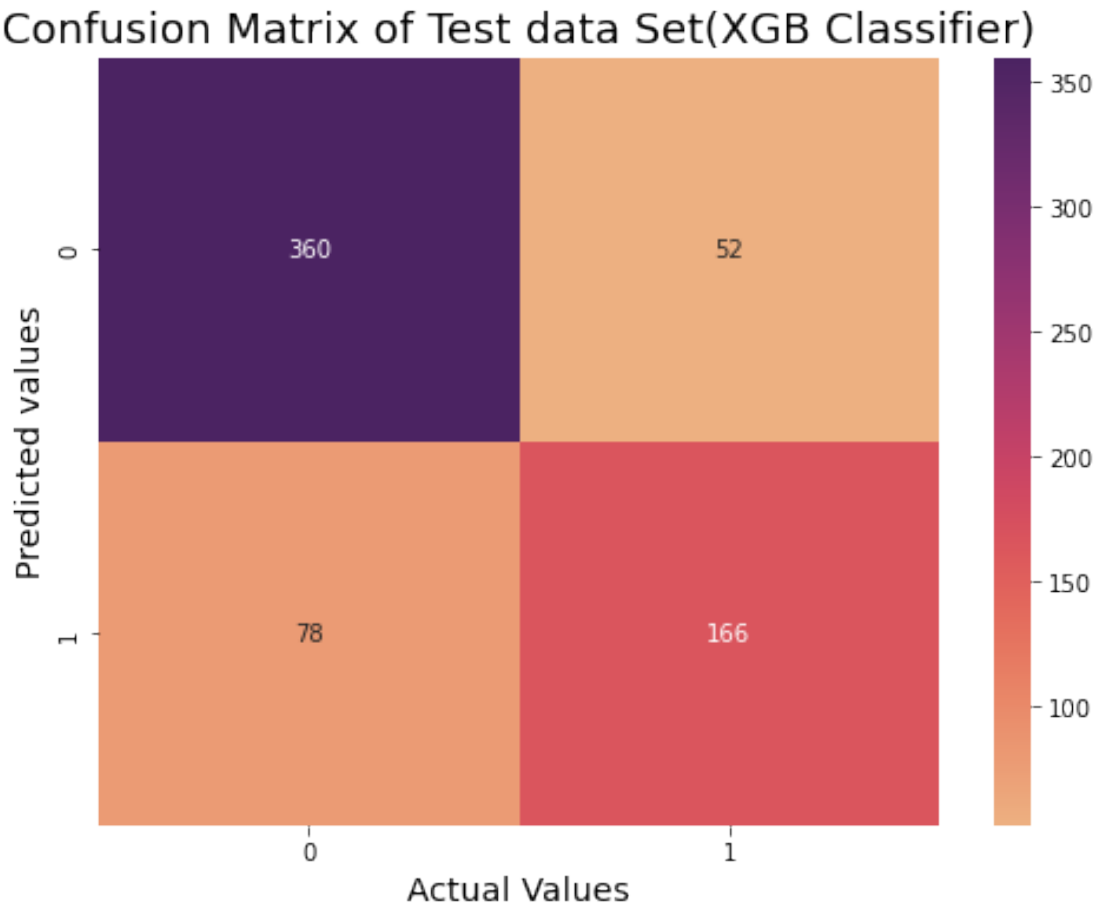
[45]:
```
# Let's create XGB classifier.
xg = XGBClassifier()

# Fit the Random Forest model to training datasets.
xg.fit(X_train, y_train)

# Creating the predictions on the test data for model validation.
xg_pred = xg.predict(X_test)

# Creating the predictions on the trianed data for model validation.
xg_pred_train = xg.predict(X_train)
```

[46]:
```
# Let's create a Confusion Matrix for the test set predictions.
xgconmatrix = confusion_matrix(y_test, xg_pred)

# Print the Confusion matrix.
print('Confusion Matrix of Test set predictions(XGB Classifier): \n',␣
 ↪xgconmatrix)
```

```
Confusion Matrix of Test set predictions(XGB Classifier):
 [[360  52]
 [ 78 166]]
```

