```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
print('setup Completed^___^')
```

```
setup Completed^___^
```

Dataset will help to determine whether the water is potable or not

---

This dataset contains water quality measurements and assessments related to potability, which is the suitability of water for human consumption. The dataset's primary objective is to provide insights into water quality parameters and assist in determining whether the water is potable or not. Each row in the dataset represents a water sample with specific attributes, and the "Potability" column indicates whether the water is suitable for consumption.

Contents

---

pH: The pH level of the water.
Hardness: Water hardness, a measure of mineral content.
Solids: Total dissolved solids in the water.
Chloramines: Chloramines concentration in the water.
Sulfate: Sulfate concentration in the water.
Conductivity: Electrical conductivity of the water.
Organic_carbon: Organic carbon content in the water.
Trihalomethanes: Trihalomethanes concentration in the water.
Turbidity: Turbidity level, a measure of water clarity.
Potability: Target variable; indicates water potability with values 1 (potable) and 0 (not potable).

```
import warnings
warnings.filterwarnings('ignore')
```

```
##!mkdir ~/.kaggle
```

```
###!cp /kaggle.json ~/.kaggle/
```

```
###! pip install kaggle
```

```
###!chmod 600 /root/.kaggle/kaggle.json
```

```
###!kaggle datasets download -d uom190346a/water-quality-and-potability
```

```
##! unzip /content/water-quality-and-potability.zip
```

```
##! pip install --upgrade pandas
```

```
###! pip install --upgrade numpy
```

```
np.version.version
```

```
'1.25.2'
```

```
###! pip install seaborn
```

```
import numpy
import numpy.linalg
import numpy.linalg._umath_linalg
```

```
import pandas as pd
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt
import re
import nltk
from collections import Counter
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB,MultinomialNB
from sklearn.svm import SVC
from sklearn import metrics
```

```
plt.style.use('dark_background')
```

```
water_portability = pd.read_csv("/content/water_potability.csv")
```

```
from sklearn.model_selection import train_test_split
```

```
train, test = train_test_split(water_portability, test_size=0.33, random_state=42)
```

```
print(train.shape, test.shape)
```

```
(2194, 10) (1082, 10)
```

```
train.to_csv("/content/train.csv")
test.to_csv("/content/test.csv")
```

```
print("The Train Columns are :", train.columns)
print("The Test Columns are :", test.columns)
```

```
The Train Columns are : Index(['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity',
       'Organic_carbon', 'Trihalomethanes', 'Turbidity', 'Potability'],
      dtype='object')
The Test Columns are : Index(['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity',
       'Organic_carbon', 'Trihalomethanes', 'Turbidity', 'Potability'],
      dtype='object')
```

```
##! pip install transformers==4.15.0
```

```
###! pip install tensorflow==2.8.0
```

```
from transformers import DistilBertTokenizer
from transformers import TFDistilBertForSequenceClassification
import tensorflow as tf
import pandas as pd
import json
import gc
```

```
train.columns
```

```
Index(['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity',
       'Organic_carbon', 'Trihalomethanes', 'Turbidity', 'Potability'],
      dtype='object')
```

```
test.columns
```

```
Index(['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity',
       'Organic_carbon', 'Trihalomethanes', 'Turbidity', 'Potability'],
      dtype='object')
```

```
import tensorflow as tf
```

```
import pandas, numpy, string, textblob
import pickle
from sklearn import model_selection, preprocessing, linear_model, naive_bayes, metrics, svm, decomposition, ensemble
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
import xgboost
from keras import layers, models, optimizers
from keras.preprocessing import text, sequence
import matplotlib.pyplot as plt
```

```
##! pip install pycaret
```

```
###! pip install mlflow
```

```
#from pycaret.nlp import *
from pycaret.classification import *

import pandas as pd
```

```
print(test.shape, train.shape)
```

```
(1082, 10) (2194, 10)
```

```
train.columns
```

```
Index(['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity',
       'Organic_carbon', 'Trihalomethanes', 'Turbidity', 'Potability'],
      dtype='object')
```

```
from pycaret.classification import *
clf1 = setup(train, target = 'Potability', session_id=123, log_experiment=True)
```

