water managment

```
# import packages
import pandas as pd
import numpy as np

from sklearn.utils import resample
from sklearn.model_selection import train_test_split

from sklearn.impute import KNNImputer
from sklearn.preprocessing import PowerTransformer
from sklearn.decomposition import PCA
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
data= pd.read_csv('/content/waterQuality1.csv')
```

EDA

```
data.head()
```

	aluminium	ammonia	arsenic	barium	cadmium	chloramine	chromium	copper	flouride	bacteria	...	lead	nitrates	nitrites	mer
0	1.65	9.08	0.04	2.85	0.007	0.35	0.83	0.17	0.05	0.20	...	0.054	16.08	1.13	
1	2.32	21.16	0.01	3.31	0.002	5.28	0.68	0.66	0.90	0.65	...	0.100	2.01	1.93	
2	1.01	14.02	0.04	0.58	0.008	4.24	0.53	0.02	0.99	0.05	...	0.078	14.16	1.11	
3	1.36	11.33	0.04	2.96	0.001	7.23	0.03	1.66	1.08	0.71	...	0.016	1.41	1.29	
4	0.92	24.33	0.03	0.20	0.006	2.67	0.69	0.57	0.61	0.13	...	0.117	6.74	1.11	

5 rows × 21 columns

```
data.shape
```

```
(7999, 21)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7999 entries, 0 to 7998
Data columns (total 21 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   aluminium    7999 non-null   float64
 1   ammonia      7999 non-null   object
 2   arsenic      7999 non-null   float64
 3   barium       7999 non-null   float64
 4   cadmium      7999 non-null   float64
 5   chloramine   7999 non-null   float64
 6   chromium     7999 non-null   float64
 7   copper       7999 non-null   float64
 8   flouride     7999 non-null   float64
 9   bacteria     7999 non-null   float64
 10  viruses      7999 non-null   float64
 11  lead         7999 non-null   float64
 12  nitrates     7999 non-null   float64
 13  nitrites     7999 non-null   float64
 14  mercury      7999 non-null   float64
 15  perchlorate  7999 non-null   float64
 16  radium       7999 non-null   float64
 17  selenium     7999 non-null   float64
 18  silver       7999 non-null   float64
 19  uranium      7999 non-null   float64
 20  is_safe      7999 non-null   object
dtypes: float64(19), object(2)
memory usage: 1.3+ MB
```

```
data.isna().sum()
```

	0
aluminium	0
ammonia	0
arsenic	0
barium	0
cadmium	0
chloramine	0
chromium	0
copper	0
flouride	0
bacteria	0
viruses	0
lead	0
nitrates	0
nitrites	0
mercury	0
perchlorate	0
radium	0
selenium	0
silver	0
uranium	0
is_safe	0

dtype: int64

```
data.loc[data['ammonia'] == '#NUM!', 'ammonia']= data['ammonia'].mode()[0]
```

```
data['ammonia']= data['ammonia'].astype(float)
```

```
data.loc[data['is_safe'] == '#NUM!', 'is_safe']= data['is_safe'].mode()[0]
data['is_safe']= data['is_safe'].astype(int)
```

```
data['is_safe'].value_counts()
```

is_safe	count
0	7087
1	912

dtype: int64

```
majority_data= data[ data['is_safe'] == 0 ]
minority_data= data[ data['is_safe'] == 1 ]
```

```
majority_data.shape, minority_data.shape
```

```
((7087, 21), (912, 21))
```

```
upsampled_minority_data = resample(minority_data, replace=True, n_samples=len(majority_data), random_state=42)
upsampled_minority_data = upsampled_minority_data.reset_index(drop=True)
majority_data = majority_data.reset_index(drop=True)
balanced_data = pd.concat([majority_data, upsampled_minority_data]).reset_index(drop=True)
```

```
balanced_data= balanced_data.sample(frac=1)
```

```
balanced_data.shape
```

```
(14174, 21)
```

```
balanced_data['is_safe'].value_counts()
```

is_safe	count
1	7087
0	7087

dtype: int64

```
balanced_data.head()
```

	aluminium	ammonia	arsenic	barium	cadmium	chloramine	chromium	copper	flouride	bacteria	...	lead	nitrates	nitrites
13282	0.08	28.10	0.01	0.42	0.010	0.13	0.07	0.51	0.07	0.00	...	0.075	7.64	0.71
4814	0.05	14.99	0.04	0.29	0.020	0.25	0.07	0.54	1.20	0.59	...	0.047	19.37	0.97
4526	0.09	4.16	0.06	0.33	0.040	0.03	0.02	1.32	0.76	0.00	...	0.128	14.79	0.26
10915	2.93	15.34	0.01	2.39	0.008	1.65	0.25	1.12	1.15	0.00	...	0.121	2.47	1.38
6682	0.09	0.21	0.01	1.22	0.030	0.39	0.03	1.83	1.30	0.00	...	0.129	8.07	1.23