[47]:
```
# Plot the confusion matrix.
# Define the size of the plot
plt.figure(figsize=(8,6))
#  Confusion matrix heat map.
sns.heatmap(xgconmatrix, annot=True,cmap = 'flare', fmt='d')
# Plot Title
plt.title('Confusion Matrix of Test data Set(XGB Classifier)', fontsize  = 18)
# x- Label
plt.xlabel('Actual Values', fontsize = 14)
# y-label
plt.ylabel('Predicted values', fontsize = 14)
```

[47]:
```
Text(51.0, 0.5, 'Predicted values')
```



[48]:
```
# Let's get the accuracy score, Precision, Recall and F1 Score of the Support␣
↪Vector Machine Classifier.
# Getting the accuracy score of the test dataset.
xg_accuracy = metrics.accuracy_score(y_test, xg_pred)

# Getting the accuracy score of the train dataset
xg_accu_train = metrics.accuracy_score(y_train, xg_pred_train)

# Getting the precision score.
xg_precision = round(precision_score(y_test, xg_pred),3)

# Getting the Recall Score.
xg_recall = round(recall_score(y_test, xg_pred),3)

# Getting the F1 Score.
xg_f1score = round(f1_score(y_test, xg_pred),3)
```

```
# Printing the accuracy of the model.
print('The accuracy score of the XGB Classifier on test dataset: {} '.
 ↪format(xg_accuracy))
print('The accuracy score of the XGB Classifier on train dataset : {} '.
 ↪format(xg_accu_train))
print('Precision Score of the test set for the XGB Classifier : {}'.
 ↪format(xg_precision))
print('Recall Score of the test set for the XGB Classifier : {}'.
 ↪format(xg_recall))
print('F1 Score of the test set for the XGB Classifier : {}'.format(xg_f1score))
```

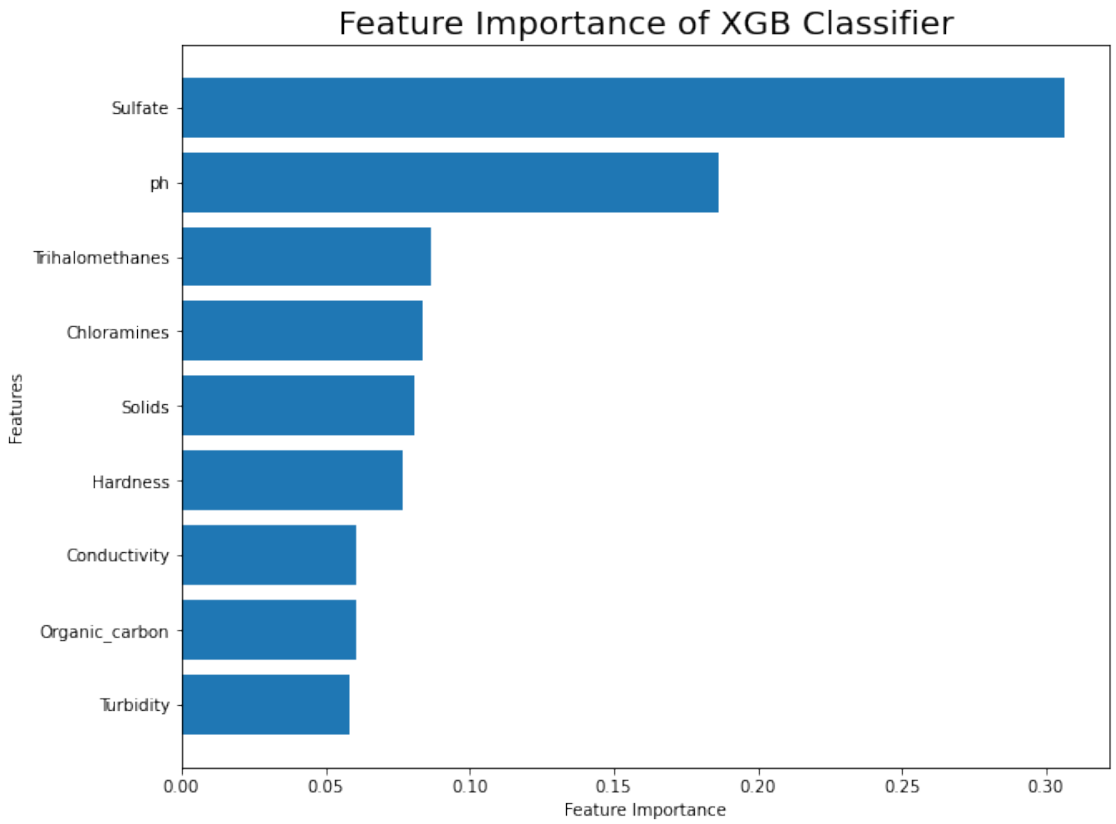
```
The accuracy score of the XGB Classifier on test dataset: 0.801829268292683
The accuracy score of the XGB Classifier on train dataset : 1.0
Precision Score of the test set for the XGB Classifier : 0.761
Recall Score of the test set for the XGB Classifier : 0.68
F1 Score of the test set for the XGB Classifier : 0.719
```