	Description	Value
0	Session id	123
1	Target	Potability
2	Target type	Binary
3	Original data shape	(2194, 10)
4	Transformed data shape	(2194, 10)
5	Transformed train set shape	(1535, 10)
6	Transformed test set shape	(659, 10)
7	Numeric features	9
8	Rows with missing values	40.0%
9	Preprocess	True
10	Imputation type	simple
11	Numeric imputation	mean
12	Categorical imputation	mode
13	Fold Generator	StratifiedKFold
14	Fold Number	10
15	CPU Jobs	-1
16	Use GPU	False
17	Log Experiment	MlflowLogger
18	Experiment Name	clf-default-name
19	USI	690a

```
best_model = compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
qda	Quadratic Discriminant Analysis	0.6593	0.6556	0.3377	0.6415	0.4396	0.2285	0.2538	0.0450
et	Extra Trees Classifier	0.6437	0.6506	0.2919	0.6159	0.3946	0.1856	0.2123	0.2510
rf	Random Forest Classifier	0.6352	0.6406	0.3114	0.5889	0.4055	0.1759	0.1963	0.6780
gbc	Gradient Boosting Classifier	0.6169	0.5848	0.2512	0.5527	0.3438	0.1227	0.1435	0.6020
nb	Naive Bayes	0.6137	0.5683	0.2332	0.5390	0.3238	0.1102	0.1296	0.0860
lightgbm	Light Gradient Boosting Machine	0.6124	0.6183	0.3766	0.5225	0.4358	0.1528	0.1582	0.6460
xgboost	Extreme Gradient Boosting	0.6046	0.6031	0.4108	0.5128	0.4541	0.1500	0.1531	0.1200
lr	Logistic Regression	0.6019	0.5088	0.0081	0.3500	0.0158	0.0058	0.0230	0.7350
dummy	Dummy Classifier	0.6007	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0190
lda	Linear Discriminant Analysis	0.6006	0.4697	0.0147	0.3667	0.0278	0.0058	0.0156	0.0360
ridge	Ridge Classifier	0.6000	0.0000	0.0114	0.3167	0.0215	0.0032	0.0052	0.0520
ada	Ada Boost Classifier	0.5837	0.5343	0.2283	0.4593	0.3021	0.0526	0.0597	0.2280
svm	SVM - Linear Kernel	0.5784	0.0000	0.1032	0.0684	0.0628	-0.0026	-0.0052	0.0470
knn	K Neighbors Classifier	0.5713	0.5255	0.3377	0.4535	0.3862	0.0676	0.0699	0.0950
dt	Decision Tree Classifier	0.5596	0.5437	0.4648	0.4530	0.4575	0.0876	0.0878	0.1020

```
qda = create_model('qda')
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.6818	0.7398	0.3548	0.7097	0.4731	0.2798	0.3143
1	0.6039	0.5629	0.2742	0.5152	0.3579	0.1086	0.1199
2	0.6948	0.6809	0.4032	0.7143	0.5155	0.3170	0.3447
3	0.6558	0.6244	0.3443	0.6176	0.4421	0.2213	0.2411
4	0.6558	0.6617	0.3770	0.6053	0.4646	0.2307	0.2448
5	0.6732	0.6231	0.4426	0.6279	0.5192	0.2828	0.2927
6	0.6667	0.6842	0.2787	0.7083	0.4000	0.2257	0.2728
7	0.6667	0.6978	0.3443	0.6562	0.4516	0.2443	0.2705
8	0.6471	0.6153	0.2951	0.6207	0.4000	0.1925	0.2193
9	0.6471	0.6654	0.2623	0.6400	0.3721	0.1826	0.2178
Mean	0.6593	0.6556	0.3377	0.6415	0.4396	0.2285	0.2538

```
et = create_model('et')
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.6623	0.6730	0.3065	0.6786	0.4222	0.2291	0.2653
1	0.5909	0.5707	0.2742	0.4857	0.3505	0.0845	0.0919
2	0.6429	0.6876	0.3548	0.5946	0.4444	0.2053	0.2202
3	0.6688	0.6540	0.3115	0.6786	0.4270	0.2367	0.2723
4	0.6299	0.6765	0.2623	0.5714	0.3596	0.1469	0.1690
5	0.6667	0.6164	0.3443	0.6562	0.4516	0.2443	0.2705
6	0.6340	0.6852	0.2295	0.6087	0.3333	0.1471	0.1804
7	0.6863	0.6909	0.3443	0.7241	0.4667	0.2823	0.3214
8	0.5948	0.6210	0.2131	0.4815	0.2955	0.0673	0.0783
9	0.6601	0.6304	0.2787	0.6800	0.3953	0.2129	0.2539
Mean	0.6437	0.6506	0.2919	0.6159	0.3946	0.1856	0.2123
Std	0.0301	0.0377	0.0464	0.0793	0.0543	0.0675	0.0767

