Water Quality Analysis

Using cognos

Phase 5: Documentation and submission

INTRODUCTION

• In this technique, our model predicts that the water is safe to drink or not , using some parameters like Ph value, conductivity, hardness, etc.

OBJECTIVE

- The main objective of the project is to determine the quality of water , whether it is drinkable or not.
- we can find it out by calculating the chemical properties such as pH, hardness,solids, chloramines, sulphate, conductivity, organic carbon, turbidity level.

DESIGN THINKING PROCESS

- **Analysis Objectives:**
  Define specific objectives for analysing water quality data, including assessing potability, identifying deviations from standards, and understanding parameter relationships.

- **Data Collection:**
  Gather the provided water quality data containing parameters like pH, Hardness, Solids, etc.

- **Visualization Strategy:**
  Plan how to visualize parameter distributions, correlations, and potability using suitable tools. (cognos)

- **Predictive Modelling:**
  Decide on the machine learning algorithms and features to use for predicting water potability.

- **Conclusion:**
  Finally we give a wonderful dashboard with some visualization to easily understand the quality of water.

DEVELOPMENT PHASES

Step 1: Import Necessary Libraries

Make sure you have Python and the Pandas library installed. You can install Pandas using pip if it's not already installed:

Program:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Step 2: Load the Dataset

Assuming you have downloaded the dataset as a CSV file, you can load it into a Pandas DataFrame like this:

Program:

```
# Load the dataset
d = pd.read_csv('/content/water_potability.csv')
```

Step 3: Data manipulation using pandas

Data Manipulation: Data Manipulation is one of the initial processes done in Data Analysis. It involves arranging or rearranging data points to make it easier for users/data analysts to perform necessary insights or business directives.

Pandas: Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data.

Program:

```
d.head()
```

Output:

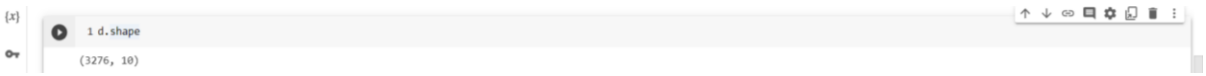
	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	0
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180013	56.329076	4.500656	0
2	8.099124	224.236259	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

Program:

```
d.shape
```

Output:

```
(3276,10)
```



Program:

```
d.isnull().sum()
```

Output:

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

Program:

```
d.describe()
```

Output:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
count	2785.000000	3276.000000	3276.000000	3276.000000	2495.000000	3276.000000	3276.000000	3114.000000	3276.000000	3276.000000
mean	7.080795	196.369496	22014.092526	7.122277	333.775777	426.205111	14.284970	66.396293	3.966786	0.390110
std	1.594320	32.879761	8768.570828	1.583085	41.416840	80.824064	3.308162	16.175008	0.780382	0.487849
min	0.000000	47.432000	320.942611	0.352000	129.000000	181.483754	2.200000	0.738000	1.450000	0.000000
25%	6.093092	176.850538	15666.690297	6.127421	307.699498	365.734414	12.065801	55.844536	3.439711	0.000000
50%	7.036752	196.967627	20927.833607	7.130299	333.073546	421.884968	14.218338	66.622485	3.955028	0.000000
75%	8.062066	216.667456	27332.762127	8.114887	359.950170	481.792304	16.557652	77.337473	4.500320	1.000000
max	14.000000	323.124000	61227.196008	13.127000	481.030642	753.342620	28.300000	124.000000	6.739000	1.000000

Program:

```
d.info()
```

Output:

```
<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
```

Program:

```
d.fillna(d.mean(),inplace=True)

d.isnull().sum()
```

Output:

```
ph 491
Hardness 0
Solids 0
Chloramines 0
Sulfate 781
Conductivity 0
Organic_carbon 0
```

```
Trihalomethanes 162
Turbidity 0
Potability 0
dtype: int64
```

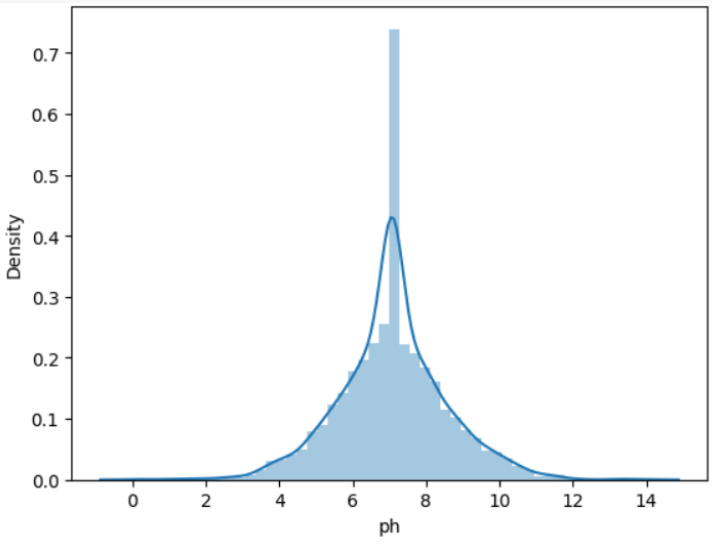
Step 4: Visualization

Use data visualization libraries such as Matplotlib and Seaborn to create visualizations that help convey your findings.

Program:

```
sns.distplot(d['ph'])
```

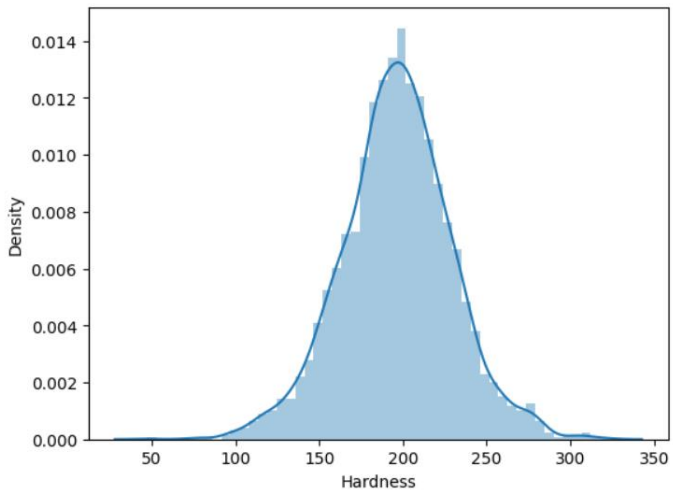
Output:



Program:

```
sns.distplot(d['Hardness'])
```

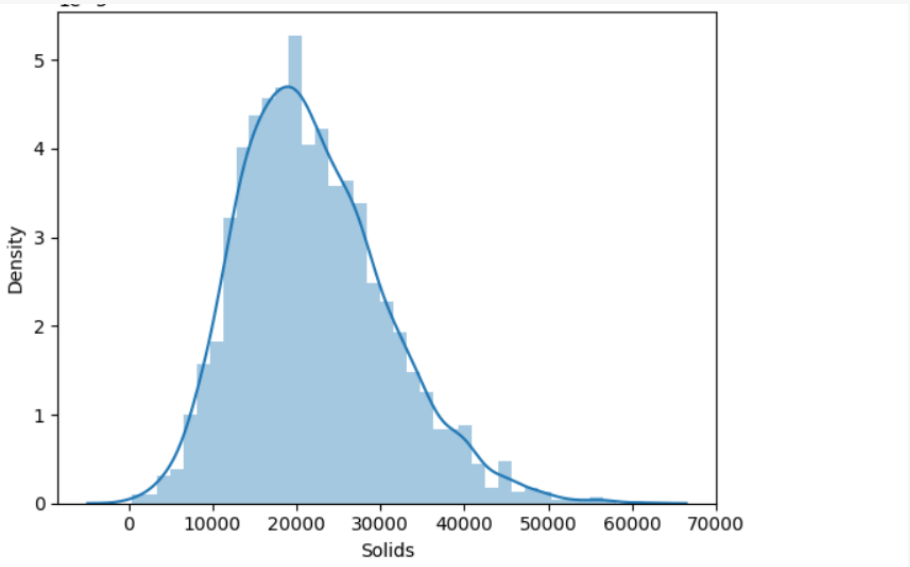
Output:



Program:

```
sns.distplot(d['Solids'])
```

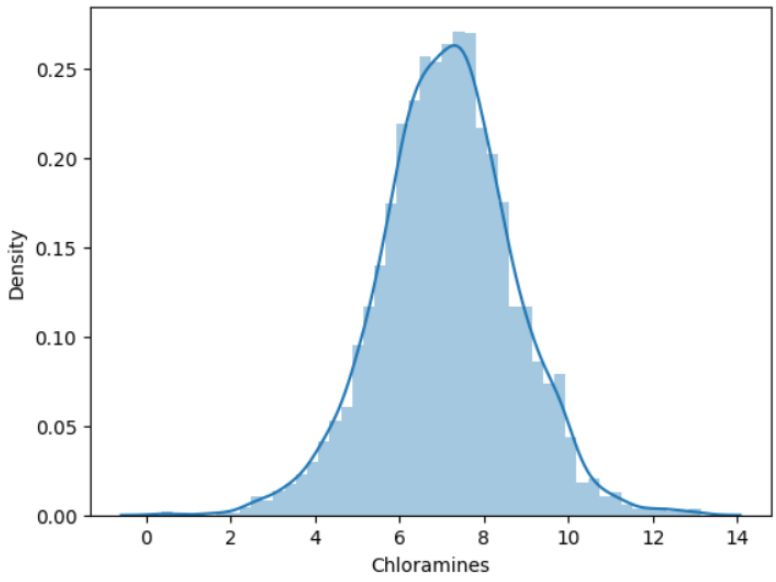
Output:



Program:

```
sns.distplot(d['Chloramines'])
```

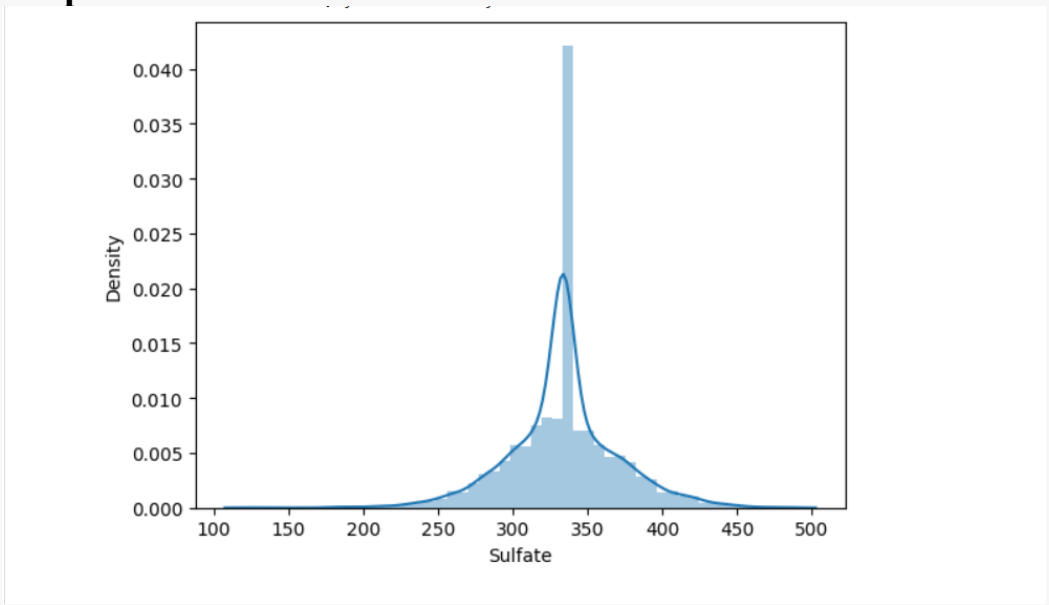
Output:



Program:

```
sns.distplot(d['Sulfate'])
```

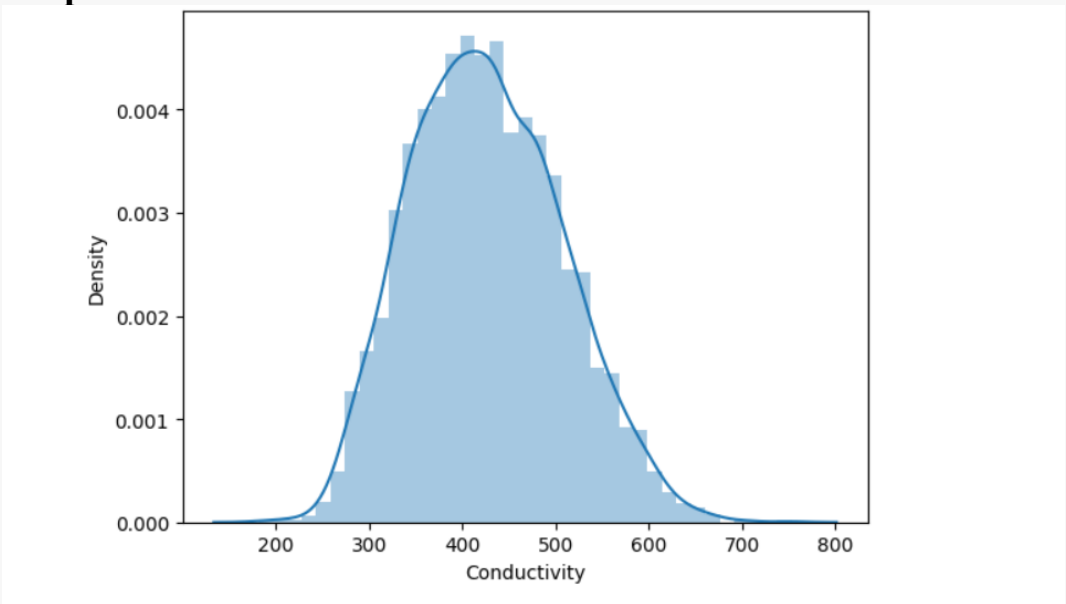
Output:



Program:

```
sns.distplot(d['Conductivity'])
```

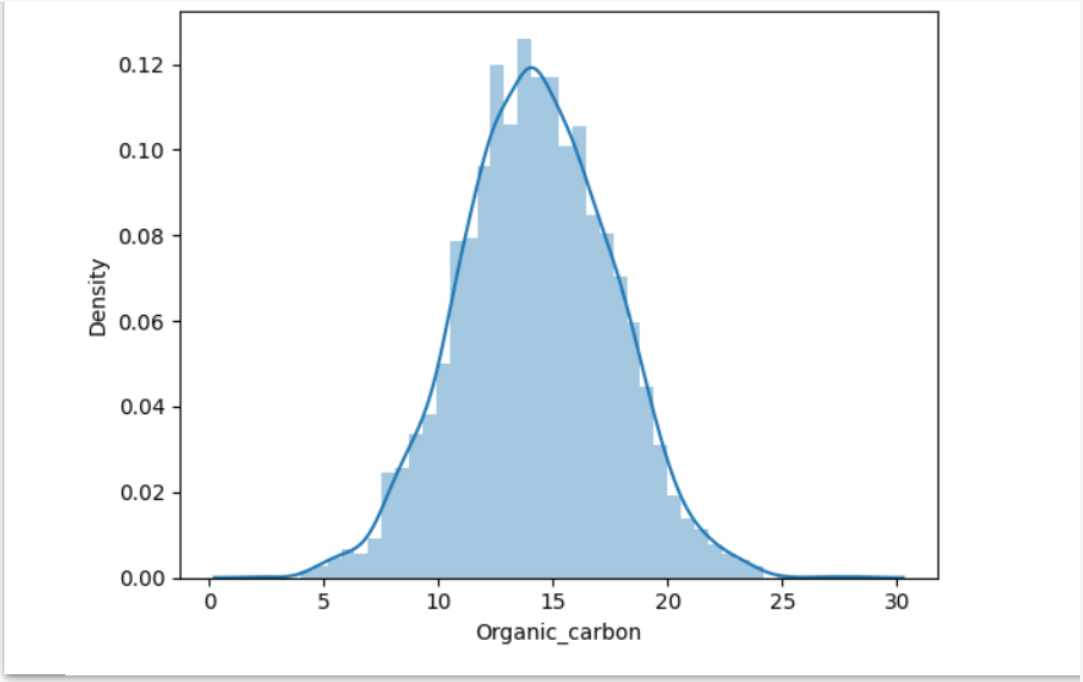
Output:



Program:

```
sns.distplot(d['Organic_carbon'])
```

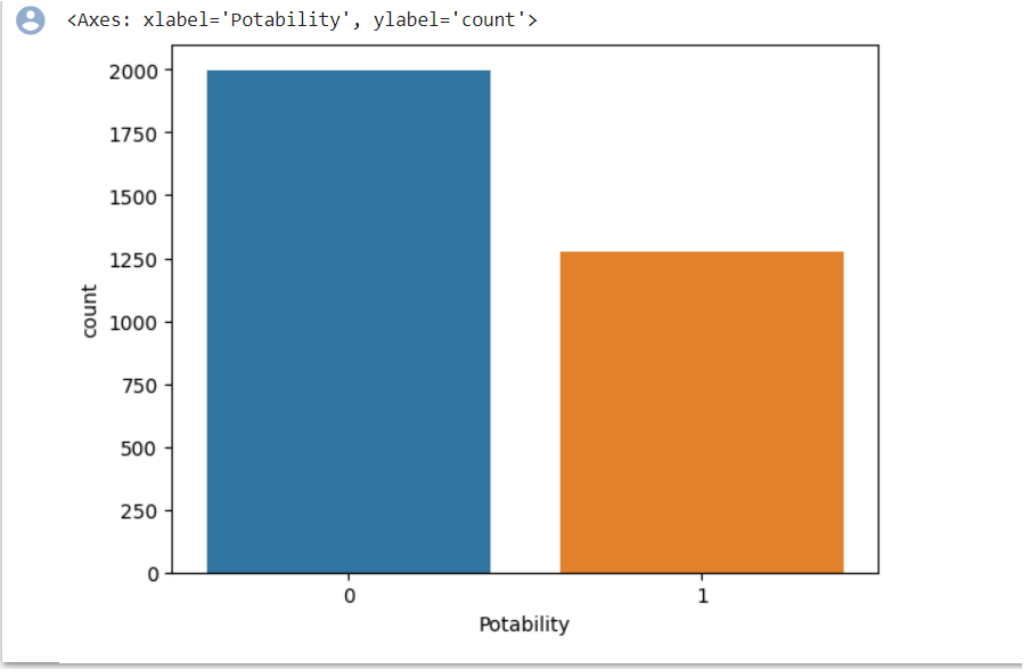
Output:



Program:

```
sns.distplot(d['Trihalomethanes'])
sns.distplot(d['Turbidity'])
sns.countplot(x='Potability',data=d)
```

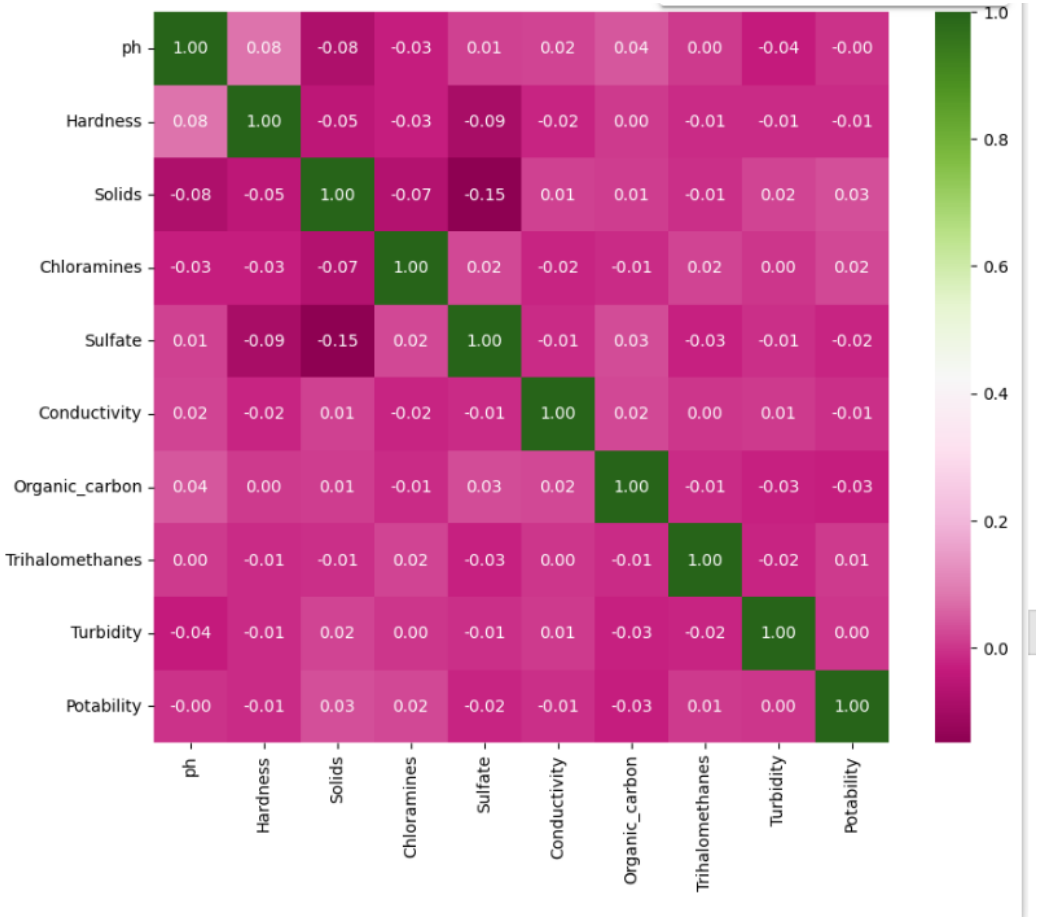
Output:



Program:

```
correlation_matrix = d.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='PiYG',fmt='.2f')
plt.show()
```

Output:

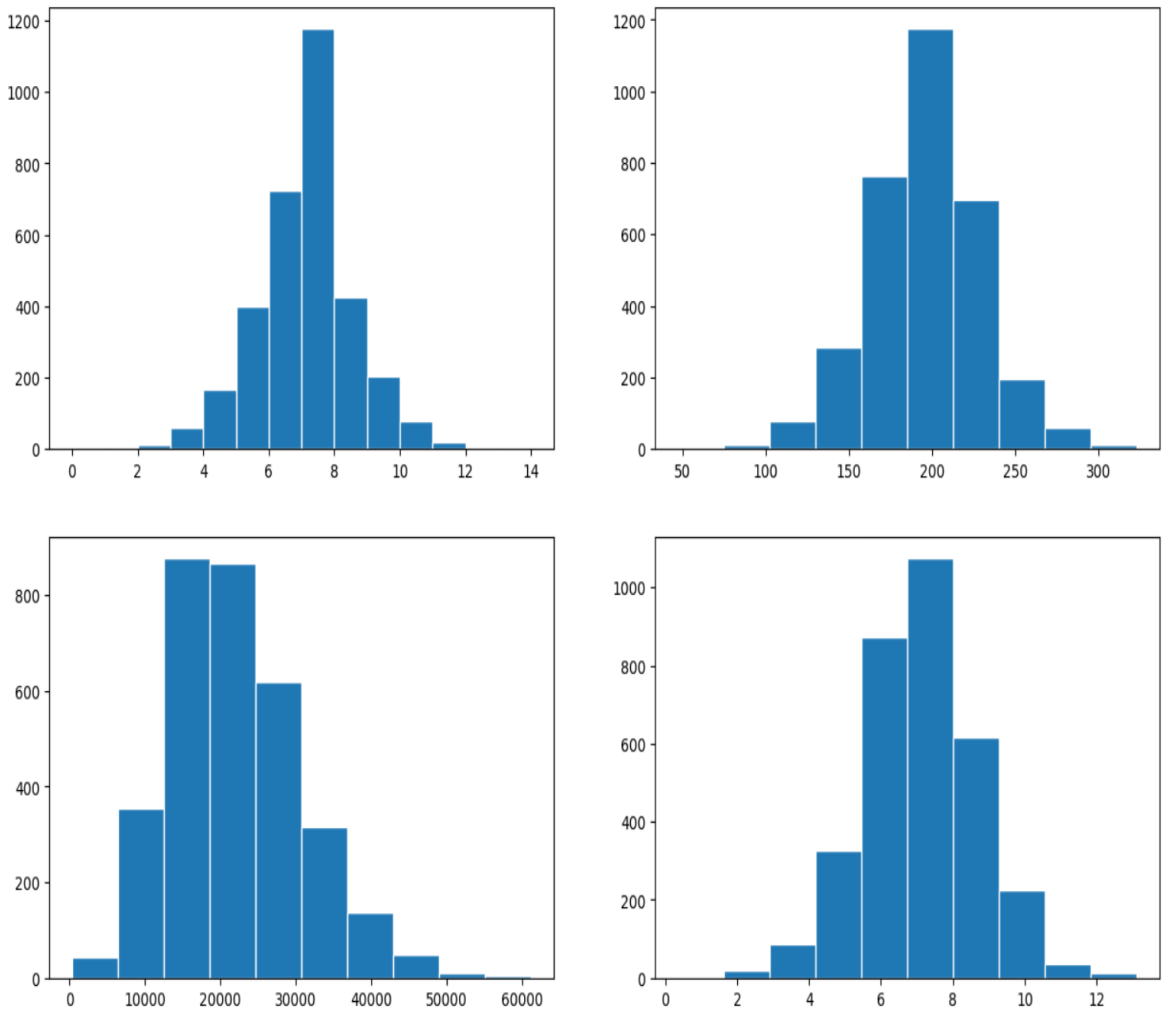


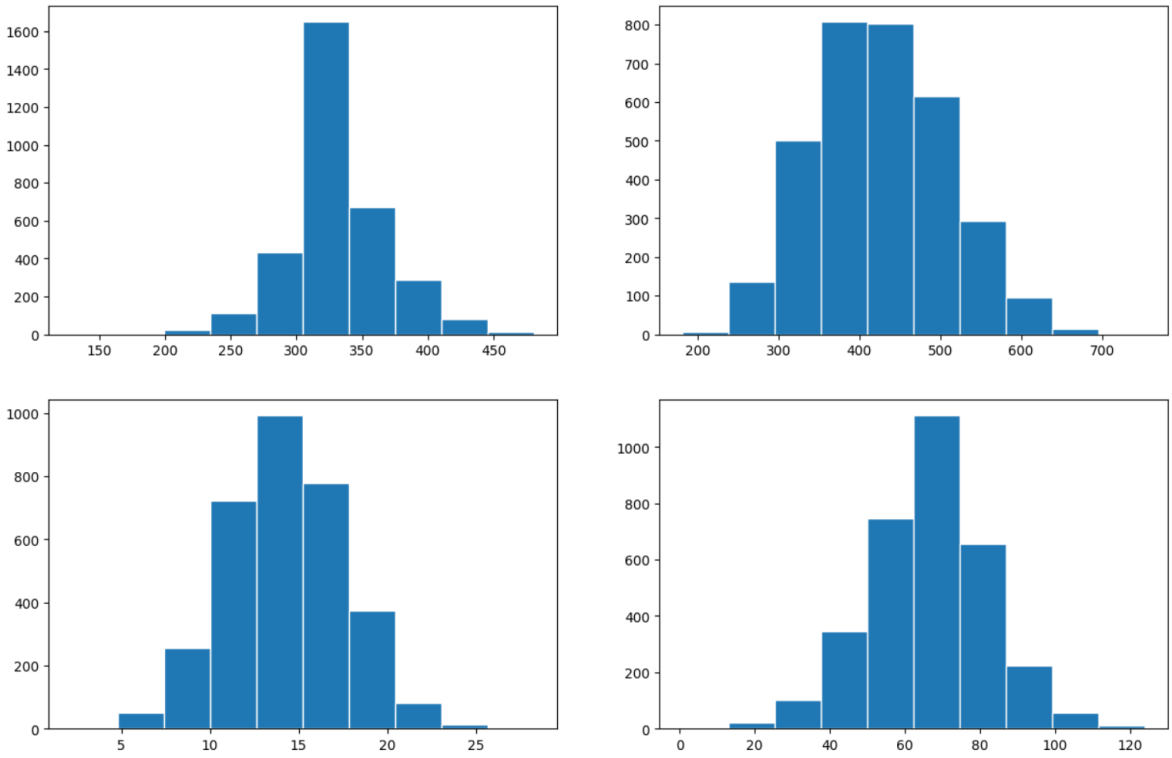
Program:

```
import matplotlib.pyplot as plt
import numpy as np
plt.figure(figsize=(15, 20))
#plot 1:
plt.subplot(4, 2, 1)
plt.hist(d.ph , bins=14,align='mid', edgecolor='w')
#plot 2:
plt.subplot(4, 2, 2)
plt.hist(d.Hardness , bins=10,align='mid', edgecolor='w')
#plot 3:
plt.subplot(4, 2, 3)
```

```
plt.hist(d.Solids , bins=10,align='mid', edgecolor='w')
#plot 4:
plt.subplot(4, 2, 4)
plt.hist(d.Chloramines , bins=10,align='mid', edgecolor='w')
#plot 5:
plt.subplot(4, 2, 5)
plt.hist(d.Sulfate , bins=10,align='mid', edgecolor='w')
#plot 6:
plt.subplot(4, 2, 6)
plt.hist(d.Conductivity , bins=10,align='mid', edgecolor='w')
#plot 7:
plt.subplot(4, 2, 7)
plt.hist(d.Organic_carbon , bins=10,align='mid', edgecolor='w')
#plot 8:
plt.subplot(4, 2, 8)
plt.hist(d.Trihalomethanes , bins=10,align='mid', edgecolor='w')
plt.show()
```

Output:



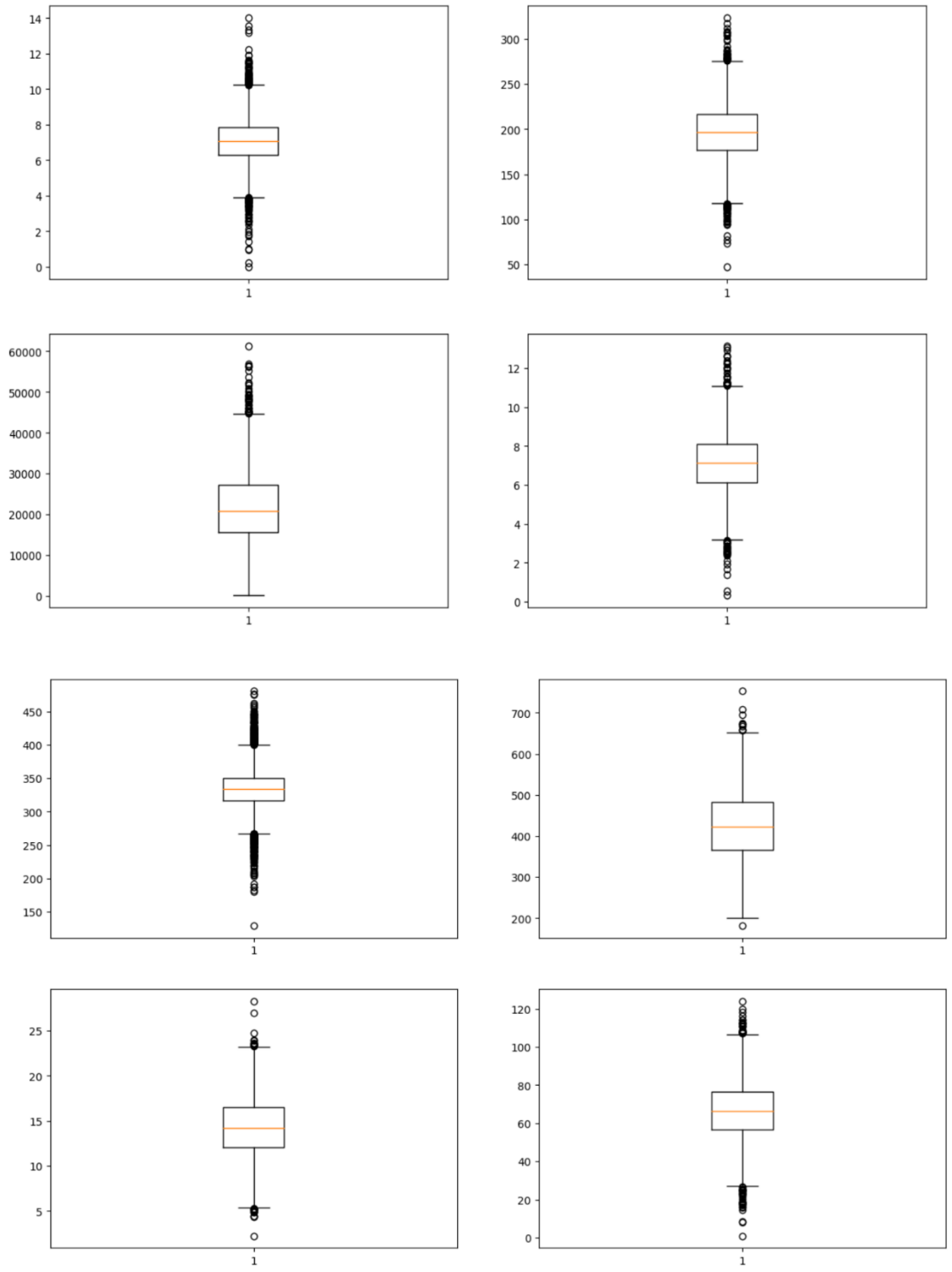


Program:

```
import matplotlib.pyplot as plt
import numpy as np
plt.figure(figsize=(15, 20))
#plot 1:
plt.subplot(4, 2, 1)
plt.boxplot(d.ph )
#plot 2:
plt.subplot(4, 2, 2)
plt.boxplot(d.Hardness)
#plot 3:
plt.subplot(4, 2, 3)
plt.boxplot(d.Solids )
#plot 4:
plt.subplot(4, 2, 4)
plt.boxplot(d.Chloramines)
#plot 5:
plt.subplot(4, 2, 5)
plt.boxplot(d.Sulfate )
#plot 6:
plt.subplot(4, 2, 6)
plt.boxplot(d.Conductivity)
#plot 7:
plt.subplot(4, 2, 7)
plt.boxplot(d.Organic_carbon )
```

```
#plot 8:
plt.subplot(4, 2, 8)
plt.boxplot(d.Trihalomethanes)
plt.show()
```

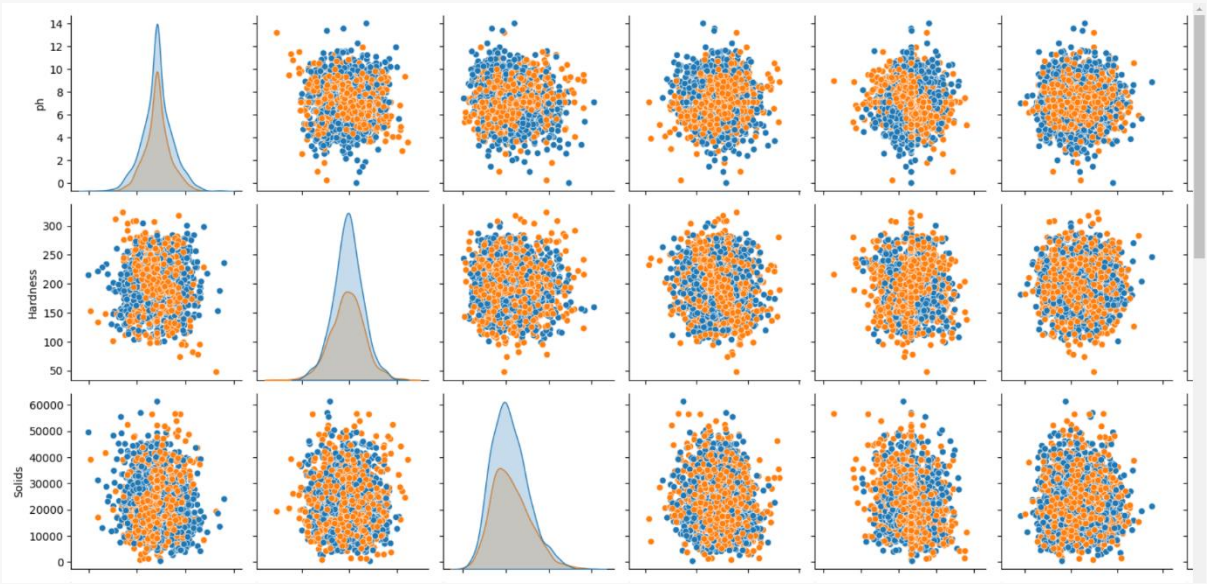
Output:

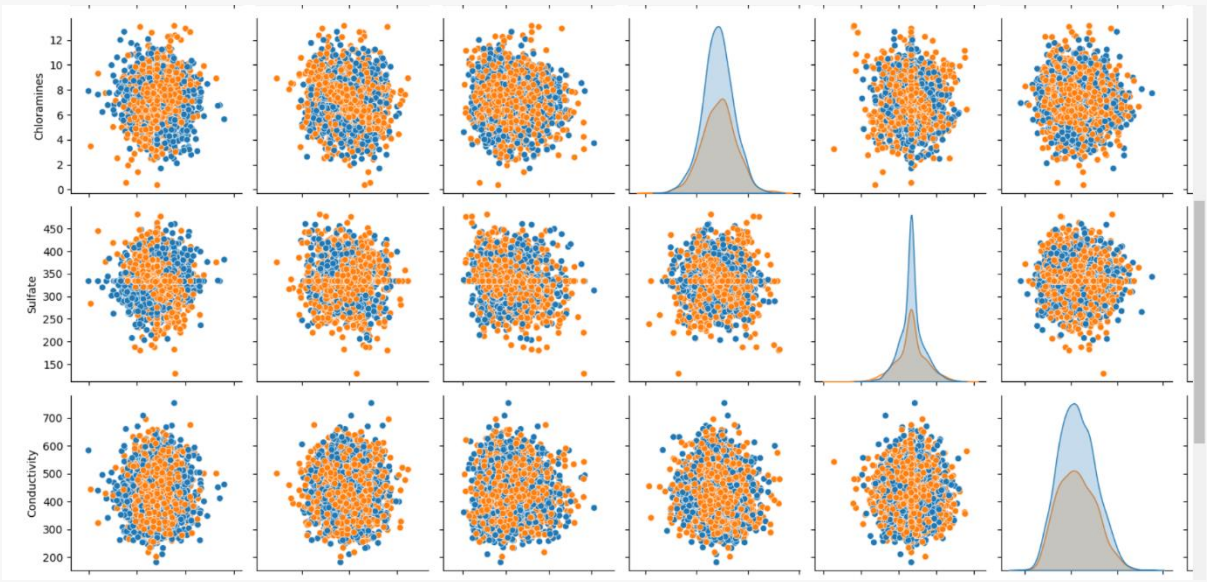


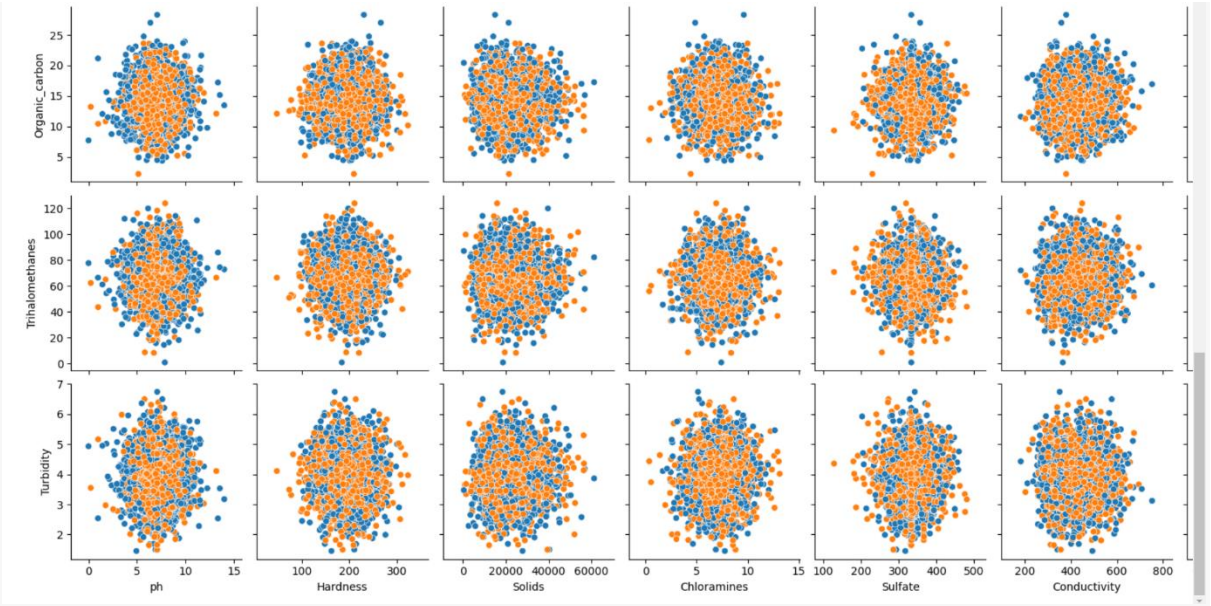
Program:

```
# Create scatter plots
sns.pairplot(d, hue='Potability')
plt.show()
```