[49]:
```
# The importance of a feature is basically: how much this feature is used in␣
 ↪each tree of the forest.
xgimportantfeatures = xg.feature_importances_
xgimportantfeatures
```

[49]:
```
array([0.18623367, 0.07650602, 0.0807505 , 0.08349272, 0.30664834,
       0.06081047, 0.06049719, 0.08683765, 0.05822343], dtype=float32)
```

[50]:
```
# initialize the idicies
indices = np.argsort(xgimportantfeatures)
# Defining the size of the plot
fig, ax = plt.subplots(figsize = (10,8))
# Plotting the graph
ax.barh(range(len(xgimportantfeatures)), xgimportantfeatures[indices])
# Y-ticks
ax.set_yticks(range(len(xgimportantfeatures)))
_ = ax.set_yticklabels(np.array(X_train.columns)[indices])
# X-label
plt.xlabel("Feature Importance", fontsize = 10)
# Y - label
plt.ylabel("Features", fontsize = 10)
# Title of the plot
plt.title("Feature Importance of XGB Classifier", fontsize = 20)
```

[50]:
```
Text(0.5, 1.0, 'Feature Importance of XGB Classifier')
```



0.0.14 Model Comparisions

[51]:
```
# Let's form arrays for the calculated accuracy score for the train and test datasets, Precision,
# Recall and F1 Score for the above three models.

KNN_classifier = {'Model':'KNN Classifier',
                  'Accuracy (test)':knnclass_accuracy,
                'Accuracy (train)':knnclass_accu_train,
                'Precision score' :knnclass_precision,
                'Recall score'    :knnclass_recall,
                 'F1 Score':knnclass_f1score,}

RandomForestclassifier = {'Model':'RandomForestClassifier',
                          'Accuracy (test)':rdreg_accuracy,
                           'Accuracy (train)':rdreg_accu_train,
                          'Precision score' :rdreg_precision,
                          'Recall score'    :rdreg_recall,
                           'F1 Score':rdreg_f1score,}

XGBClassifier = {'Model':'XGBClassifier',
```

```
                        'Accuracy (test)':xg_accuracy,
                         'Accuracy (train)':xg_accu_train,
                        'Precision score' :xg_precision,
                        'Recall score'    :xg_recall,
                         'F1 Score':xg_f1score,}
```

[52]:
```
# Let's group the results of the three models using the pd.series()
models_evalmetrics = pd.DataFrame({'KNN Classifier': pd.Series(KNN_classifier),
                       'RandomForest Classifier':pd.
 ↪Series(RandomForestclassifier),
                       'XGB Classifier':pd.Series(XGBClassifier)
                      })

models_evalmetrics
```

[52]:

	KNN Classifier	RandomForest Classifier	XGB Classifier
Model	KNN Classifier	RandomForestClassifier	XGBClassifier
Accuracy (test)	0.53811	0.807927	0.801829
Accuracy (train)	0.714504	1.0	1.0
Precision score	0.335	0.811	0.761
Recall score	0.246	0.631	0.68
F1 Score	0.284	0.71	0.719

[ ]:
```

```