Water Analysis

The aim of this project is to train machine learning models and use it to predict the quality of water for usage using its properties..

Context

Water is the most significant resource of life, crucial for supporting the life of most existing creatures and all human beings. Living organisms need water with enough quality to continue their lives. There are certain limits of pollutions that water species can tolerate. Exceeding these limits affect the existence of these creatures and threaten their lives.

Most ambient water bodies such as rivers, lakes, and streams have specific quality standards that indicate their quality. Moreover, water specifications for other applications/usages also possess their standards. For example, irrigation water must be neither too saline nor contain toxic materials that can be transferred to plants or soil in order not to destroy ecosystems. Water quality for industrial uses also require different chemical properties based on required industrial processes. Some of the low-priced resources of fresh water, such as ground and surface water, are natural water resources. However, such resources can be polluted by human/industrial activities and other natural processes.

Access to safe drinking-water is essential to health, a basic human right and a component of effective policy for health protection. This is important as a health and development issue at a national, regional and local level. In some regions, it has been shown that investments in water

supply and sanitation can yield a net economic benefit, since the reductions in adverse health effects and health care costs outweigh the costs of undertaking the interventions.

Data

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

- ***1. pH value:*** PH is an important parameter in evaluating the acid–base balance of water. It is also the indicator of acidic or alkaline condition of water status. WHO has recommended maximum permissible limit of pH from 6.5 to 8.5. The current investigation ranges were 6.52–6.83 which are in the range of WHO standards.
- ***2. Hardness:*** Hardness is mainly caused by calcium and magnesium salts. These salts are dissolved from geological/rock deposits through which water travels. The length of time water is in contact with a hardness producing material helps determines its level of hardness. Hardness was originally defined as the capacity of water to precipitate soap caused by Calcium and Magnesium.
- ***3. Solids (Total dissolved solids - TDS):*** Water has the ability to dissolve a wide range of inorganic and some organic minerals or salts such as potassium, calcium, sodium, bicarbonates, chlorides, magnesium, sulfates etc. These minerals produced un-wanted taste and diluted color in appearance of water. This is the important parameter for the use of water. The water with high TDS value indicates that water is highly mineralized. Desirable limit for TDS is 500 mg/l and maximum limit is 1000 mg/l which prescribed for drinking purpose.
- ***4. Chloramines:*** Chlorine and chloramine are the major disinfectants used in public water systems. Chloramines are most commonly formed when ammonia is added to chlorine to treat drinking water. Chlorine levels up to 4 milligrams per liter (mg/L or 4 parts per million (ppm)) are considered safe in drinking water.
- ***5. Sulfate:*** Sulfates are naturally occurring substances that are found in minerals, soil and rocks. They are present in ambient air, groundwater, plants and food. The principal commercial use of sulfate is in the chemical industry. Sulfate concentration in seawater is about 2,700 milligrams per liter (mg/L). It ranges from 3 to 30 mg/L in most freshwater supplies, although much higher concentrations (1000 mg/L) are found in some geographic locations.
- ***6. Conductivity:*** Pure water is not a good conductor of electric current rather’s a good insulator. Increase in ions concentration enhances the electrical conductivity of water. Generally, the amount of dissolved solids in water determines the electrical conductivity. Electrical conductivity (EC) actually measures the ionic process of a solution that enables it to transmit current. According to WHO standards, EC value should not exceeded 400 µS/cm.
- ***7. Organic_carbon:*** Total Organic Carbon (TOC) in sources water come from decaying natural organic matter (NOM) as well as synthetic sources. TOC is a measure of the total amount of carbon in organic compounds in pure water. According to US EPA < 2 mg/L as TOC in treated / drinking water, and < 4 mg/Lit in source water which is use for treatment.
- ***8. Trihalomethanes:*** THMs are chemicals which may be found in water treated with chlorine. The concentration of THMs in drinking water varies according to the level of organic material in the water, the amount of chlorine required to treat the water, and the temperature of the water that is being treated. THM levels up to 80 ppm is considered safe in drinking water.
- ***9. Turbidity:*** The turbidity of water depends on the quantity of solid matter present in the suspended state. It is a measure of light emitting properties of water and the test is used to indicate the quality of waste discharge with respect to colloidal matter. The mean turbidity

value obtained for Wondo Genet Campus (0.98 NTU) is lower than the WHO recommended value of 5.00 NTU.