```
rf = create_model('rf', fold = 15)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7087	0.7415	0.3659	0.7895	0.5000	0.3315	0.3803
1	0.6505	0.6180	0.3902	0.5926	0.4706	0.2259	0.2369
2	0.6019	0.5921	0.2927	0.5000	0.3692	0.1066	0.1148
3	0.6699	0.6863	0.3902	0.6400	0.4848	0.2624	0.2798
4	0.7184	0.6892	0.3902	0.8000	0.5246	0.3567	0.4031
5	0.6078	0.6244	0.2683	0.5238	0.3548	0.1134	0.1265
6	0.6176	0.6421	0.2683	0.5500	0.3607	0.1318	0.1491
7	0.6176	0.5594	0.2683	0.5500	0.3607	0.1318	0.1491
8	0.6471	0.6150	0.2927	0.6316	0.4000	0.1951	0.2241
9	0.6961	0.7373	0.3415	0.7778	0.4746	0.3038	0.3548
10	0.6373	0.6052	0.2927	0.6000	0.3934	0.1763	0.1995
11	0.6863	0.7607	0.3902	0.6957	0.5000	0.2969	0.3232
12	0.5882	0.5882	0.2439	0.4762	0.3226	0.0691	0.0771
13	0.5882	0.5591	0.2250	0.4500	0.3000	0.0522	0.0585
14	0.6373	0.6204	0.2750	0.5789	0.3729	0.1610	0.1830
Mean	0.6449	0.6426	0.3130	0.6104	0.4126	0.1943	0.2173
Std	0.0414	0.0631	0.0570	0.1086	0.0701	0.0940	0.1064

```
models()
```

ID	Name	Reference	Turbo
lr	Logistic Regression	sklearn.linear_model._logistic.LogisticRegression	True
knn	K Neighbors Classifier	sklearn.neighbors._classification.KNeighborsCl...	True
nb	Naive Bayes	sklearn.naive_bayes.GaussianNB	True
dt	Decision Tree Classifier	sklearn.tree._classes.DecisionTreeClassifier	True
svm	SVM - Linear Kernel	sklearn.linear_model._stochastic_gradient.SGDC...	True
rbfsvm	SVM - Radial Kernel	sklearn.svm._classes.SVC	False
gpc	Gaussian Process Classifier	sklearn.gaussian_process._gpc.GaussianProcessC...	False
mlp	MLP Classifier	sklearn.neural_network._multilayer_perceptron....	False
ridge	Ridge Classifier	sklearn.linear_model._ridge.RidgeClassifier	True
rf	Random Forest Classifier	sklearn.ensemble._forest.RandomForestClassifier	True
qda	Quadratic Discriminant Analysis	sklearn.discriminant_analysis.QuadraticDiscrim...	True
ada	Ada Boost Classifier	sklearn.ensemble._weight_boosting.AdaBoostClas...	True
gbc	Gradient Boosting Classifier	sklearn.ensemble._gb.GradientBoostingClassifier	True

```
models(type='ensemble').index.tolist()
```

```
['rf', 'ada', 'gbc', 'et', 'xgboost', 'lightgbm']
```

```
tuned_rf = tune_model(rf)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.5909	0.5376	0.1774	0.4783	0.2588	0.0524	0.0646
1	0.6169	0.5806	0.2903	0.5455	0.3789	0.1378	0.1521
2	0.6299	0.5657	0.2581	0.5926	0.3596	0.1525	0.1786
3	0.6299	0.5457	0.2131	0.5909	0.3133	0.1307	0.1626
4	0.6039	0.6097	0.1475	0.5000	0.2278	0.0578	0.0773
5	0.6209	0.4849	0.2459	0.5556	0.3409	0.1274	0.1483
6	0.6667	0.6410	0.2951	0.6923	0.4138	0.2304	0.2713
7	0.6405	0.5567	0.2951	0.6000	0.3956	0.1801	0.2030
8	0.6013	0.5211	0.1639	0.5000	0.2469	0.0623	0.0802
9	0.5882	0.5249	0.2295	0.4667	0.3077	0.0608	0.0686
Mean	0.6189	0.5568	0.2316	0.5522	0.3243	0.1192	0.1407
Std	0.0229	0.0430	0.0523	0.0658	0.0613	0.0571	0.0646

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Original model was better than the tuned model, hence it will be returned. NOTE: The di
```

```
bagged_rf = ensemble_model(rf)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.6364	0.7012	0.2581	0.6154	0.3636	0.1650	0.1956
1	0.6039	0.5871	0.2903	0.5143	0.3711	0.1136	0.1235
2	0.6818	0.6856	0.3710	0.6970	0.4842	0.2839	0.3135
3	0.6234	0.6236	0.2295	0.5600	0.3256	0.1238	0.1475
4	0.6299	0.6508	0.2459	0.5769	0.3448	0.1416	0.1666
5	0.6340	0.5802	0.3115	0.5758	0.4043	0.1727	0.1896
6	0.6601	0.7063	0.2459	0.7143	0.3659	0.2031	0.2571
7	0.6928	0.6904	0.3115	0.7917	0.4471	0.2864	0.3462
8	0.6013	0.5912	0.1475	0.5000	0.2278	0.0564	0.0756
9	0.6863	0.6144	0.2623	0.8421	0.4000	0.2598	0.3410
Mean	0.6450	0.6431	0.2673	0.6387	0.3734	0.1806	0.2156
Std	0.0317	0.0474	0.0565	0.1110	0.0662	0.0732	0.0896