5 rows × 21 columns

```
balanced_data.shape
```

```
(14174, 21)
```

```
balanced_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 14174 entries, 13282 to 5415
Data columns (total 21 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   aluminium    14174 non-null  float64
 1   ammonia      14174 non-null  float64
 2   arsenic      14174 non-null  float64
 3   barium       14174 non-null  float64
 4   cadmium      14174 non-null  float64
 5   chloramine   14174 non-null  float64
 6   chromium     14174 non-null  float64
 7   copper       14174 non-null  float64
 8   flouride     14174 non-null  float64
 9   bacteria     14174 non-null  float64
 10  viruses      14174 non-null  float64
 11  lead         14174 non-null  float64
 12  nitrates     14174 non-null  float64
 13  nitrites     14174 non-null  float64
 14  mercury      14174 non-null  float64
 15  perchlorate  14174 non-null  float64
 16  radium       14174 non-null  float64
 17  selenium     14174 non-null  float64
 18  silver       14174 non-null  float64
 19  uranium      14174 non-null  float64
 20  is_safe      14174 non-null  int64
dtypes: float64(20), int64(1)
memory usage: 2.4 MB
```

```
balanced_data.isna().sum()
```

	0
aluminium	0
ammonia	0
arsenic	0
barium	0
cadmium	0
chloramine	0
chromium	0
copper	0
flouride	0
bacteria	0
viruses	0
lead	0
nitrates	0
nitrites	0
mercury	0
perchlorate	0
radium	0
selenium	0
silver	0
uranium	0
is_safe	0

dtype: int64

```
X= balanced_data.drop('is_safe', axis= 1)

Y= balanced_data['is_safe']


X_train, X_test, y_train, y_test= train_test_split(X, Y, test_size=0.2)


# numeric transformer := imputer + power transformer
numeric_transformer= Pipeline(
    [
        ('numeric_imputer', KNNImputer(weights='distance')),
        ('power_transformer', PowerTransformer())
    ]
)


# preprocessor
preprocessor= ColumnTransformer([
    ('numeric_transformer', numeric_transformer, slice(0, 20))
])


# model
model= LogisticRegression()



# pipeline
pipe= Pipeline([
    ('preprocessor', preprocessor),
    ('PCA', PCA(n_components=17)),
    ('model', model),
])


# fit it
pipe.fit(X_train, y_train)
```

```
Pipeline
preprocessor: ColumnTransformer
numeric_transformer
KNNImputer
PowerTransformer
PCA
LogisticRegression
```

```
# predict
y_preds= pipe.predict(X_test)
```

evaluate

```
accuracy_score(y_test, y_preds)
```

```
0.799647266313933
```

```
precision_score(y_test, y_preds)
```

```
0.7978102189781022
```

```
recall_score(y_test, y_preds)
```

```
0.7897398843930635
```

```
f1_score(y_test, y_preds)
```

```
0.7937545388525781
```

```
# find the best params for PCA
max_acc= 0
best_comps= 0

for i in range(1, 20):

    pipe= Pipeline([
        ('preprocessor', preprocessor),
        ('PCA', PCA(n_components=i)),
        ('model', model),
    ])

    pipe.fit(X_train, y_train)

    y_preds= pipe.predict(X_test)

    acc= accuracy_score(y_test, y_preds)

    if acc > max_acc:
        max_acc= acc
        best_comps= i

    print(f"{i} comps := {acc}")

print(f"\n\nBest comps: {best_comps}, Accuracy: {max_acc}")
```

```
1 comps := 0.6825396825396826
2 comps := 0.7509700176366843
3 comps := 0.7407407407407407
4 comps := 0.7407407407407407
5 comps := 0.7548500881834215
6 comps := 0.7597883597883598
7 comps := 0.7611992945326279
8 comps := 0.7679012345679013
9 comps := 0.762962962962963
10 comps := 0.76331569664903
11 comps := 0.7693121693121693
12 comps := 0.7735449735449735
```

```
13 comps := 0.7746031746031746
14 comps := 0.7728395061728395
15 comps := 0.7774250440917108
```