Output:



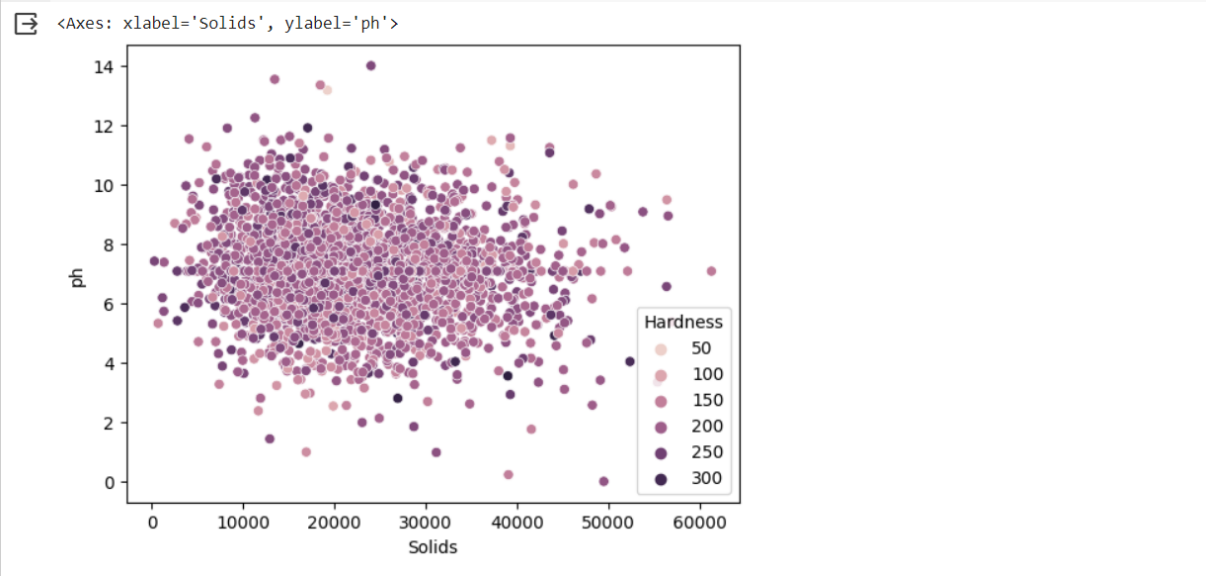




Program:

```
sns.scatterplot(data=d,x="Solids",y="ph",hue="Hardness")
```

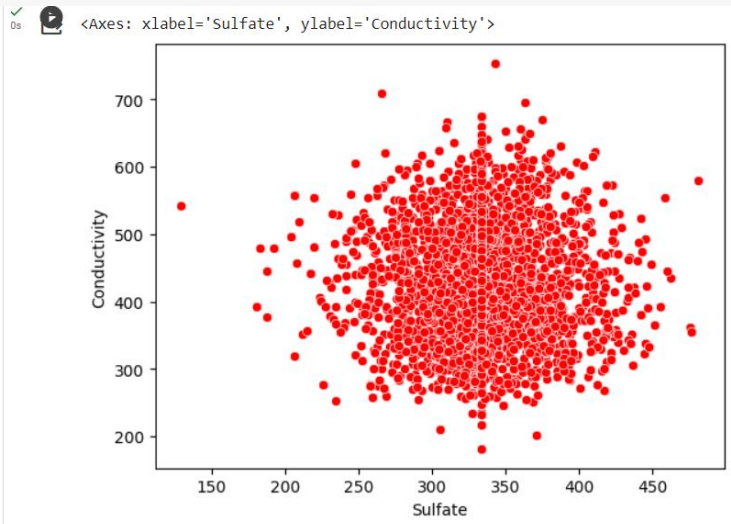
Output:



Program:

```
sns.scatterplot(data=d,x="Sulfate",y="Conductivity",c="red")
```

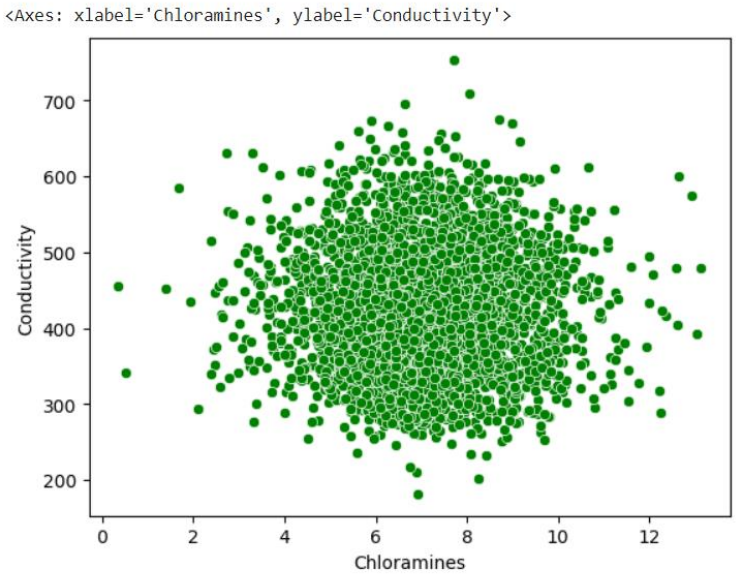
Output:



Program:

```
sns.scatterplot(data=d,x="Chloramines",y="Conductivity",c="green")
```

Output:



Step 5: Preprocessing

Data preprocessing may involve various tasks, such as handling missing values, data cleaning, and feature engineering.

detect missing and null values and filling

Program:

```
d.isnull().sum()
```

Output:

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

Program:

```
d.fillna(d.mean(),inplace=True)
d.isnull().sum()
```

Output:

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

Outlier detection and handling

Detection of outliers isn't a trivial problem. We're trying to identify observations that don't fit into the general characteristics of the dataset.

Detection and handling of outliers is a fundamental problem in data science and machine learning. Solving this problem isn't straightforward, just like the missing values problem.

Outlier:Solids

Program:

```
l=d.Solids
q1=l.quantile(0.25)
q2=l.quantile(0.75)
iqr=q2-q1
low=q1-1.5*iqr
up=q2+1.5*iqr
```

```
l[(l<low) | (l>up)]
```

Output:

142 46140.126850
186 45222.506665
283 48621.563952
287 46113.957485
366 52318.917298
378 45249.449033
405 46931.884293
516 45510.584319
546 49074.730407
613 44868.458368
666 52060.226800
987 48002.084596
1031 56867.859236
1068 55334.702799
1077 48410.471014
1096 45141.686036
1186 56351.396304
1302 44896.984112
1332 45166.912141
1343 48204.172192
1445 45166.639389
1462 45939.689158
1527 46718.555965
1554 56488.672413
1556 45243.028953
1746 49125.360084
1784 50279.262429
1815 45041.149139
1858 46077.358485
1955 49009.924656
1984 47022.745845
2012 47852.888871

```
2497 49341.421279
2602 61227.196008
2680 48175.852093
2758 47591.283183
2891 45050.002276
2993 45148.808118
3014 49456.587108
3062 44982.734155
3130 50793.898917
3150 56320.586979
3162 53735.899194
3190 50166.533323
3226 51731.820553
3236 48007.868134
3271 47580.991603
Name: Solids, dtype: float64
```