```
evaluate_model(tuned_rf)
```

Plot Type:

Pipeline Plot	Hyperparameters	AUC	Confusion Matrix
Threshold	Precision Recall	Prediction Error	Class Report
Feature Selection	Learning Curve	Manifold Learning	Calibration Curve
Validation Curve	Dimensions	Feature Importance	Feature Importance…
Decision Boundary	Lift Chart	Gain Chart	Decision Tree
KS Statistic Plot			


```
pred_holdouts_rf = predict_model(tuned_rf)
pred_holdouts_rf.head(4)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Random Forest Classifier	0.6495	0.6565	0.3460	0.6067	0.4407	0.2123	0.2301

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_
517	NaN	208.177460	17264.841797	3.296157	387.070831	631.304199	8
3128	4.959853	215.854874	9887.831055	6.954231	379.504730	527.479675	14
1764	NaN	220.985153	26492.716797	4.949788	NaN	388.969025	21
198	5.554437	229.122833	31344.460938	7.761431	NaN	306.872528	13

```
final_rf_cl = finalize_model(tuned_rf)
```

```
prediction_rf = predict_model(final_rf_cl, data = test)
prediction_rf.tail()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Random Forest Classifier	0.6701	0.6735	0.3408	0.5983	0.4342	0.2253	0.2431

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_
1662	6.006770	226.874100	20279.701172	8.166416	225.516632	275.986603	9
445	6.728004	201.126892	22888.787109	7.663988	319.463501	325.537537	16
617	6.284985	196.775055	29213.621094	8.528792	334.477783	574.540649	11
1474	5.821261	204.048889	37174.003906	7.867815	329.019562	466.783264	13
2555	5.681811	151.085938	26373.496094	5.651589	NaN	468.472595	8

```
prediction_rf.columns
```

```
Index(['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity',
       'Organic_carbon', 'Trihalomethanes', 'Turbidity', 'Potability',
       'prediction_label', 'prediction_score'],
      dtype='object')
```

```
prediction_rf["prediction_label"].value_counts()
```

```
0    853
1    229
Name: prediction_label, dtype: int64
```

```
prediction_rf.to_csv("prediction.csv")
```

```
prediction_rf.head(4)
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_
2947	NaN	183.521103	20461.251953	7.333212	333.119476	356.369019	20
2782	6.643159	188.913544	32873.820312	6.791509	333.848846	336.561493	14
1644	7.846058	224.058884	23264.109375	5.922367	300.402618	387.971344	13
70	7.160467	183.089310	6743.346191	3.803036	277.599091	428.036346	9

```
prediction_rf.columns
```

```
Index(['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity',
       'Organic_carbon', 'Trihalomethanes', 'Turbidity', 'Potability',
       'prediction_label', 'prediction_score'],
      dtype='object')
```

```
prediction_rf.shape
```

```
(1082, 12)
```

```
prediction_rf["prediction_label"].value_counts()
```

```
0    853
1    229
Name: prediction_label, dtype: int64
```

Deep Learning

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
# Importing all datasets
water_potability = pd.read_csv("/content/water_potability.csv")
water_potability.head(4)
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_car
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.868
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436

```
water_potability.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
```

```
water_potability.isnull().sum()
```

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

```
water_potability = water_potability.fillna(0)
```

```
water_potability.isnull().sum()
```

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

```
water_potability.columns
```

```
Index(['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity',
       'Organic_carbon', 'Trihalomethanes', 'Turbidity', 'Potability'],
      dtype='object')
```

```
# Checking for outliers in the continuous variables
num_water_potability = water_potability[['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity',
       'Organic_carbon', 'Trihalomethanes', 'Turbidity', 'Potability']]
```

```
# Checking outliers at 25%, 50%, 75%, 90%, 95% and 99%
num_water_potability.describe(percentiles=[.25, .5, .75, .90, .95, .99])
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Or
count	3276.000000	3276.000000	3276.000000	3276.000000	3276.000000	3276.000000	
mean	6.019540	196.369496	22014.092526	7.122277	254.203468	426.205111	
std	2.924207	32.879761	8768.570828	1.583085	146.765192	80.824064	
min	0.000000	47.432000	320.942611	0.352000	0.000000	181.483754	
25%	5.283146	176.850538	15666.690297	6.127421	240.722848	365.734414	
50%	6.735249	196.967627	20927.833607	7.130299	318.660382	421.884968	
75%	7.870050	216.667456	27332.762127	8.114887	350.385756	481.792304	
90%	8.925047	236.350707	33814.935230	9.122578	378.478321	533.297241	
95%	9.616936	249.609769	38474.990249	9.753101	395.554101	566.349320	
99%	10.812925	278.062602	45974.106490	10.967153	429.028139	617.626558	
max	14.000000	323.124000	61227.196008	13.127000	481.030642	753.342620	

```
Q1 = water_potability.quantile(0.25)
Q3 = water_potability.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
ph                    2.586904
Hardness             39.816918
Solids            11666.071830
Chloramines           1.987466
Sulfate             109.662908
Conductivity        116.057890
Organic_carbon        4.491850
Trihalomethanes      22.872921
Turbidity             1.060609
Potability            1.000000
dtype: float64
```