- ***10. Potability:*** Indicates if water is safe for human consumption where 1 means Potable and 0 means Not potable. i.e (0) Water is not safe to drink and (1) Water is safe to drink

Import libraries and data.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [33]:
```
df = pd.read_csv('water_potability.csv')
df.head()
```

Out[33]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trih
0	NaN	204.890456	20791.31898	7.300212	368.516441	564.308654	10.379783	
1	3.716080	129.422921	18630.05786	6.635246	NaN	592.885359	15.180013	
2	8.099124	224.236259	19909.54173	9.275884	NaN	418.606213	16.868637	
3	8.316766	214.373394	22018.41744	8.059332	356.886136	363.266516	18.436525	
4	9.092223	181.101509	17978.98634	6.546600	310.135738	398.410813	11.558279	

Exploratory Data Analysis

In [3]:
```
df.shape
```

Out[3]:
```
(3276, 10)
```

Check for null values

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
ph                491
Hardness            0
Solids              0
Chloramines         0
Sulfate           781
Conductivity        0
Organic_carbon      0
Trihalomethanes   162
Turbidity           0
Potability          0
dtype: int64
```

In [5]:
```
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
```

In [6]:
```
df.describe()
```

Out[6]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_c
count	2785.000000	3276.000000	3276.000000	3276.000000	2495.000000	3276.000000	3276.0
mean	7.080795	196.369496	22014.092526	7.122277	333.775777	426.205111	14.2
std	1.594320	32.879761	8768.570828	1.583085	41.416840	80.824064	3.3
min	0.000000	47.432000	320.942611	0.352000	129.000000	181.483754	2.2
25%	6.093092	176.850538	15666.690300	6.127421	307.699498	365.734414	12.0
50%	7.036752	196.967627	20927.833605	7.130299	333.073546	421.884968	14.2
75%	8.062066	216.667456	27332.762125	8.114887	359.950170	481.792305	16.5
max	14.000000	323.124000	61227.196010	13.127000	481.030642	753.342620	28.3

Let's use the mean to handle the null values.

In [7]:
```
df.fillna(df.mean(),inplace=True)
df.isnull().sum()
```

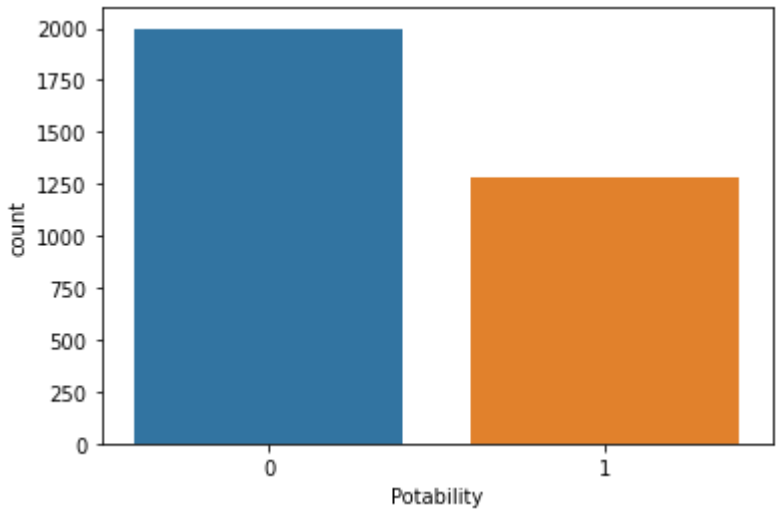
Out[7]:
```
ph                0
Hardness          0
Solids            0
Chloramines       0
Sulfate           0
Conductivity      0
Organic_carbon    0
Trihalomethanes   0
Turbidity         0
Potability        0
dtype: int64
```

In [8]:
```
df.Potability.value_counts()
```

Out[8]:
```
0    1998
1    1278
Name: Potability, dtype: int64
```

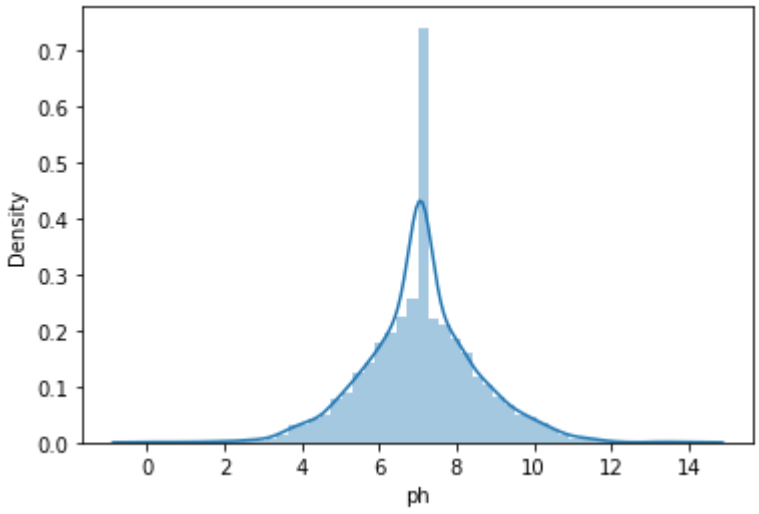
In [9]:
```
sns.countplot(df['Potability'])
plt.show()
```