Program:

```
d.Solids[d.Solids>up]=low
l[(l<low) | (l>up)]
```

Output:

```
Series([], Name: Solids, dtype: float64)
```

Outlier: ph

Program:

```
l=d.ph
q1=l.quantile(0.25)
q2=l.quantile(0.75)
iqr=q2-q1
low=q1-1.5*iqr
up=q2+1.5*iqr

l[(l<low) | (l>up)]
```

Output:

```
1      3.716080
9     11.180284
26     3.445062
32    10.433291
36     3.641630
      ...
3233  10.485604
3246  10.667364
3249  10.808157
```

3261 3.629922
3269 11.491011
Name: ph, Length: 142, dtype: float64

Program:

```
d.ph[d.ph>up]=low
```

```
l[(l<low) | (l>up)].count()
```

Output:

64

Outlier:Hardness

Program:

```
l=d.Hardness
q1=l.quantile(0.25)
q2=l.quantile(0.75)
iqr=q2-q1
low=q1-1.5*iqr
up=q2+1.5*iqr
```

```
l[(l<low) | (l>up)]
```

Output:

```
18 279.357167
37 304.235912
51 100.457615
67 103.464759
71 116.299330
    ...
3218 287.975540
3221 303.702627
3230 114.807578
3244 277.065713
3269 94.812545
Name: Hardness, Length: 83, dtype: float64
```

Program:

```
d.Hardness[d.Hardness>up]=low
```

```
l[(l<low) | (l>up)]
```

Output:

```
51 100.457615
```

```
67 103.464759
71 116.299330
149 104.752425
189 112.299485
215 105.859264
227 112.820254
246 103.173587
258 98.771644
263 47.432000
278 81.710895
309 113.831112
317 77.459586
335 94.091307
379 73.492234
1525 108.699077
1536 106.380113
1542 97.280909
1649 98.452931
1650 116.061950
1743 107.383327
1815 116.725122
1829 113.504698
1858 115.392979
1859 117.057314
1868 110.865788
2044 108.916629
2343 100.806520
2383 107.341982
2400 111.246412
2418 116.905479
2606 116.338278
2648 113.024472
2715 111.478582
2742 113.175965
2775 114.463900
2777 111.994028
2811 114.733545
2877 114.371450
3179 94.908977
3186 110.903598
3217 98.367915
3230 114.807578
3269 94.812545
Name: Hardness, dtype: float64
```

Outlier:Chloramines

Program:

```
l=d.Chloramines
q1=l.quantile(0.25)
q2=l.quantile(0.75)
iqr=q2-q1
low=q1-1.5*iqr
up=q2+1.5*iqr
```

```
l[(l<low) | (l>up)]
```

Output:

```
272 12.580026
275 13.043806
285 0.530351
287 12.912187
304 12.363285
      ...
2677 2.562255
2694 12.653362
2699 11.996015
2714 2.648390
2796 3.016033
Name: Chloramines, Length: 61, dtype: float64
```

Program:

```
d.Chloramines[d.Chloramines>up]=low
```

```
l[(l<low) | (l>up)]
```

Output:

```
285 0.530351
408 2.484380
437 2.981379
454 2.993744
509 1.683993
651 2.750837
666 2.577555
772 2.866073
806 2.862535
1106 2.741712
1698 2.456014
1776 3.139553
1868 2.621268
2206 2.458609
2212 2.785718
2300 2.498597
2302 0.352000
2326 3.117441
2346 2.855790
2350 1.390871
2352 2.397985
2401 1.920271
2449 2.386653
2470 2.102691
2511 3.124833
2535 2.726766
2566 2.654491
2586 3.074316
2677 2.562255
2714 2.648390
2796 3.016033
Name: Chloramines, dtype: float64
```

Outlier:Sulfate

Program:

```
l=d.Sulfate
q1=l.quantile(0.25)
q2=l.quantile(0.75)
iqr=q2-q1
low=q1-1.5*iqr
up=q2+1.5*iqr
```

```
l[(l<low) | (l>up)]
```

Output:

```
9 404.041635
51 247.200826
84 266.908428
112 426.543590
118 241.607532
...
3115 407.417977
3116 413.914001
3162 254.040977
3176 264.052839
3269 258.930600
Name: Sulfate, Length: 264, dtype: float64
```

Program:

```
d.Sulfate[d.Sulfate>up]=low
```

```
l[(l<low) | (l>up)]
```

Output:

```
51 247.200826
84 266.908428
118 241.607532
130 240.936722
199 261.444798
...
3023 238.844004
3082 267.026093
3162 254.040977
3176 264.052839
3269 258.930600
Name: Sulfate, Length: 129, dtype: float64
```

Outlier:Conductivity

Program:

```
l=d.Conductivity
q1=l.quantile(0.25)
q2=l.quantile(0.75)
iqr=q2-q1
low=q1-1.5*iqr
up=q2+1.5*iqr

l[(l<low) | (l>up)]
```

Output:

```
66 669.725086
140 672.556999
342 695.369528
1183 656.924128
1269 660.254946
1295 666.690618
1384 181.483754
2134 708.226364
2704 753.342620
2737 657.570422
3142 674.443476
Name: Conductivity, dtype: float64
```

Program:

```
d.Conductivity[d.Conductivity>up]=low

l[(l<low) | (l>up)]
```

Output:

```
1384   181.483754
Name: Conductivity, dtype: float64
```

Outlier:Organic_carbon

Program:

```
l=d.Organic_carbon
q1=l.quantile(0.25)
```

```
q2=l.quantile(0.75)
iqr=q2-q1
low=q1-1.5*iqr
up=q2+1.5*iqr
l[(l<low) | (l>up)]
```

Output:

```
43 23.917601
227 23.399516
333 5.315287
420 23.373265
558 23.514774
698 23.569645
785 2.200000
876 4.966862
1055 23.952450
1390 4.371899
1447 4.861631
1536 5.218233
1663 4.473092
1689 5.051695
1792 28.300000
1827 23.317699
2057 24.755392
2082 5.188466
2138 4.902888
2224 4.466772
2236 27.006707
2414 5.196717
2601 23.667667
2680 5.159380
3169 23.604298
Name: Organic_carbon, dtype: float64
```

Program:

```
d.Organic_carbon[d.Organic_carbon>up]=low

l[(l<low) | (l>up)]
```

Output:

```
333 5.315287
785 2.200000
876 4.966862
1390 4.371899
1447 4.861631
```

```
1536 5.218233
1663 4.473092
1689 5.051695
2082 5.188466
2138 4.902888
2224 4.466772
2414 5.196717
2680 5.159380
Name: Organic_carbon, dtype: float64
```

Outlier:Trihalomethanes

Program:

```
l=d.Trihalomethanes
q1=l.quantile(0.25)
q2=l.quantile(0.75)
iqr=q2-q1
low=q1-1.5*iqr
up=q2+1.5*iqr

l[(l<low) | (l>up)]
```

Output:

```
61 17.915723
128 110.739299
133 23.817020
204 25.525267
245 18.400012
284 107.754043
330 8.175876
350 112.622733
356 107.189584
374 20.337753
377 26.505484
518 23.792950
531 120.030077
630 25.057375
698 19.175175
719 21.355275
951 18.101222
1038 107.900842
1041 23.136611
1053 107.282329
1075 17.000683
1081 108.849568
1123 8.577013
```

```
1156 116.161622
1157 118.357275
1316 114.208671
1337 26.140863
1360 16.291505
1630 17.527765
1699 22.749735
1767 22.219327
1864 108.589414
1876 110.431080
1893 107.610806
1962 113.048886
2006 24.532773
2077 107.585967
2099 112.061027
2121 15.684877
2256 112.412210
2353 18.015272
2376 124.000000
2402 24.914971
2490 25.061904
2585 24.734227
2689 108.265227
2742 23.075806
2949 14.343161
3010 111.115310
3035 0.738000
3101 107.306343
3102 111.595448
3165 108.213981
3184 114.034946
Name: Trihalomethanes, dtype: float64
```

Program:

```
d.Trihalomethanes[d.Trihalomethanes>up]=low

l[(l<low) | (l>up)]
```