```
print(water_potability.quantile(0.05))
print(water_potability.quantile(0.95))
```

```
ph                   0.000000
Hardness           141.763281
Solids            9545.812579
Chloramines          4.503054
Sulfate              0.000000
Conductivity       300.109466
Organic_carbon       8.815362
Trihalomethanes      8.476729
Turbidity            2.684279
Potability           0.000000
Name: 0.05, dtype: float64
ph                    9.616936
Hardness            249.609769
Solids            38474.990249
Chloramines           9.753101
Sulfate             395.554101
Conductivity        566.349320
Organic_carbon       19.637254
Trihalomethanes      91.744595
Turbidity             5.220925
Potability            1.000000
Name: 0.95, dtype: float64
```

```
water_potability = water_potability[~((water_potability < (Q1 - 1.5 * IQR)) | (water_potability > (Q3 + 1.5 * IQR))).any(axis=1)
print(water_potability.shape)
```

```
(1848, 10)
```

```
from sklearn.model_selection import train_test_split
```

```
X = water_potability.drop("Potability", axis=1)
y = water_potability["Potability"]
```

```
  # Splitting the data into training and validation sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(1293, 9) (1293,)
(555, 9) (555,)
```

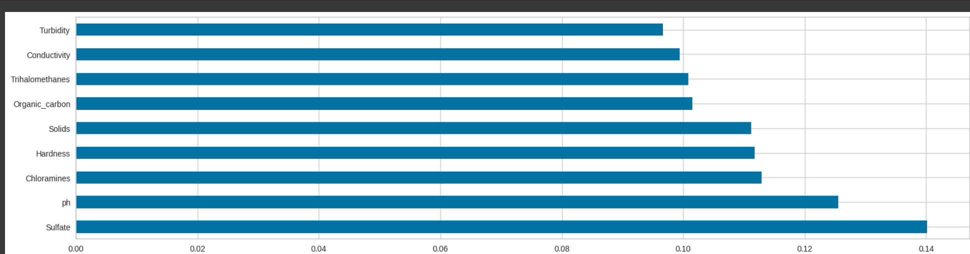
```
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt
model=ExtraTreesClassifier()
model.fit(X_train,y_train)
```

```
▾ ExtraTreesClassifier
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='sqrt',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_samples_leaf=1,
                     min_samples_split=2, min_weight_fraction_leaf=0.0,
                     n_estimators=100, n_jobs=None, oob_score=False,
                     random_state=None, verbose=0, warm_start=False)
```

```
print(model.feature_importances_)
```

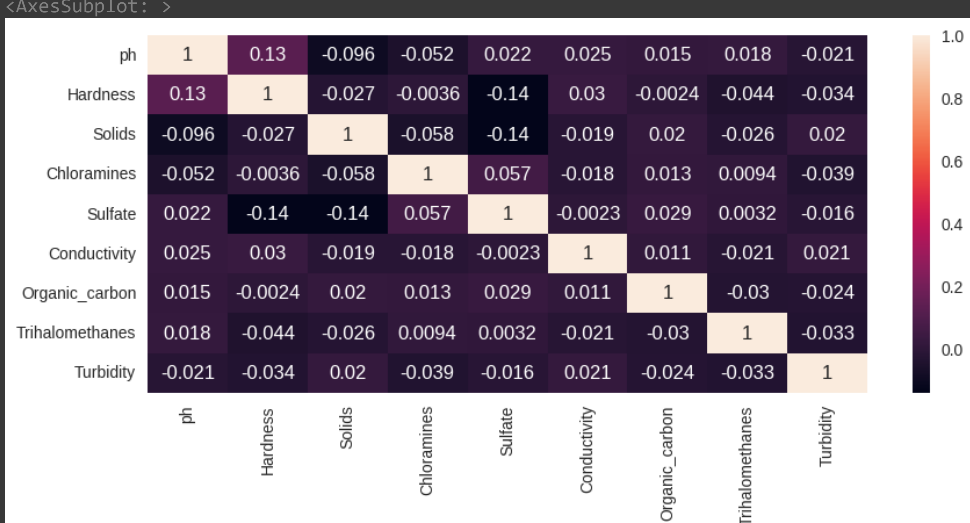
```
[0.12549512 0.11179293 0.11122817 0.11291422 0.14017545 0.09942991
 0.10153381 0.10081819 0.09661221]
```

```
plt.figure(figsize=(20,5))
ranked_features=pd.Series(model.feature_importances_,index=X.columns)
ranked_features.nlargest(10).plot(kind='barh')
plt.show()
```




```
import seaborn as sns
corr=X_train.corr()
top_features=corr.index
plt.figure(figsize=(10,4))
sns.heatmap(X_train[top_features].corr(),annot=True)
```