```
C:\Users\hp\Anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarnin
g: Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without a
n explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



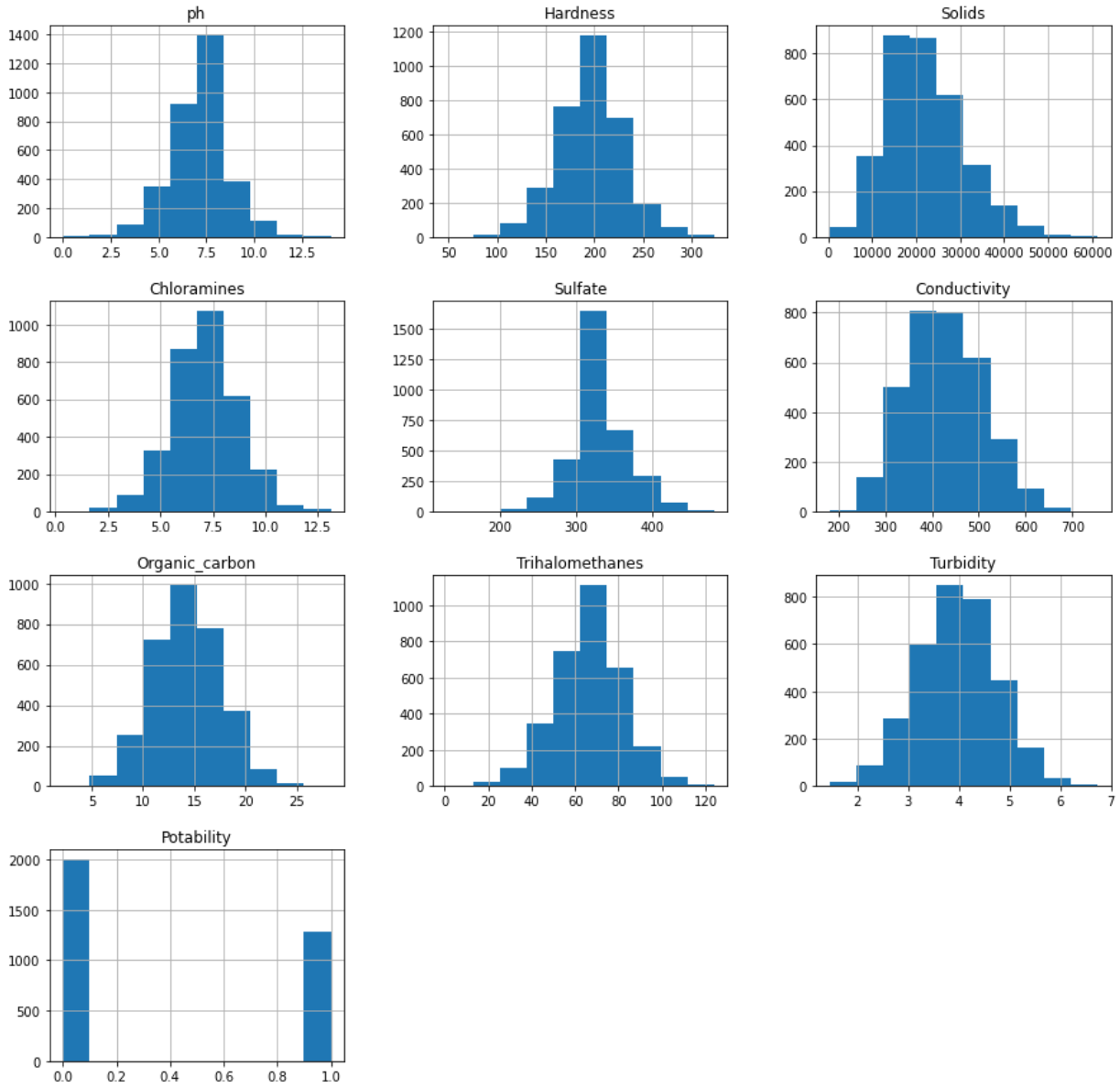
In [10]:
```
sns.distplot(df['ph'])
plt.show()
```

```
C:\Users\hp\Anaconda\lib\site-packages\seaborn\distributions.py:2619: FutureWar
ning: `distplot` is a deprecated function and will be removed in a future versi
on. Please adapt your code to use either `displot` (a figure-level function wit
h similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

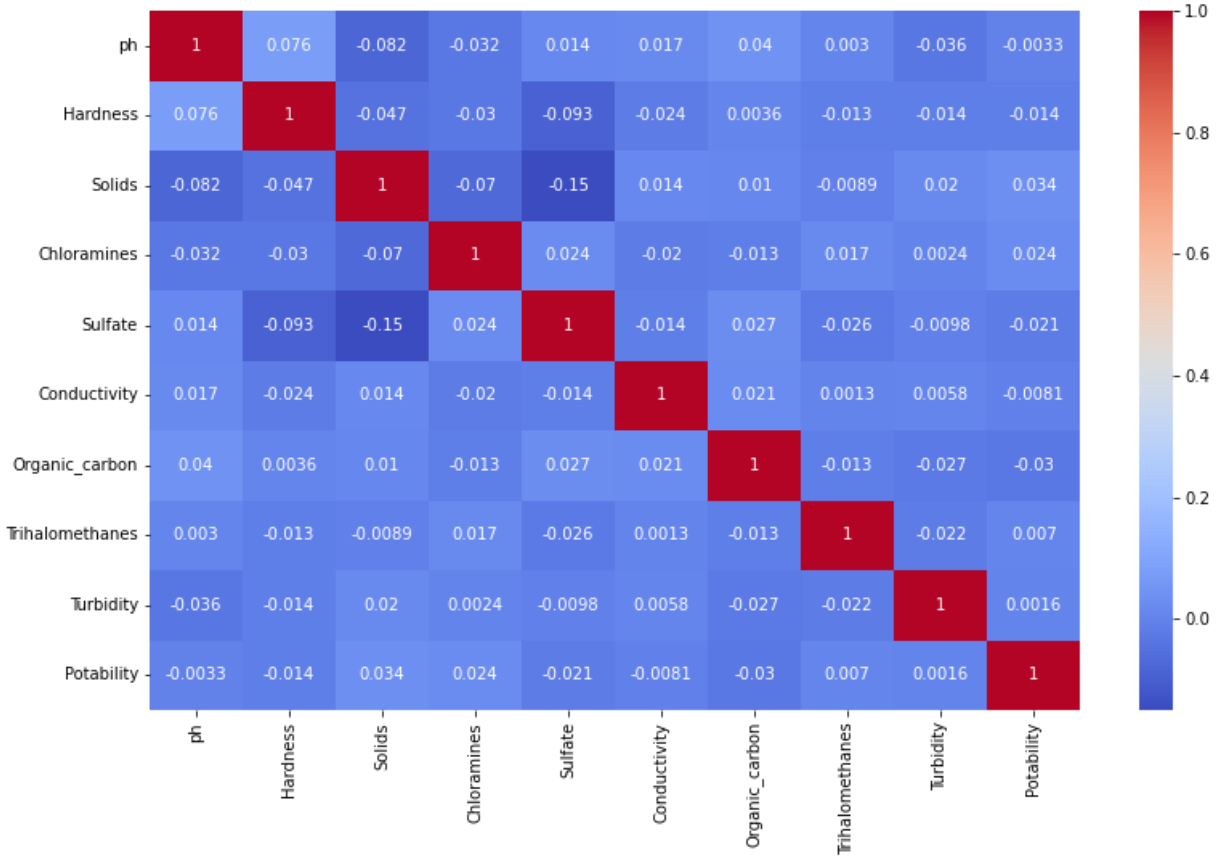


As we can see, the mode of the ph is clustered around the mean.

In [11]:
```
df.hist(figsize=(15,15))
plt.show()
```

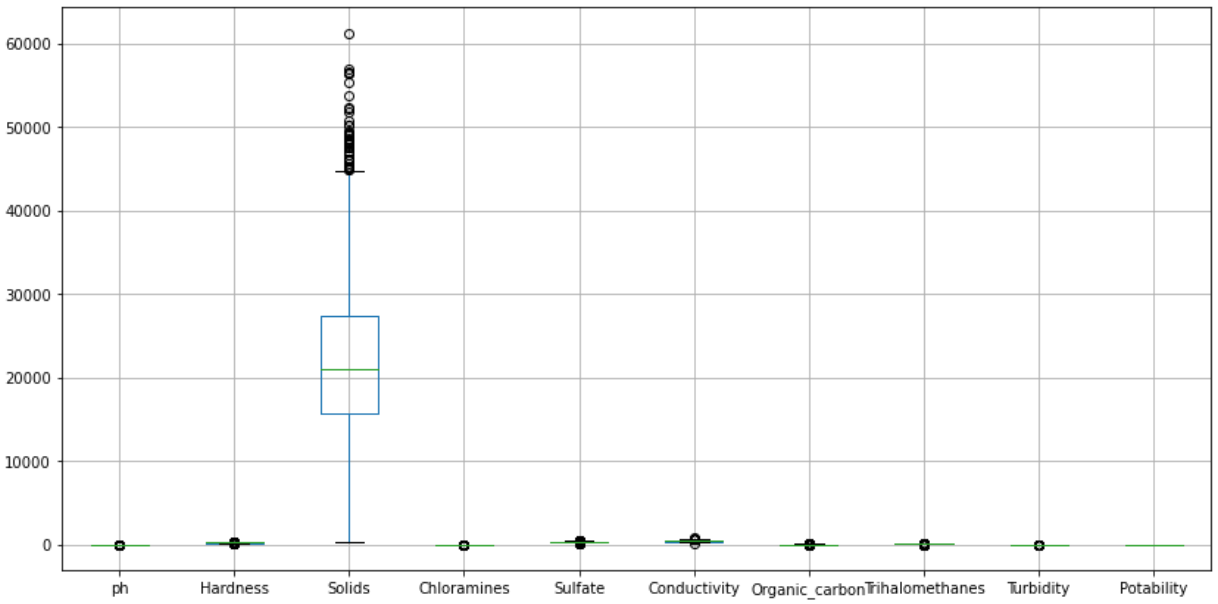


In [16]:
```
plt.figure(figsize=(13,8))
sns.heatmap(df.corr(),annot=True,cmap='coolwarm')
plt.show()
```



In [17]:
```
df.boxplot(figsize=(14,7))
```

Out[17]: <AxesSubplot:>



In [18]:
```
X = df.drop('Potability',axis=1)
Y= df['Potability']
```

In [26]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size= 0.2, random_s
```