Output:

```
61 17.915723
133 23.817020
204 25.525267
245 18.400012
330 8.175876
374 20.337753
377 26.505484
```

```
518 23.792950
630 25.057375
698 19.175175
719 21.355275
951 18.101222
1041 23.136611
1075 17.000683
1123 8.577013
1337 26.140863
1360 16.291505
1630 17.527765
1699 22.749735
1767 22.219327
2006 24.532773
2121 15.684877
2353 18.015272
2402 24.914971
2490 25.061904
2585 24.734227
2742 23.075806
2949 14.343161
3035 0.738000
Name: Trihalomethanes, dtype: float64
```

Outlier: Turbidity

Program:

```
l=d.Turbidity
q1=l.quantile(0.25)
q2=l.quantile(0.75)
iqr=q2-q1
low=q1-1.5*iqr
up=q2+1.5*iqr

l[(l<low) | (l>up)]
```

Output:

```
119 6.204846
382 6.494249
492 6.739000
593 1.680554
789 1.812529
990 6.357439
1073 6.389161
1290 1.496101
1682 1.687625
```

```
1892 1.492207
1927 1.659799
2377 6.226580
2554 6.099632
2724 1.641515
2757 6.307678
2796 1.844372
2921 6.494749
3042 1.450000
3166 1.801327
Name: Turbidity, dtype: float64
```

Program:

```
d.Turbidity[d.Turbidity>up]=low

l[(l<low) | (l>up)]
```

Output:

```
593 1.680554
789 1.812529
1290 1.496101
1682 1.687625
1892 1.492207
1927 1.659799
2724 1.641515
2796 1.844372
3042 1.450000
3166 1.801327
Name: Turbidity, dtype: float64
```

Step 6: Data spliting

Data splitting is when data is divided into two or more subsets. Typically, with a two-part split, one part is used to evaluate or test the data and the other to train the model. Data splitting is an important aspect of data science, particularly for creating models based on data.

Program:

```
from sklearn.model_selection import train_test_split
X = d.drop('Potability', axis=1)
y = d['Potability']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42)
```

Step 7: Model Fitting and Evaluating

Model fitting is a measure of how well a machine learning model generalizes to similar data to that on which it was trained. A model that is well-fitted produces more accurate outcomes. A model that is overfitted matches the data too closely. A model that is underfitted doesn't match closely enough.

Program:

```
#logistic regression model
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
```

```
from sklearn.metrics import
accuracy_score,confusion_matrix,precision_score
accuracy_lr = accuracy_score(y_test, y_pred)*100
accuracy_lr
```

Output:

```
62.27106227106227
```

Program:

```
#DecisionTree Classifier model
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier(criterion= 'gini', min_samples_split= 10,
splitter= 'best')
dt.fit(X_train,y_train)
prediction=dt.predict(X_test)

from sklearn.metrics import
accuracy_score,confusion_matrix,precision_score
accuracy_dt=accuracy_score(y_test,prediction)*100
accuracy_dt
```

Output:

```
56.043956043956044
```

Program:

```
#RandomForest Classifier model
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)

from sklearn.metrics import classification_report, accuracy_score,
roc_curve, roc_auc_score
accuracy_rf = accuracy_score(y_test, y_pred)*100
accuracy_rf
```

Output:

```
66.17826617826617
```

Step 8: Sample testing

The one sample t-test is a statistical procedure used to determine whether a sample of observations could have been generated by a process with a specific mean. Suppose you are interested in determining whether an assembly line produces laptop computers that weigh five pounds.

Program:

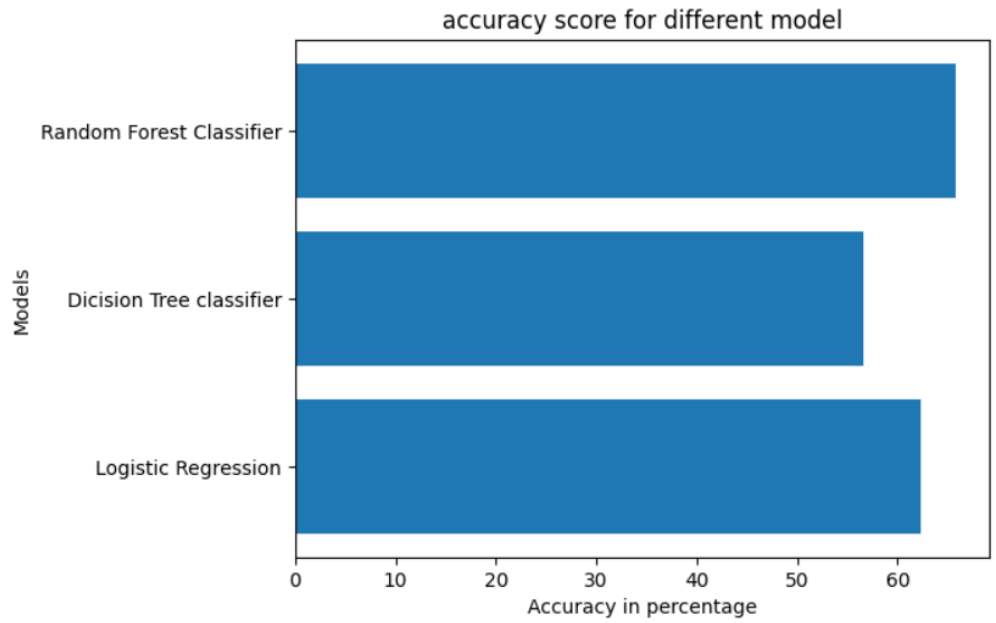
```
#logistic regression model
lrm=model.predict([[5.735724,
158.318741,25363.016594,7.728601,377.543291,568.304671,13.626624,75.952
337,4.732954]])
lrm

#DecisionTree Classifier model
dtm=dt.predict([[5.735724,
158.318741,25363.016594,7.728601,377.543291,568.304671,13.626624,75.952
337,4.732954]])
dtm

#RandomForest Classifier model
rfm=rf_model.predict([[5.735724,
158.318741,25363.016594,7.728601,377.543291,568.304671,13.626624,75.952
337,4.732954]])
rfm
```

Step 9: Accuracy score graph for different model

Program:

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
xpoints = np.array(['Logistic Regression','Dicision Tree
classifier','Random Forest Classifier'])
y=np.array([62.27106227106227,56.65445665445665,65.8119658119658])
plt.barh(xpoints,y)
plt.title("accuracy score for different model")
plt.ylabel("Models")
plt.xlabel("Accuracy in percentage")
plt.show()
```

Output:



Step 10: Predictive model

Program:

```
def model():
  ph=float(input("Enter ph value : "))
  hardness=float(input("Enter Hardness value : "))
  solids=float(input("Enter Solids value : "))
  chloramines=float(input("Enter Chloramines value : "))
  sulfate=float(input("Enter Sulfate value : "))
  conductivity=float(input("Enter Conductivity value : "))
  organic_carbon=float(input("Enter Organic_carbon value : "))
  trihalomethanes=float(input("Enter Trihalomethanes value : "))
  turbidity=float(input("Enter Turbidity value : "))
  x=np.array([ph,hardness,solids,chloramines,sulfate,conductivity,organ
ic_carbon,trihalomethanes,turbidity])
  x=x.reshape(1,-1)
  y=dt.predict(x)
  if y[0]==1:
    print("\033[1m "+"This Water Is Not Drinkable"+"\033[0;0m")
  else:
    print("\033[1m "+"This Water Is  Drinkable"+"\033[0;0m")
```

```
model()
```

Output:

Enter ph value : 5.735724
Enter Hardness value : 158.318741
Enter Solids value : 25363.016594
Enter Chloramines value : 7.728601
Enter Sulfate value : 377.543291
Enter Conductivity value : 568.304671
Enter Organic_carbon value : 13.626624
Enter Trihalomethanes value : 75.952337
Enter Turbidity value : 4.732954

This Water Is Drinkable

ABOUT DATASET

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 a 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 an unwanted taste and diluted color in the 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.
- The Desired limit for TDS is 500 mg/l and maximum limit is 1000 mg/l which is prescribed for drinking purposes.

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 it'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 exceed 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 the US EPA < 2 mg/L as TOC in treated / drinking water, and < 4 mg/Lit in source water which is used 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.
- Water is not safe to drink and (1) Water is safe to drink.