```
<AxesSubplot: >
```




```
y_train.value_counts()
```

```
0    787
1    506
Name: Potability, dtype: int64
```

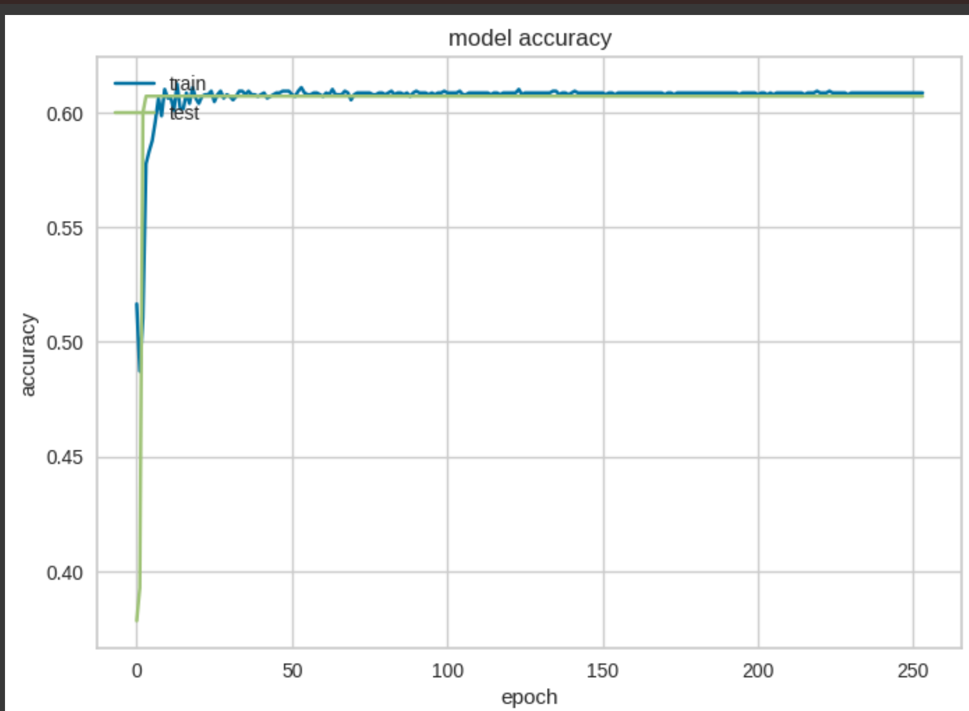
```
from keras.models import Sequential
from keras.layers import InputLayer
from keras.layers import Dense
from keras.layers import Dropout
from keras.constraints import maxnorm

from  IPython import display
```

```
import tensorflow as tf
```

```
nn_model = Sequential()
nn_model.add(Dense(64,kernel_regularizer=tf.keras.regularizers.l2(0.001),
                input_dim=9, activation='relu' ))
nn_model.add(Dropout(rate=0.2))
nn_model.add(Dense(8,kernel_regularizer=tf.keras.regularizers.l2(0.001),
                    activation='relu'))
nn_model.add(Dropout(rate=0.1))
nn_model.add(Dense(1, activation='sigmoid'))
lr_schedule = tf.keras.optimizers.schedules.InverseTimeDecay(
      0.001,
      decay_steps=(X_train.shape[0]/32)*50,
      decay_rate=1,
      staircase=False)

def get_optimizer():
    return tf.keras.optimizers.Adam(lr_schedule)
def get_callbacks():
    return [tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',patience=250,restore_best_weights=True)]
nn_model.compile(loss = "binary_crossentropy",
                  optimizer = get_optimizer(),
                  metrics=['accuracy'])
```

```
history = nn_model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=300, batch_size=32,
                    callbacks= get_callbacks(),verbose=0)
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```




```
yprednn=nn_model.predict(X_test)
yprednn=yprednn.round()
```

```
from sklearn import metrics
```

```
print('Neural Network Classification Report :\n {}\n'.format(
    metrics.classification_report(yprednn, y_test)))
nn_conf_matrix=metrics.confusion_matrix(yprednn,y_test)
conf_mat_nn = pd.DataFrame(
    nn_conf_matrix,
    columns=["Predicted NO", "Predicted YES"],
    index=["Actual NO", "Actual YES"]
)
print('Neural Network Confusion Matrix :\n')
print(conf_mat_nn)
```

```
     Neural Network Classification Report :
                   precision    recall  f1-score   support

              0.0       1.00      0.61      0.76       555
              1.0       0.00      0.00      0.00         0

         accuracy                           0.61       555
        macro avg       0.50      0.30      0.38       555
```

```
weighted avg       1.00      0.61      0.76       555


Neural Network Confusion Matrix :

            Predicted NO  Predicted YES
Actual NO            337            218
Actual YES             0              0
```

Explainable AI

```
###! pip install shap
```

```
###! pip install interpret
```