Train Decision Tree Classifier and check accuracy

In [27]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
dt=DecisionTreeClassifier(criterion= 'gini', min_samples_split= 10, splitter= 'be
dt.fit(X_train,Y_train)
```

Out[27]:
```
DecisionTreeClassifier(min_samples_split=10)
```

In [28]:
```
prediction=dt.predict(X_test)
print(f"Accuracy Score = {accuracy_score(Y_test,prediction)*100}")
print(f"Confusion Matrix =\n {confusion_matrix(Y_test,prediction)}")
print(f"Classification Report =\n {classification_report(Y_test,prediction)}")
```

```
Accuracy Score = 62.04268292682927
Confusion Matrix =
 [[278 115]
 [134 129]]
Classification Report =
               precision    recall  f1-score   support

           0       0.67      0.71      0.69       393
           1       0.53      0.49      0.51       263

    accuracy                           0.62       656
   macro avg       0.60      0.60      0.60       656
weighted avg       0.62      0.62      0.62       656
```

Test with the test set.

In [34]:
```
res = dt.predict([[5.735724, 158.318741,25363.016594,7.728601,377.543291,568.3046
res
```

Out[34]:
```
0
```

Apply Hyper Parameter Tuning

In [35]:
```
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV

# define models and parameters
model = DecisionTreeClassifier()
criterion = ["gini", "entropy"]
splitter = ["best", "random"]
min_samples_split = [2,4,6,8,10,12,14]