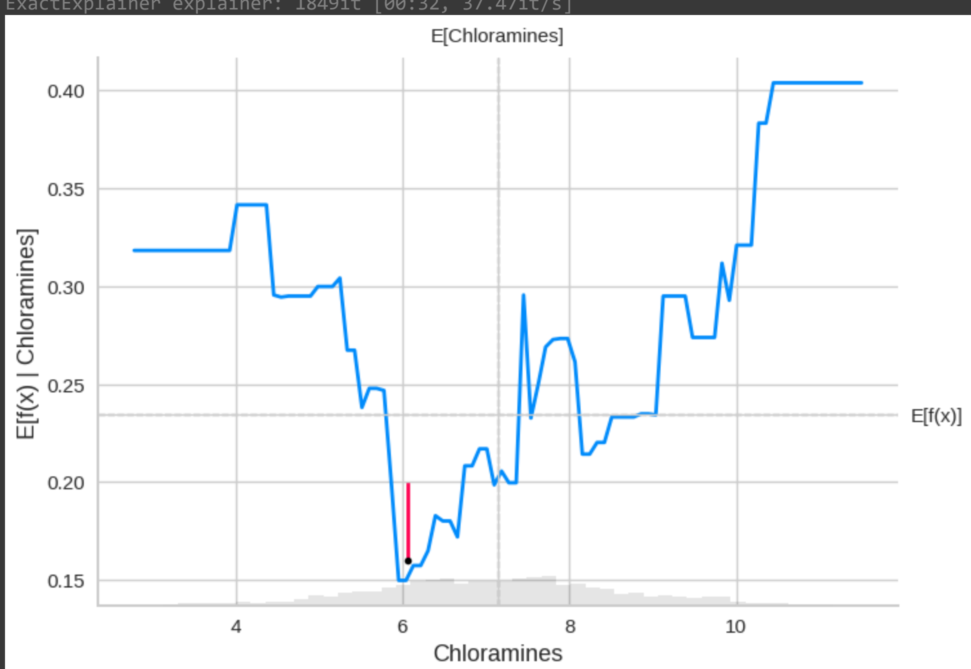
```
import shap
```

```
print(X.shape, y.shape)
```

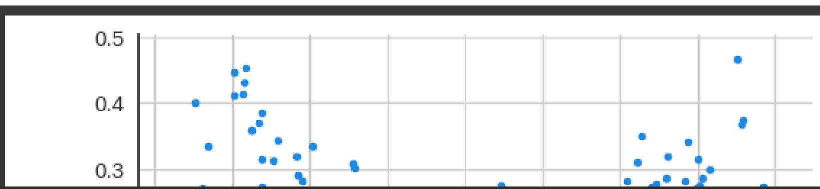
```
(1848, 9) (1848,)
```

```
# fit a GAM model to the data
import interpret.glassbox
model_ebm = interpret.glassbox.ExplainableBoostingClassifier()
model_ebm.fit(X, y)
sample_ind = 18

# explain the GAM model with SHAP
explainer_ebm = shap.Explainer(model_ebm.predict, X)
shap_values_ebm = explainer_ebm(X)

# make a standard partial dependence plot with a single SHAP value overlaid
fig,ax = shap.partial_dependence_plot(
    "Chloramines", model_ebm.predict, X, model_expected_value=True,
    feature_expected_value=True, show=False, ice=False,
    shap_values=shap_values_ebm[sample_ind:sample_ind+1,:]
)
```

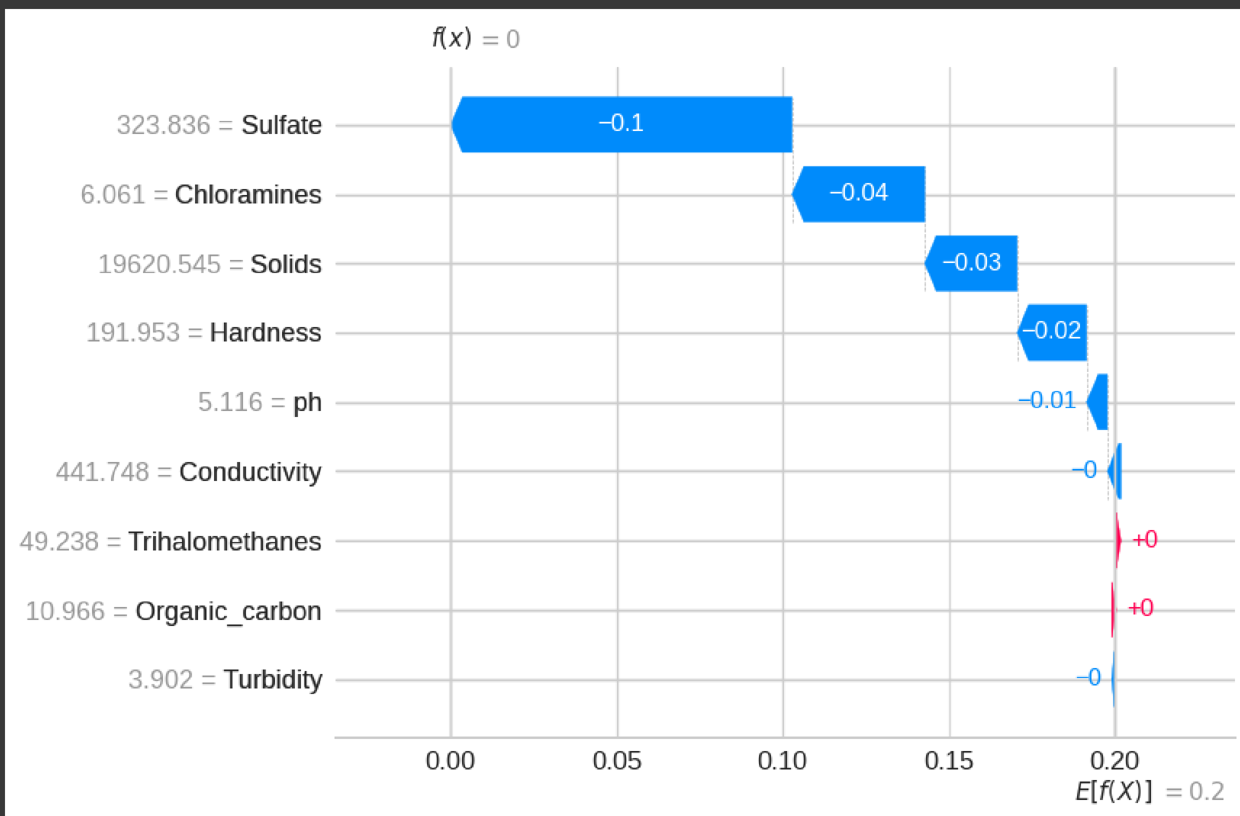
```
ExactExplainer explainer: 1849it [00:32, 37.47it/s]
```



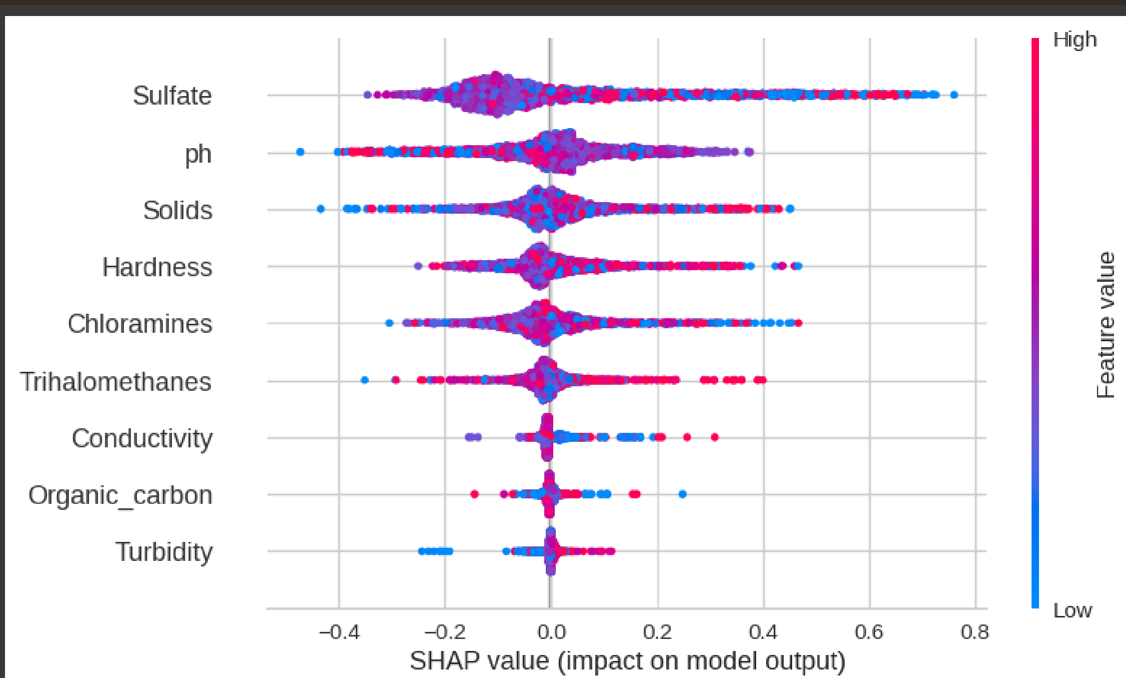

```
shap.plots.scatter(shap_values_ebm[:,"Chloramines"])
```




```
# the waterfall_plot shows how we get from explainer.expected_value to model.predict(X)[sample_ind]
shap.plots.waterfall(shap_values_ebm[sample_ind], max_display=14)
```




```
# the waterfall_plot shows how we get from explainer.expected_value to model.predict(X)[sample_ind]
shap.plots.beeswarm(shap_values_ebm, max_display=14)
```



Start coding or generate with AI.