# define grid search
grid = dict(splitter=splitter, criterion=criterion, min_samples_split=min_samples
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search_dt = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv,
                              scoring='accuracy',error_score=0)
grid_search_dt.fit(X_train, Y_train)
```

Out[35]:
```
GridSearchCV(cv=RepeatedStratifiedKFold(n_repeats=3, n_splits=10, random_state=
1),
             error_score=0, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'min_samples_split': [2, 4, 6, 8, 10, 12, 14],
                         'splitter': ['best', 'random']},
             scoring='accuracy')
```

In [36]:
```
print(f"Best: {grid_search_dt.best_score_:.3f} using {grid_search_dt.best_params_
means = grid_search_dt.cv_results_['mean_test_score']
stds = grid_search_dt.cv_results_['std_test_score']
params = grid_search_dt.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print(f"{mean:.3f} ({stdev:.3f}) with: {param}")

print("Training Score:",grid_search_dt.score(X_train, Y_train)*100)
print("Testing Score:", grid_search_dt.score(X_test, Y_test)*100)
```

```
Best: 0.597 using {'criterion': 'gini', 'min_samples_split': 14, 'splitter': 'r
andom'}
0.576 (0.029) with: {'criterion': 'gini', 'min_samples_split': 2, 'splitter':
'best'}
0.567 (0.026) with: {'criterion': 'gini', 'min_samples_split': 2, 'splitter':
'random'}
0.582 (0.030) with: {'criterion': 'gini', 'min_samples_split': 4, 'splitter':
'best'}
0.578 (0.031) with: {'criterion': 'gini', 'min_samples_split': 4, 'splitter':
'random'}
0.583 (0.029) with: {'criterion': 'gini', 'min_samples_split': 6, 'splitter':
'best'}
0.573 (0.030) with: {'criterion': 'gini', 'min_samples_split': 6, 'splitter':
'random'}
0.581 (0.029) with: {'criterion': 'gini', 'min_samples_split': 8, 'splitter':
'best'}
0.582 (0.033) with: {'criterion': 'gini', 'min_samples_split': 8, 'splitter':
'random'}
0.585 (0.029) with: {'criterion': 'gini', 'min_samples_split': 10, 'splitter':
'best'}
0.584 (0.028) with: {'criterion': 'gini', 'min_samples_split': 10, 'splitter':
'random'}
0.586 (0.031) with: {'criterion': 'gini', 'min_samples_split': 12, 'splitter':
'best'}
0.582 (0.027) with: {'criterion': 'gini', 'min_samples_split': 12, 'splitter':
'random'}
0.589 (0.033) with: {'criterion': 'gini', 'min_samples_split': 14, 'splitter':
'best'}
0.597 (0.025) with: {'criterion': 'gini', 'min_samples_split': 14, 'splitter':
'random'}
0.572 (0.031) with: {'criterion': 'entropy', 'min_samples_split': 2, 'splitte
r': 'best'}
0.573 (0.028) with: {'criterion': 'entropy', 'min_samples_split': 2, 'splitte
r': 'random'}
0.575 (0.032) with: {'criterion': 'entropy', 'min_samples_split': 4, 'splitte
r': 'best'}
0.581 (0.026) with: {'criterion': 'entropy', 'min_samples_split': 4, 'splitte
r': 'random'}
0.572 (0.030) with: {'criterion': 'entropy', 'min_samples_split': 6, 'splitte
r': 'best'}
0.590 (0.026) with: {'criterion': 'entropy', 'min_samples_split': 6, 'splitte
r': 'random'}
0.575 (0.034) with: {'criterion': 'entropy', 'min_samples_split': 8, 'splitte
r': 'best'}
0.579 (0.020) with: {'criterion': 'entropy', 'min_samples_split': 8, 'splitte
```

```
r': 'random'}
0.578 (0.029) with: {'criterion': 'entropy', 'min_samples_split': 10, 'splitte
r': 'best'}
0.581 (0.030) with: {'criterion': 'entropy', 'min_samples_split': 10, 'splitte
r': 'random'}
0.577 (0.022) with: {'criterion': 'entropy', 'min_samples_split': 12, 'splitte
r': 'best'}
0.596 (0.034) with: {'criterion': 'entropy', 'min_samples_split': 12, 'splitte
r': 'random'}
0.577 (0.022) with: {'criterion': 'entropy', 'min_samples_split': 14, 'splitte
r': 'best'}
0.590 (0.024) with: {'criterion': 'entropy', 'min_samples_split': 14, 'splitte
r': 'random'}
Training Score: 80.53435114503816
Testing Score: 61.4329268292683
```